Using Reinforcement Learning to Enhance a Knowledge Management System for Web Recommendation Based on Self-Organization of Communities

Achbany Youssef

January 2006

Contents

Acknowledgements

First of all I would like to thank my parents for all that they have given me and in particular for the education which they provided me with. I am also grateful to my family and friends who provided the support and motivation that made it possible for me to start the DEA in the first place.

Next, I would like to thank my supervisors Marco Searens and Manuel Kolp for their help. They always found time for me in their busy schedule and provide numerous helpful comments for my research and other publications.

Very special thanks go to Stephane Faulkner, who pointed the way to the relevant literature and gave me some very useful comments.
For discussions and the friendly atmosphere provided by the IAG department, special thanks to Yves Wautelet, Adrien Coyette, François Fouss, Benoît Collignon, Luh Yen, Laurent Bouillon and the other members of the IAG departement.

All the other people who helped me along the way that I have not mentioned, my thanks and apologies.

Chapter 1

Introduction

The use of Internet as principal source of information was quickly spread, in particular thanks to its impressive quantity of information placed at the disposal of the users. The consequence of this situation consists in a constantly increasing number of hosts and information on the Internet. The pages become appreciably bigger, returning the search for required information more complex. A survey on user search behaviour data [24] show that user rarely goes further than the second page of results provided by a search engine. It means that web search tools should improve quality of the results on the first two pages.

An important issue currently not addressed by most of search engines is the consideration of user profiles during the searching process. Indeed, most of search engines base their searching process from the user keywords, without regard to the meaning of theses keywords for the users (i.e., the context where theses keywords are significant for the users). Yet, this could lead to a high score for a page simply because it contains the given keywords. However this page could be valueless for the user because of the context. The difficulty for users to find what they are looking for, among the provided selection of links, is especially raised when searched words are overloaded with multiple meanings. The context must be taken into consideration to determine a narrower range of search results. Unfortunately, since users typically do not or cannot provide such a context, most of search engines can only rely on the occurrence of user-provided keywords.

A solution to cope with these issues (i.e., quantity of information and consideration of the user profile) is to filter the information on the basis of experiment held by a community of people (whence the collaborative filtering term) sharing the same interest. We define the concept of people community to gather experiences having the same semantic (i.e., experiences that can be beneficial for every community members) together.

The system will have to use mechanisms of self-organization to allow the management of the communities in an autonomous way. Indeed, according to the experiment of the users (i.e., their profile) the system will have to self-organize them in community where they will be able efficiently to manage and share their experiment. This concept of self-organization is described in detail further in this document.

So, to find information of quality, a recommendation system has to combine the self-management of communities and the use of collaborative-filtering methods on the basis of the community experiences and the user profile.

A complementary approach which would allow the design of a recommendation system even more powerful would be the use of Reinforcement Learning. This learning method addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goal. In the continuation, we will present in detail the Reinforcement Learning and we will think of the contribution of such a method for our recommendation system.

In addition to the introduction an the conclusion, this dissertation is organized in 5 chapters.

Chapter 2 introduce some major concepts to be used in the dissertation. Amongst all, we introduce the concept of Knowledge and Knowledge Management, the meaning and the mechanisms of Self-Organization system and the agent paradigm.

Chapter 3 provided a state of the art on the various types of recommender systems and categorizes also the different techniques of exploration in the Reinforcement Learning.

Chapter 4 presents the recommendation system, its agent oriented architecture, the general searching process, the collaborative filtering method used and the external search with Gosis system.

Chapter 5 introduces the Reinforcement Learning method and presents a model that integrates both exploration and exploitation in a common framework. First of all, we define the concept of degree of exploration from a state as the entropy of the probability distribution on the set of admissible actions in this state. This entropy value allows to control the degree of exploration linked to this state, and should be provided by the user. The theoretical results are confirmed by simple simulations showing that the model behaves as expected.

Chapter 6 concludes by discussing the appropriateness of the solution proposed in this dissertation. Our contribution are summarized and some ideas for future works and extensions are proposed.

Chapter 2

Concepts

This chapter explains some relevant concepts and paradigm for the reading of this dissertation. The first Section introduces the agent paradigm and the BDI (belief-desire-intention) behavioural model. Section 2 explains what we understand by knowledge, how to manage it, its hierarchy, types of knowledge existing as well as the subjacent processes. The last Section explore the meaning and the mechanism of a self-organization system.

2.1 Agent and Multi-Agent Systems

An agent defines a system entity, situated in some environment that is capable 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 able to operate without direct, continuous supervision, 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. Proactiviness means that behavior is not exclusively reactive but it is also driven by internal goals, i.e., it may take initiative.

From this, a multi-agent system can be defined as an organization composed of autonomous and proactive agents that interact with each other to achieve common or private goals.

MAS may be either cooperative or competitive agents. In cooperative MAS, the system has a global goal (or set of goals) and the agents that compose the MAS cooperate, possibly by performing diverse tasks, in order to achieve the global goal. This kind of systems is typically adapted to perform distributed problem solving. There is a unique high-level goal decomposed recursively into parallel activities to be performed by a set of agents.

In competitive MAS, each of the component agents has its own set of goals that may or may not meet those of other agents. In this case the MAS is an architecture that allows agents to interact, each one to pursue personal goals and defend its own interests. This kind of systems meets typically engineering requirements of e-commerce, information retrieval applications, web services or peer-to-peer networks. In such environments, every agent generally represents either a client, who wants to obtain some resources or have some service accomplished, or a provider, who wants to sell resources or services at a certain (not necessarily financial) cost. Each agent pursues the goals of the (human or system) actor it represents, and these goals can usually be in conflict.

In order to reason and act in an autonomous way, agents are usually built on rationale models and reasoning strategies that have roots in various disciplines including artificial intelligence, cognitive science, psychology or philosophy. An exhaustive evaluation of these models would be out of the scope of this paper or even this research work. Agent models are proliferating; some include learning capabilities, others intelligent agendas based on statistics, others yet are based on genetic algorithms and so on. However, a simple yet powerful and mature model coming from cognitive science and philosophy that has received a great deal of attention, notably in artificial intelligence, is the Belief-Desire-Intention (BDI) model [9]. This approach has been intensively used to study the design rationale of agents and is proposed as a keystone model in numerous agent-oriented development environments such as Jack or Jade. The main concepts of the BDI agent model are (except the notion of agent itself we have just explained):

- Beliefs that represent the informational state of a BDI agent, that is, what it knows about itself and the world
- Desires (or goals) that are its motivational state, that is, what the agent is trying to achieve
- Intentions that represent the deliberative state of the agent, that is, which plans the agent has chosen for possible execution.

In more detail, a BDI agent has a set of plans, which defines sequences of actions and steps available to achieve a certain goal or react to a specific situation. The agent reacts to events, which are generated by modifications to its beliefs, additions of new goals, or messages arriving from the environment or from another agent. An event may trigger one or more plans; the agent commits to execute one of them, that is, it becomes intention.

Plans are executed one step at a time. A step can query or change the beliefs, performs actions on the external world, and submits new goals. The operations performed by a step may generate new events that, in turn, may start new plans. A plan succeeds when all its steps have been completed; it fails when certain conditions are not met.

2.2 Knowledge management

Our economic and social life is becoming more and more knowledge-driven. It has by now become a truism to say that we live in an information society. But we have only just begun to explore and understand the very real and everyday consequences. One of these consequences is the growing importance of knowledge.

"Knowledge" is a term of which all of us have a good intuitive understanding of what it means, but which is hard to define in any formal way. Many people have tried to come up with satisfactory definitions, but these seem to be at best good approximations.

2.2.1 Concept of Knowledge

What is knowledge? This is a question frequently asked of people in the fields of knowledge management. The same question has been at the roots of philosophical investigation for over two millennia. So it is not likely that we are going to give you a definitive answer.

Knowledge is a concept difficult to define. We will not give a definition of this concept, but rather we will try to understand it.

Data, information, and knowledge are three often-encountered words that belong closely together, seem to have slightly different meanings, yet are often used interchangeably as synonyms, and thus lead to continuing confusion. It is customary to talk about knowledge engineering, information technology, databases, and electronic data processing. But it would be equally reasonable, although uncommon, to speak of, for example, electronic knowledge processing, information bases or data technology. Hence, a frequently asked question is what are the differences between data, information and knowledge?

Knowledge, information, and data are often represented as having a hierarchical relationship.

Let's give a rigorous definition of theses concepts.

- Data - Data are discrete, objective facts about events or objects. A red, green, or yellow light at an intersection is one example. Computers are

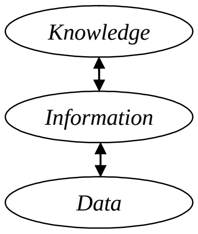


Figure 2.1: Hierarchical relationship of knowledge, information and data

full of data: signal consisting of string of numbers, characters, and other symbols that are blindly and mechanically handled in large quantities.

- Information - Data become information when equipped with meaning. Dixon [18] adroitly notes that information is data "in formation." For a human car driver, a red traffic light is not just a signal of some colored object, rather, it is interpreted as an indication to stop. Let us note that for two different people, the same data can have a different meaning (e.g. the information can be interpreted differently)
- Knowledge - Knowledge is the body of data and information that people bring to bear to practical use in action, in order to carry out tasks and create new information. Knowledge adds two distinct aspects: first, a sense of purpose, since knowledge is the "intellectual machinery" used to achieve a goal; second, a generation capability, because one of the major functions of knowledge is to produce new information. It is not accidental, therefore, that knowledge is proclaimed to be new "factor of production".

2.2.2 Types of Knowledge

In addition to levels of knowledge, Nonaka and Takeuchi [34] distinguish between two types of knowledge – explicit and tacit. Explicit knowledge refers to intellectual artifacts (books, documents, manuals, theories, models, simulations and their interpretations, mathematical expressions, tables, graphs, databases, and so on). It encompasses all levels of cognition (including information and data) that can be put into visual presentations, words, or numbers.

Tacit knowledge refers to cognition that resides in people's heads, such as cumulated wisdom and understanding, institutional knowledge, organizational lore, and basic orientations. It also includes personal knowledge embedded in individual experience in the form of rules of thumb, values, preferences, intuitions, and insights.

As shown in Figure 2.2, Nonaka and Takeuchi [34] assert that the four conversion processes involving these two types of knowledge constitute the essence

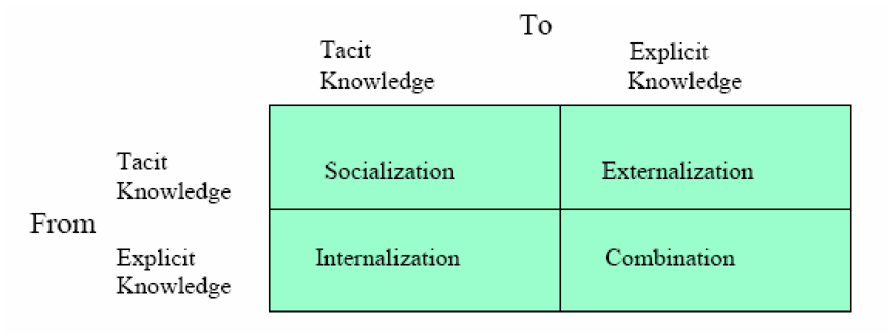


Figure 2.2: Human Processes Corresponding to Knowledge Conversion Processes

of knowledge creation:

- From tacit to tacit (i.e., socialization),
- From tacit to explicit (i.e., externalization),
- From explicit to tacit (i.e., internalization), and
- From explicit to explicit (i.e., combination).

Socialization

It is the creation of tacit knowledge starting from other tacit knowledge thanks to experiments shared by several members of the company. By strong interaction, the tacit knowledge of a person or some people can become the knowledge of other people. The coaching, the formation, the techniques of brainstorming are examples of process of socialization.

Combination

The combination is the explicit creation of knowledge starting from explicit knowledge. Through a common language and varied supports of transmission (paper, network, Intranet. . .), explicit knowledge of the actors of the company can be combined to give form to new knowledge. The formation, the exchange of documents and the division of information via Intranets support the combination of knowledge.

Externalisation

It is the tacit conversion of knowledge into explicit knowledge. The lack of common concepts returns the formalization of tacit knowledge difficult. This articulation is however strategic since it makes it possible to make exploitable the richness of the tacit knowledge of the individuals. It must call upon mechanisms of analogy and metaphor to succeed. It concerns the convergence of the people knowledge on a shared objective.

Internalization

It is by the practice of knowledge or know-how that this one goes establish in the behaviours of the actors (experiment). The explicit knowledge then forms part of the memory of the firm and what we can call its "culture".

They further claim that conversions between tacit and explicit knowledge are particularly important. Only by tapping into tacit knowledge can new and improved explicit knowledge be created. In turn, better explicit knowledge is essential for stimulating the development of new, higher level, tacit knowledge.

Although knowledge management has tended to focus on improving and managing explicit knowledge (e.g., artefacts), Nonaka and Takeuchi argue that this is not where the emphasis ought to lie. Knowledge creation and application require far more than well-structured knowledge artefacts. But because tacit knowledge is difficult to formalize, make explicit, and manage, it has been overlooked by organizations. But tacit knowledge, especially high level tacit knowledge, will become increasingly important as organizations face the ever pressing need to create new knowledge. Also, as organizations develop more systematic practices and techniques to foster this knowledge and to facilitate its conversion to explicit knowledge, tacit knowledge will no longer be seen as "unmanageable."

2.3 Self-organization

The term self-organization is often referred in articles, but it however does not have a general meaning. It is a vague term which the authors like to use to define systems having some dynamic characteristics. The large number of field using this term is also the causes of the abundance of definitions. One finds definitions in cybernetics, thermodynamics, mathematics, information theory, synergetic and of others. We will not attempt to propose yet another definition of self-organizing systems. However we will try to understand these systems by exploring the necessary conditions in order to call a system "self-organizing".

2.3.1 Self-organizing system

For a system self-organized, the regulation and the adaptation are not the only functionalities. The creation of its own organization is also another of its functions. From this point of view, there is a fundamental difference with the current systems where the organization in this case is created by the designers themselves. [23] define the organization as a structure with a function. Structure indicates that there is a particular order among the components of the system. It requires both connections that integrate the parts into a whole, and separations that differentiate subsystem, so to avoid interference. Function means that the structure achieves an objective.

Self-organization then means that a functional structure appears and maintains spontaneously. The control needed to achieve this must be distributed over all participating components. If it was centralized in a subsystem or module, then this module could in principle be removed and the system would lose its organization [23].

The robustness is one of qualities which characterize self-organized systems. Indeed, they have a good resistance compared to various types of error, disturbance and even to a partial destruction. They will repair or correct most damage themselves, getting back to their initial state. When the damage becomes too significant, the system starts to be degraded gradually but without causing a sudden breakdown. It is also about a dynamic system that is able to adapt its organization in relation to the modifications of the environment.

2.3.2 Mechanisms of self-organizing

These descriptions show the power of self-organization, but only in very limited contexts, where both the components and their desired function are well-defined.

How can we apply self-organization to an environment as complex and diverse as the Web, a corporate Intranet, or even a desktop computer with its ever changing software and data configuration? To achieve self-organization on that scale, we need a much deeper insight into how it precisely works [22].

A self-organizing system consists of a large number of interacting components. The system is dynamic in the way that the components are constantly changing state relative to each other. But because of mutual dependency changes are not arbitrary: some relative states are "preferable", in the sense that they will be reinforced or stabilized, while others are inhibited or eliminated

Changes initially are local because components only interact with their immediate "neighbours". They are virtually independent of components farther away. But self-organization is often defined as global order emerging from local interactions. We can picture this process as follows. Two interacting components pass through a variety of configurations until they find one that is mutually "satisfactory", i.e. stable. We might say that they have adapted to each other, and now fit together. To achieve global order, this fit must propagate to the other components. For example, two people who discovered a common interest may start talking about this to others, and end up founding a club, political movement, or company. If the components are interchangeable the resulting structure will be regular. If each component has its own, individual characteristics the structure will be more complex: each component has to fit in its own niche within the environment formed by the others.

This propagation of fit is typically self-reinforcing: additional components join ever more quickly. The reason is that a larger assembly exerts a stronger attraction on the remaining independent components, offering more niches in which they can fit. This positive feedback produces an explosive growth or reproduction of the assembly. Growth only stops when the resources are exhausted, that is, when all components that could be fit into the assembly have been fit. This may happen because the remaining components are too different to fit in this type of configuration, or because they got assimilated into a rival assembly.

The assembly has now stabilized and feedback becomes mostly negative. This means that it will counteract any loss of organization. Self-maintenance has become its implicit purpose, and each component will perform its function toward this goal. The assembly as a whole, however, can still interact with other assemblies, but at a different level. Thus, assemblies formed from individual components start acting like higher level components. These can in turn self-organize into even higher level components. This process continues recursively, for as long as there are components to interact with, generating ever higher levels of complexity.

Chapter 3

State of the art

3.1 Recommendation system

This section aims to present a survey of the state of the art of recommender systems based on [1]. Firstly, we present the motivations of such systems, then we will evoke the main categories of recommendation methods (the recommendation based on the contents, the collaborative recommendation and the hybrid recommendation).

3.1.1 Motivation

Recommender systems have become an important research area since the appearance of the first papers on collaborative filtering in the mid-1990s. There has been much work done both in the industry and academia on developing new approaches to recommender systems over the last decade. The interest in this area still remains high because it constitutes a problem-rich research area and because of the abundance of practical applications that help users to deal with information overload and provide personalized recommendations, content, and services to them. Examples of such applications include recommending books, CDs, and other products at Amazon.com, movies by MovieLens, and news at VERSIFI Technologies (formerly AdaptiveInfo.com). Moreover, some of the vendors have incorporated recommendation capabilities into their commerce servers.

However, despite all of these advances, the current generation of recommender systems still requires further improvements to make recommendation methods more effective and applicable to an even broader range of real-life applications, including recommending vacations, certain types of financial services to investors, and products to purchase in a store made by a “smart” shopping cart. These improvements include better methods for representing user behavior and the information about the items to be recommended, more advanced recommendation modelling methods, incorporation of various contextual information into the recommendation process, utilization of multcriteria ratings, development of less intrusive and more flexible recommendation methods that also rely on the measures that more effectively determine performance of recommender systems.

3.1.2 Categories of recommendation system

Although the roots of recommender systems can be traced back to the extensive work in cognitive science, approximation theory, information retrieval, forecasting theories, and also have links to management science and to consumer choice modeling in marketing, recommender systems emerged as an independent research area in the mid-1990s when researchers started focusing on recommendation problems that explicitly rely on the ratings structure. In its most common formulation, the recommendation problem is reduced to the problem of estimating ratings for the items that have not been seen by a user. Intuitively, this estimation is usually based on the ratings given by this user to other items and on some other information that will be formally described below. Once we can estimate ratings for the yet unrated items, we can recommend to the user the item(s) with the highest estimated rating(s).

More formally, the recommendation problem can be formulated as follows: Let C be the set of all users and let S be the set of all possible items that can be recommended, such as books, movies, or restaurants. The space S of possible items can be very large, ranging in hundreds of thousands or even millions of items in some applications, such as recommending books or CDs. Similarly, the user space can also be very large—millions in some cases. Let u be a utility function that measures the usefulness of item s to user c, i.e., $u : C \times S \to R$, where R is a totally ordered set (e.g., nonnegative integers or real numbers within a certain range). Then, for each user $c \in C$, we want to choose such item $s' \in S$ that maximizes the user's utility. More formally:

$$\forall c \in C, s'_c = \arg\max_{s \in S} u(c, s)$$

In recommender systems, the utility of an item is usually represented by a rating, which indicates how a particular user liked a particular item, e.g., John Doe gave the movie "Harry Potter" the rating of 7 (out of 10). However, as indicated earlier, in general, utility can be an arbitrary function, including a profit function. Depending on the application, utility u can either be specified by the user, as is often done for the user-defined ratings, or is computed by the application, as can be the case for a profit-based utility function.

Each element of the user space C can be defined with a profile that includes various user characteristics, such as age, gender, income, marital status, etc. In the simplest case, the profile can contain only a single (unique) element, such as User ID. Similarly, each element of the item space S is defined with a set of characteristics. For example, in a movie recommendation application, where S is a collection of movies, each movie can be represented not only by its ID, but also by its title, genre, director, year of release, leading actors, etc.

The central problem of recommender systems lies in that utility u is usually not defined on the whole $C \times S$ space, but only on some subset of it. This means u needs to be extrapolated to the whole space $C \times S$. In recommender systems, utility is typically represented by ratings and is initially defined only on the items previously rated by the users. For example, in a movie recommendation application (such as the one at MovieLens.org), users initially rate some subset of movies that they have already seen. Therefore, the recommendation engine should be able to estimate (predict) the ratings of the nonrated movie/user combinations and issue appropriate recommendations based on these predictions.

Extrapolations from known to unknown ratings are usually done by 1) specifying heuristics that define the utility function and empirically validating its performance and 2) estimating the utility function that optimizes certain performance criterion, such as the mean square error.

Once the unknown ratings are estimated, actual recommendations of an item to a user are made by selecting the highest rating among all the estimated ratings for that user. Alternatively, we can recommend the N best items to a user or a set of users to an item.

The new ratings of the not-yet-rated items can be estimated in many different ways using methods from machine learning, approximation theory, and various heuristics. Recommender systems are usually classified according to their approach to rating estimation and, in the next section, we will present such a classification that was proposed in the literature and will provide a survey of different types of recommender systems. Recommender systems are usually classified into the following categories, based on how recommendations are made [3]:

- *Content-based recommendations*: The user will be recommended items similar to the ones the user preferred in the past;
- *Collaborative recommendations*: The user will be recommended items that people with similar tastes and preferences liked in the past;
- *Hybrid approaches*: These methods combine collaborative and content-based methods.

In addition to recommender systems that predict the absolute values of ratings that individual users would give to the yet unseen items (as discussed above), there has been work done on preference-based filtering, i.e., predicting the relative preferences of users. For example, in a movie recommendation application, preference-based filtering techniques would focus on predicting the correct relative order of the movies, rather than their individual ratings. However, this Section focuses primarily on rating-based recommenders since it constitutes the most popular approach to recommender systems.

Content-based recommendations

In content-based recommendation methods, the utility $u(c, s)$ of item s for user c is estimated based on the utilities $u(c, s_j)$ assigned by user c to items $s_j \in S$ that are "similar" to item s. For example, in a movie recommendation application, in order to recommend movies to user c, the content-based recommender system tries to understand the commonalities among the movies user c has rated highly in the past (specific actors, directors, genres, subject matter, etc.). Then, only the movies that have a high degree of similarity to whatever the user's preferences are would be recommended.

The content-based approach to recommendation has its roots in information retrieval and information filtering research. Because of the significant and early advancements made by the information retrieval and filtering communities and because of the importance of several text-based applications, many current content-based systems focus on recommending items containing textual information, such as documents, Web sites (URLs), and Usenet news messages. The improvement over the traditional information retrieval approaches comes from the use of user profiles that contain information about users' tastes, preferences, and needs. The profiling information can be elicited from users explicitly, e.g., through questionnaires, or implicitly (learned from their transactional behaviour over time).

More formally, let $Content(s)$ be an item profile, i.e., a set of attributes characterizing item s. It is usually computed by extracting a set of features from item s (its content) and is used to determine the appropriateness of the item for recommendation purposes. Since, as mentioned earlier, content-based systems are designed mostly to recommend text-based items, the content in these systems is usually described with keywords. For example, a content-based component of the Fab system [3], which recommends Web pages to users, represents Web page content with the 100 most important words. Similarly, the Syskill & Webert system represents documents with the 128 most informative words. The "importance" (or "informativeness") of word k_j in document d_j is determined with some weighting measure w_{ij} that can be defined in several different ways.

As was observed in [3] content-based recommender systems have several limitations. Content-based techniques are limited by the features that are explicitly associated with the objects that these systems recommend. Therefore, in order to have a sufficient set of features, the content must either be in a form that can be parsed automatically by a computer (e.g., text) or the features should be assigned to items manually. While information retrieval techniques work well in extracting features from text documents, some other domains have an inherent problem with automatic feature extraction. For example, automatic feature extraction methods are much harder to apply to multimedia data, e.g., graphical images, audio streams, and video streams. Moreover, it is often not practical to assign attributes by hand due to limitations of resources.

Another problem with limited content analysis is that, if two different items are represented by the same set of features, they are indistinguishable. Therefore, since text-based documents are usually represented by their most important keywords, content-based systems cannot distinguish between a well-written article and a badly written one, if they happen to use the same terms.

Collaborative recommendations

Unlike content-based recommendation methods, collaborative recommender systems (or collaborative filtering systems) try to predict the utility of items for a particular user based on the items previously rated by other users. More formally, the utility $u(c, s)$ of item s for user c is estimated based on the utilities $u(c_j, s)$ assigned to item s by those users $c_j \in C$ who are "similar" to user c. For example, in a movie recommendation application, in order to recommend movies to user c, the collaborative recommender system tries to find the "peers" of user c, i.e., other users that have similar tastes in movies (rate the same movies similarly). Then, only the movies that are most liked by the "peers" of user c would be recommended.

There have been many collaborative systems developed in the academia and the industry. It can be argued that the Grundy system was the first recommender system, which proposed using stereotypes as a mechanism for building models of users based on a limited amount of information on each individual user. Using stereotypes, the Grundy system would build individual user models and use them to recommend relevant books to each user. Later on, the Tapestry system relied on each user to identify like-minded users manually. GroupLens, Video Recommender, and Ringo were the first systems to use collaborative filtering algorithms to automate prediction. Other examples of collaborative recommender systems include the book recommendation system from Amazon. com, the PHOAKS system that helps people find relevant information on the WWW, and the Jester system that recommends jokes.

According to [10], algorithms for collaborative recommendations can be grouped into two general classes:

- *Memory-based algorithms* essentially are heuristics that make rating predictions based on the entire collection of previously rated items by the users.
- *Model-based algorithms* use the collection of ratings to learn a model, which is then used to make rating predictions.

The pure collaborative recommender systems do not have some of the shortcomings that content-based systems have. In particular, since collaborative systems use other users' recommendations (ratings), they can deal with any kind of content and recommend any items, even the ones that are dissimilar to those seen in the past. However, collaborative systems have their own limitations [3], as described below.

In any recommender system, the number of ratings already obtained is usually very small compared to the number of ratings that need to be predicted. Effective prediction of ratings from a small number of examples is important. Also, the success of the collaborative recommender system depends on the availability of a critical mass of users. For example, in the movie recommendation system, there may be many movies that have been rated by only few people and these movies would be recommended very rarely, even if those few users gave high ratings to them. Also, for the user whose tastes are unusual compared to the rest of the population, there will not be any other users who are particularly similar, leading to poor recommendations [3]. One way to overcome the problem of rating sparsity is to use user profile information when calculating user similarity. That is, two users could be considered similar not only if they rated the same movies similarly, but also if they belong to the same demographic segment. For example, we can use the gender, age, area code, education, and employment information of users in the restaurant recommendation application. This extension of traditional collaborative filtering techniques is sometimes called "demographic filtering". Another approach that also explores similarities among users has been proposed, where the sparsity problem is addressed by applying associative retrieval framework and related spreading activation algorithms to explore transitive associations among consumers through their past transactions and feedback.

Hybrid approaches

Several recommendation systems use a hybrid approach by combining collaborative and content-based methods, which helps to avoid certain limitations of content-based and collaborative systems. Different ways to combine collaborative and content-based methods into a hybrid recommender system can be classified as follows:

1. implementing collaborative and content-based methods separately and combining their predictions,
2. incorporating some content-based characteristics into a collaborative approach,
3. incorporating some collaborative characteristics into a content-based approach, and
4. constructing a general unifying model that incorporates both content-based and collaborative characteristics

All of the above approaches have been used by recommender systems researchers, as described below.

Combining Separate Recommenders

One way to build hybrid recommender systems is to implement separate collaborative and content-based systems.

Then, we can have two different scenarios. First, we can combine the outputs (ratings) obtained from individual recommender systems into one final recommendation using either a linear combination of ratings or a voting scheme. Alternatively, we can use one of the individual recommenders, at any given moment choosing to use the one that is "better" than others based on some recommendation "quality" metric. For example, the DailyLearner system selects the recommender system that can give the recommendation with the higher level of confidence, while another recommender system can choose the one whose recommendation is more consistent with past ratings of the user.

Adding Content-Based Characteristics to Collaborative Models

Several hybrid recommender systems, including Fab and the "collaboration via content" approach are based on traditional collaborative techniques but also maintain the content-based profiles for each user. These content-based profiles, and not the commonly rated items, are then used to calculate the similarity between two users.

This allows to overcome some sparsity-related problems of a purely collaborative approach since, typically, not many pairs of users will have a significant number of commonly rated items. Another benefit of this approach is that users can be recommended an item not only when this item is rated highly by users with similar profiles, but also directly, i.e., when this item scores highly against the user's profile [3]. Good et al. employ a somewhat similar approach in using the variety of different filterbots—specialized content-analysis agents that act as additional participants in a collaborative filtering community. As a result, the users whose ratings agree with some of the filterbots' ratings would be able to receive better recommendations.

Adding Collaborative Characteristics to Content-Based Models

The most popular approach in this category is to use some dimensionality reduction technique on a group of contentbased profiles. For example, we can use latent semantic indexing (LSI) to create a collaborative view of a collection of user profiles, where user profiles are represented by term vectors, resulting in a performance improvement compared to the pure contentbased approach.

Developing a Single Unifying Recommendation Model
Many researchers have followed this approach in recent years. For instance, Basu et al. propose using content-based and collaborative characteristics (e.g., the age or gender of users or the genre of movies) in a single rule-based classifier. Popescul and Schein propose a unified probabilistic method for combining collaborative and content-based recommendations, which is based on the probabilistic latent semantic analysis. Yet, another approach is proposed by [15], where Bayesian mixed-effects regression models are used that employ Markov chain Monte Carlo methods for parameter estimation and prediction.

Hybrid recommendation systems can also be augmented by knowledge-based techniques, such as case-based reasoning, in order to improve recommendation accuracy and to address some of the limitations (e.g., new user, new item problems) of traditional recommender systems. For example, knowledge-based recommender system Entree uses some domain knowledge about restaurants, cuisines, and foods (e.g., that "seafood" is not "vegetarian") to recommend restaurants to its users. The main drawback of knowledge-based systems is a need for knowledge acquisition—a well-known bottleneck for many artificial intelligence applications. However, knowledge-based recommendation systems have been developed for application domains where domain knowledge is readily available in some structured machine-readable form, e.g., as an ontology. For example, the Quickstep and Foxtrot systems use research paper topic ontology to recommend online research articles to the users.

Moreover, several papers empirically compare the performance of the hybrid with the pure collaborative and content-based methods and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches.

3.2 Reinforcement Learning

This section presents several exploration techniques and categorizes them in two families:*undirected* and *directed* methods

3.2.1 Introduction

One of the key features of reinforcement learning (RL) is that a learning agent is not instructed what actions it should perform; instead, the agent has to evaluate all available actions, and then decide for itself on the best way of behaving. This creates the need for an RL agent to actively explore its environment, in order to discover good behavior strategies. Ensuring an efficient exploration process and balancing the risk of taking exploratory actions with the benefit of information gathering are of great practical importance for RL agents, and have been the topic of much recent research.

Existing exploration strategies can be divided into two broad classes: undirected and directed methods. Undirected methods are concerned only with ensuring sufficient exploration, by selecting all actions infinitely often. The ϵ-greedy and Boltzman exploration strategies are notable examples of such methods. Undirected methods are very popular because of their simplicity, and because they do not have additional requirements of storage or computation. However, they can be very inefficient for certain domains. For example, in deterministic goal directed tasks with a positive reward received only upon entering the goal state, undirected exploration is exponential in the number of steps needed for an optimal agent to reach the goal state [46]. On the other hand, by using some information about the course of learning, the same tasks can be solved in time polynomial in the number of states and maximum number of actions available in each state [46]. The impact of exploration is believed to be even more important for stochastic environments. Directed exploration strategies attempt not only to ensure a sufficient amount of exploration, but also to make the exploration efficient, by using additional information about the learning process. These techniques often aim to achieve a more uniform exploration of the state space, or to balance the relative profit of discovering new information versus exploiting current knowledge. Typically, directed methods keep track of information regarding the learning process and/or learn a model of the system. This requires extra computation and storage in addition to the resources needed by general on-line RL algorithms, in order to make better exploration decisions.

3.2.2 Exploration in Reinforcement Learning

The goal of an exploration policy is to allow the RL agent to gather experience with the environment in such a way as to find the optimal policy as quickly as possible, while also gathering as much reward as possible during learning. This goal can be itself cast a learning problem, often called optimal learning [6]. Solving this problem would require the agent to have a probabilistic model of the uncertainty about its own knowledge of the environment, and to update this model as learning progresses. Solving the optimal learning problem then becomes equivalent to solving the partially observable MDP (POMDP) defined by this model, which is generally intractable. However, various heuristics can be used to decide which exploration policy to follow, based only on certain aspects of the uncertainty about the agent's knowledge of the environment.

As discussed previously, existing exploration techniques can be grouped in two main categories: undirected and directed methods. Undirected methods ensure that each action will be selected with non-zero probability in each visited state. For instance, the ϵ-*greedy* exploration strategy selects the currently greedy action (the best according to the current estimate of the optimal action-value function $Q(s,a)$), in any given state, with probability $(1-\epsilon)$, and selects a uniformly random action with probability ϵ.

Another popular choice for undirected exploration, the *Boltzman* distribution assigns probability $\pi(s,a)$ of taking action a in state s as

$$\pi(s,a) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{b \in A} e^{\frac{Q(s,b)}{\tau}}}$$

, where τ is a positive temperature parameter that decreases the amount of randomness as it approaches zero. When using on-policy RL algorithms, such as SARSA, the exploration rate (ϵ in the ϵ-greedy exploration and τ in Boltzman exploration) has to decrease to zero with time in an appropriate manner in order to ensure convergence to the optimal (deterministic) policy. In practice, however, constant exploration rates are often used.

Directed exploration methods typically keep some information about the state of knowledge of the agent, estimating certain aspects of its uncertainty. The action to be taken is usually selected by maximizing an evaluation function that combines action-values with some kind of exploration bonuses, δ_i:

$$N(s,a) = K_0 Q(s,a) + K_1 \delta_1(s,a) + ... + K_k \delta_k(s,a)$$

Exploration is driven mainly by the exploration bonuses that change over time. The positive constants K_i control the exploration-exploitation balance.

Directed exploration methods differ in the kind of exploration bonuses they define, which reflect different heuristics regarding what states are important to revisit. For example, counter-based methods [46] direct exploration toward the states that were visited least frequently in the past. Recency-based exploration [46] prefers instead the states that were visited least recently. In both of these cases, the result is a more homogeneous exploration of the state space. Error-based exploration prefers actions leading to states whose value changed most in past updates. Interval Estimation (IE), as well as its global equivalent, IEQL+, bias exploration toward actions that have the highest variance in the action value samples. In the value of information strategy, the exploration-exploitation tradeoff is solved with a myopic approximation of the value of perfect information. The E^3 algorithm [26] learns a model of the MDP. Based on the estimated accuracy of this model and a priori knowledge of the worst-case mixing time of the MDP and the maximum attainable returns, E^3 explicitly balances the profit of exploitation and the possibility of efficient exploration. Due to this balancing, E^3 provably achieves near-optimal performance in polynomial time. However, there is little practical experience available with this algorithm.

Chapter 4

Recommendation system

This chapter presents a multi-agent recommendation system based on the concepts of knowledge sharing. The system is a generalization of Collaborative Filtering that is a technique by which the user interest for information is predicted from the knowledge of the other user profile. We compute similarities between users to cluster them into groups with similar interest (i.e., community). To compute the similarities, we use an innovative collaborative filtering method based on the Markov-chain model, the random-walk model

The first Section introduce the purpose of our recommendation system Section 2 gives description of the architecture of our recommendation system. The search process is also explained in this Section. Section 3 details the collaborative-filtering method based on the Markov-chain model. Finally, Section 4 details the external search process.

4.1 Introduction

The goal of our work is to conceive a recommendation system which would self-organize the users in community and would allow a collaborative filtering of information on the basis of users profile.

However, such recommendation system is difficult to implement when the collection of information to analyze is too significant and the collaborative-filtering methods are too heavy (i.e., most of collaborative-filtering methods need an important computation capacity). We need to choose an approach that is distributed, flexible and capable of certain autonomy.

Distributed system is a collection of autonomous entities connected which enable the coordination of theirs activities and the sharing of the resources. Contrary to centralized systems, distributed approach provides an important computation capacity essentially thanks to tasks distribution. Given this definition and to be efficient in term of computation capacity, we need to select a distributed architecture (especially by using collaborative-filtering methods) for our recommendation system.

Flexible and autonomous requirements refer to a system with a high degree of adaptability and capable of autonomous action in order to meet its design objectives. A recommendation system with these requirements can take decisions himself (e.g., which members of the community ask for a recommendation, how to ranks the results . . .) and adapts its behaviour according to user profile, community experiences and so on.

These three characteristics (i.e., distributed architecture, flexibility and adaptability) are found naturally in the multi-agent paradigm.

In this perspective, multi-agent systems (MAS) seem to be popular to build robust and flexible application [49] by distributing responsibilities among autonomous and cooperating agents. An agent is a computational entity that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behaviour at least partially depends on its own experience. These requirements make possible for a MAS to cope with problems (e.g., the processing of huge amounts of data, or of data that arises at geographically distinct locations, the requirement of more autonomy, . . .) that conventional systems (majority being systems centralised) were not able to solve.

Agents and multi-agent systems applied to search area are reported in literature. The main proposal is to use an agent that assists its user during web search [13][30][47]. The agent can track user browsing or it can form user profiles in different areas in order to anticipate items of interest. Multi-agent systems aimed to help user during web search implement various approaches. It can be alliance of several agents providing user with result [31]. It is also possible to apply auction protocol and reward mechanism to agent collaboration [48]. Other authors [8][12][50] propose personal agents acting on behalf of their users, collaborating with each other and having the goal to improve their user' browsing. In some of the systems considered so far user is supposed to perform an extra work during search, e.g. he/she needs to specify the areas of his/her interest or to analyze a lot of results of searches similar to the current one. Sometimes there are also restrictions like ability to use only certain part of pre-defined knowledge or ontology.

4.2 Architecture and process descriptions

4.2.1 Architecture

This section describes the general architecture and process of the system. The aim of the system is to help a community of people to find information that is well suited to their expectation. To attain this goal, we propose a multi-agent system composed by a set of personal agents working within a community to exploit a great amount of knowledge.

A personal agent is assigned to each user of the system and its task is to assists him in his work. Firstly, it helps the user in his searching by finding the good information using the community knowledge and external resources (e.g. search engine like Google). Secondly, it builds a user profile based on his past actions. The user past actions are essentially the information (links) proposed by his personal agent and accepted by him. The most a link is accepted by a user, the most the link become important for the information requested by the user. Finally, the personal agent must compare the profiles of the community members to calculate the similarities between them and its user. Knowing the similarity that exists between the user profile and the others, the personal agent will be able to give an appreciation and a priority to the information which it receives from them. Intuitively information received from a member having a profile rather close to the user has more chance to be accepted by this last than information sent by another with a profile much more distant.

In the design phase, two choices were made concerning the architectures of the global system and of the personal agents. Theses architectures must make it possible for the system and its agents to fulfil their mission of assistance and recommendation in an optimal way. To find the best architecture, we use the organisational styles oriented-agent provided by Tropos [28][29] methodology. Organisational styles guide the development of the organizational model for a system.

While basing on Tropos organisational styles, our choice has naturally preferred the co-optation at the highest level (the global system). The co-optation is a style that involves the incorporation of representatives of external systems into the decision-making or advisory structure and behaviour of an initiating organization. By co-opting representatives of external systems, organizations are, in effect, trading confidentiality and authority for resource, knowledge assets and support. The initiating system has to come to terms with the contractors which is being made on its behalf, and each co-optated actor has to reconcile and adjust its own views with the policy of the system it has to communicate. Taking into consideration this definition, it is easy to conceive the whole system like a multi-agent system composed by the personal agents representing the external systems.

Figure 4.1 represents the system with a co-optation structure. The personal agents of the system cooperate with each others to share community's experience and to provide a support for collaborative searching. A different external system (external search engine API) is used by the agents to provide new links relatively to the community knowledge.

At a lower level, the personal agents are too complex to be seen like simple system agent. To control this complexity, these agents must themselves be implemented like multi-agent systems. Regarding the personal agent goals and behaviours, our choice has naturally preferred the joint venture. This style

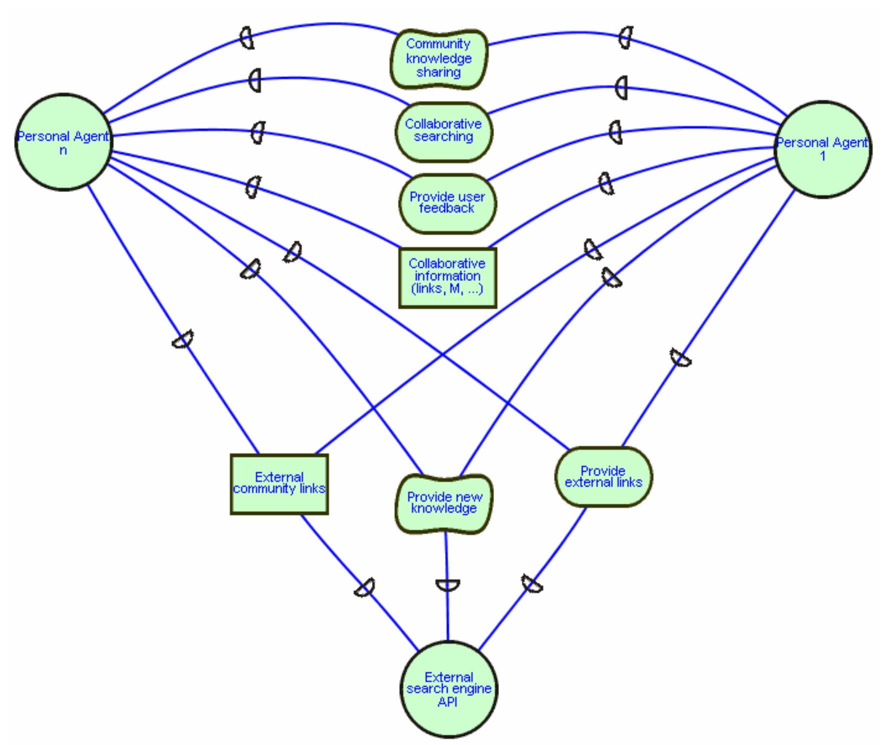


Figure 4.1: The co-optation style is used to model the interactions of the personal agents at the highest level (i.e., at the platform level).

is used to model how business stakeholders (individual, physical or social systems) coordinate in order to achieve common goal. Theses alternative styles are developed by organizational theory and strategic alliances literature. The joint venture is a decentralized style that involves an agreement between two or more principal partners in order to obtain the benefits derived from operating at a larger scale and reusing the experience and knowledge of the partners. Each principal partner can manage and control itself on a local dimension and interact directly with other principal partners so as to exchange, provide and receive services, data and knowledge. However, the strategic operation and coordination is delegated to a Joint Management actor, who coordinates tasks and manages the sharing of knowledge and resources.

Figure 4.2 shows the instantiation of the joint venture structure of our system. This instantiation is realised using the social patterns provided by Tropos ontology. Unlike organizational styles, they focus on the social structure necessary to achieve one particular goal, instead of the overall goals of the organizations. A social pattern defines the actors (their roles and responsibilities) and

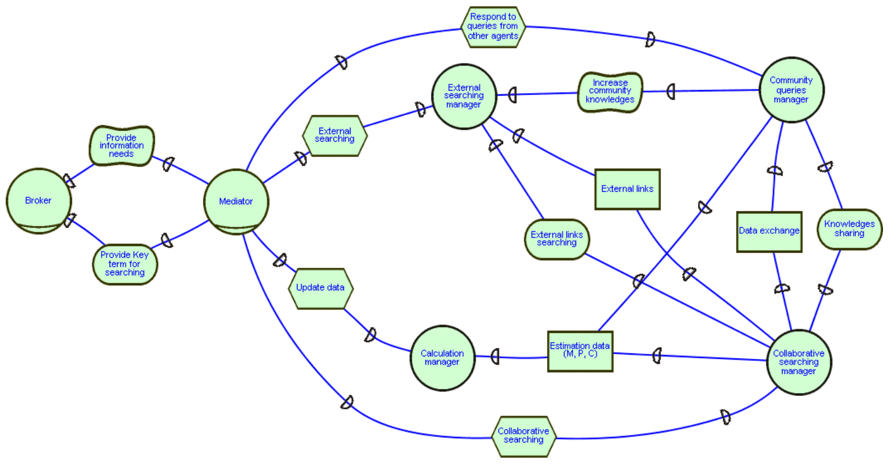

Respond to queries from other agents
Broker
Provide information needs
Provide Key term for searching
Mediator
External searching
External searching manager
Increase community knowledges
Community queries manager
External links
External links searching
Data exchange
Knowledges sharing
Update data
Calculation manager
Estimation data (M, P, C)
Collaborative searching manager
Collaborative searching

Figure 4.2: The joint venture style is used at a lower level to model the goals, behaviours, etc. of a personal agent

the social dependencies that are necessary for achievement of the goal. We use two social patterns, the broker and the mediator respectively for the public and the private interface role. Tropos defines a broker like an arbiter and intermediary who has access services of an actor (Provider) in order to satisfy the request of a consumer. The second pattern, the mediator, mediates interactions among different actor. It has acquaintance models of colleagues and coordinates the cooperation between them.

The role of the collaborative searching manager is to answer the user request by carrying out 3 tasks. Firstly, it must check if the user did not already bring an action on same information into the past. If that is the case, it will provide to the user the most relevant links accepted during the preceding requests. For recall, the relevance of a link is measured by the number of acceptance made by the user on this link. Secondly, it must propagate the request near the other members of the community so that the experiment sharing can be done. The propagation will be done to the members having a profile close to that of the initiator of the request. Finally, to allow the introduction of new knowledge (represented by links) within the community, this manager will also have to propagate the request towards an external search engine to the system. Once these three tasks finished, before returning the results to the user, the received links will have to be filtered in order to eliminate the doubles (it is of course possible that the same links are obtained several times for the same request) and they will have also to be sorted according to their degree of relevance and their source.

The main task of the external searching manager is to provide external links to the community by using an external search engine (e.g. Google). Its role is primordial because it supplies the community experience with new knowledge.

The community queries manager is responsible for the requests coming from the other community members, and not from its user. Its responsibility extends on two dimensions. Firstly, it must answer the request by using its local experiment, i.e. the links accepted by its own user. Secondly, it must propagate the request near the members who are closest to his profile. The sharing of knowledge within the community is done mainly thanks to this manager. Thus each member will be able to bring his experiment while answering the requests coming from the others.

Finally, the calculation manager deals with the calculation of the various data (M, C, P. . .) necessary to the creation of the profiles, the similarities between profiles and the scores concerning the links. By dedicating a manager specifically for all these calculations, the system will be capable to answer to requests while carrying out these calculations.

4.2.2 Process

In this part, we will describe the process of search for our recommendation system. Following the request of a user, this process provides relevant links towards Web pages. For that, research is based on the keywords of the user, but also on its profile and the knowledge held by the community to which this user belongs.

Taking into account only the knowledge of the user and his community, the system will not be able to answer the requests containing unknown keywords (i.e. keywords for which the user and the members of his community never made request).

To cope with this issue, the result of research must also include links coming from an external source. This external entity will answer a request while being based only on the keywords and without regard to any other information.

Notice that when the keywords of a request are known of a community, it is nevertheless necessary to use an external entity to obtain new links. Indeed, obtaining these new links will increase the user knowledge (and thus the knowledge of his community) by presenting for example links to pages recently created.

We will now describe more deeply the searching process. We will separate this process in three different levels of research: intern, collaborative and external (Figure 4.3).

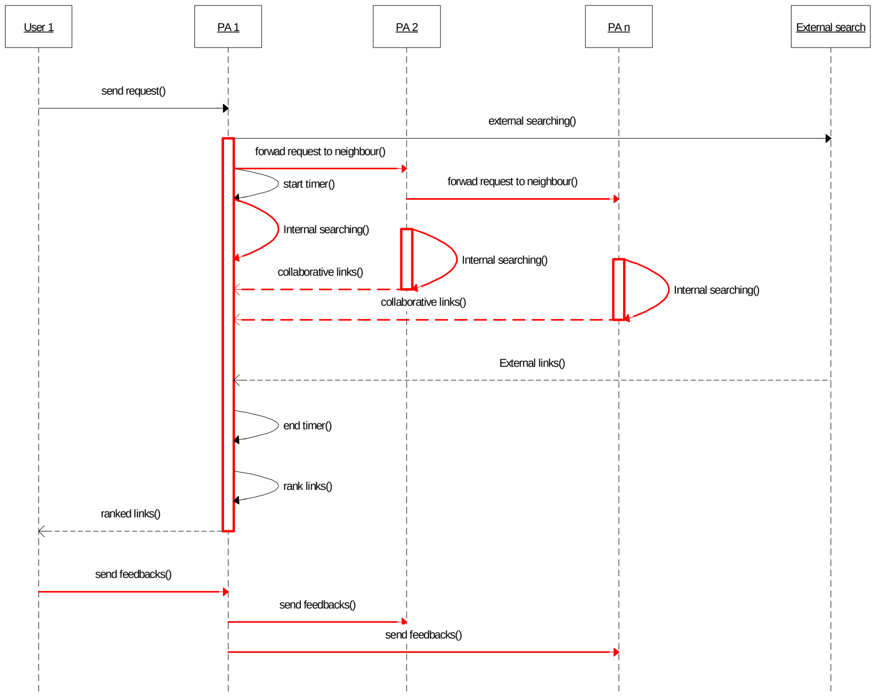


Figure 4.3: The sequence diagram of the searching process including the internal, collaborative and external search

Internal search

To answer the requests of a user, internal search uses knowledge specific to this user. This knowledge was created by using the links accepted by this user in the past. This research is based on the assumption that links accepted by a user in previous requests, can be still accepted by this user in next requests.

Collaborative search

Whereas internal research used only knowledge specific to a user, collaborative research will take into account the set of knowledge specific to a community.

Indeed, this research will use a collaborative-filtering method (this method is presented in the following section) which exploits knowledge of the users having a profile similar to the profile of the search initiator (i.e., the community members of the initiator).

This research is based on the assumption that knowledge of a community can be used by the system to suggest the members of this community the interesting links. In other words, the relevant links for a major part of the community members can be also relevant for the other members of this same community.

External search

Contrary to the two previous modes of research, external search is not based on any knowledge acquired as a preliminary by the user or the community. This research requires an external entity which launches a search using only the user keywords. In our case, this external entity is an external search engine (e.g., Google).

As remind, this external search is of primary importance for our recommendation system. It provides results even for keywords which have never been mentioned in a request. In addition, it increases knowledge by adding new links considered interesting by the user.

4.3 Collaborative search: A Markov-chain model of MAS architecture

4.3.1 Definition of the weighted graph

A weighted graph G is associated with a MAS (Multi-agents system) architecture in the following obvious way: agents correspond to nodes of the graph and each interaction between two agents is expressed as an edge connecting the corresponding nodes.

In our case, this means that each instantiated agent (i.e. personal agent) corresponds to a node of the graph, and each information exchange about keywords is expressed as an edge connecting the corresponding nodes.This model is an extension of the work on collaborative recommendation of F. Fouss et al [20],[21].

The weight $w_{ij} > 0$ of the edge connecting node i and node j (say there are n nodes in total) should be set to some meaningful value, with the following convention: the more important the relation between node i and node j, the larger the value of w_{ij} and consequently the easier the communication through the edge. Notice that we require that the weights be both positive ($w_{ij} > 0$) and symmetric ($w_{ij} = w_{ji}$). The elements a_{ij} of the adjacency matrix A of the graph are defined in a standard way as

$$a_{ij} = \begin{cases} w_{ij} \text{ if node } i \text{ is connected to node } j \\ 0 \text{ otherwise} \end{cases}$$

where A is symmetric.

Because of the way the graph is defined, user agents who have the same keywords, and therefore have similar interest, will have a comparatively large number of short paths connecting them. On the contrary, for user agents with different interests, we can expect that there will be fewer paths connecting them and that these paths will be longer.

4.3.2 A random-walk model on the graph

The Markov chain describing the sequence of nodes visited by a random walker is called a random walk. We associate a state of the Markov chain to every node; we also define a random variable, $s(t)$, representing the state of the Markov model at time step t. If the random walker is in state $i$at time t, then $s(t) = i$.

We define a random walk with the following single-step transition probabilities

$$P(s(t+1) = j|s(t) = i) = \frac{a_{ij}}{a_{i.}} = p_{ij} \text{ where } a_{i.} = \sum_{j=1}^{n} a_{ij}$$

In other words, to any state or node $i = s(t)$, we associate a probability of jumping to an adjacent node $j = s(t + 1)$, which is proportional to the weight $w_{ij}$of the edge connecting i and j. The transition probabilities only depend on the current state and not on the past ones (first-order Markov chain). Since the graph is totally connected, the Markov chain is irreducible, that is, every state can be reached from any other state. If this is not the case, the Markov chain can be decomposed into closed sets of states which are completely independent (there is no communication between them), each closed set being irreducible

If we denote the probability of being in state i at time t by $x_i(t) = P(s(t) = i)$ and define P as the transition matrix with entries $p_{ij} = P(s(t+1) = j|s(t) = i)$, the evolution of the Markov chain is characterized by

$$\begin{cases} x_i(0) = x_i^0 \\ x_i(t+1) = P(s(t+1) = i|s(t) = j)x_j(t) = \sum\limits_{j=1}^{n} p_{ij}x_j(t) \end{cases}$$

Or, in matrix form,

$$\begin{cases} x(0) = x^0 \\ x(t+1) = P^T x(t) \end{cases}$$

where T is the matrix transpose. This provides the state probability distribution $x(t) = [x_1(t), x_2(t), ..., x_n(t)]^T$ at time t once the initial probability density x^0 is known. It is well-know (see [41]) that such a Markov chain of random walk on a graph is time-reversible ($\pi_i P_{ij} = \pi_j P_{ji}$) with stationary probabilities given by

$$\pi_i = \frac{\sum\limits_{j=1}^{n} a_{ij}}{\sum\limits_{i,j=1}^{n} a_{ij}} = \frac{a_{i.}}{a_{..}}$$

This value is the probability of finding the Markov chain in state $s = i$ in the long-run behaviour.

This has important implications. For instance, it is known that all the eigenvalues of the transition matrix of a reversible Markov chain are real and distinct (see for instance [11])

For more details on Markov chains, the reader is invited to consult standard textbooks on the subject (e.g., [48], [35]).

4.3.3 Association of a weight to the keywords and the edges

In our recommendation system, the weight of a keyword in a profile represents the importance of this word for the user compared to all the other keywords.

We will note $k_{ij}$the weight of the keyword j in the user profile i. This weight must take into account the number of request made by the user on this keyword, as well as the number of visited links. So, the computation of the weight for a keyword j in the user profile i is given by

$$k_{ij} = f_{ij} + l_{ij}$$

where f_{ij} represents the number of request done by the user i on the keyword j, and l_{ik} corresponds to the number of links for the keyword j contained in the profile of the user i.

Now that we have defined the meaning of weight for a keyword, we can specify the concept of weight for an edge in our recommendation system.

The weight of an edge between two users (i.e., their personal agents) represents the similarities degree which expresses the resemblance of their profiles. The profiles of two agents are strongly similar, if the users related to these agents are interested by the same keywords and accept the same links for theses keywords.

We will note c_{ij} the similarities between the profiles of user i and user j, and the computation of this weight is characterized by

$$c_{ij} = sim(Pf_i, Pf_j) = \frac{\sum\limits_{k=1}^{r} k_{il}k_{jl}}{\sqrt{\sum\limits_{l=1}^{r_i} k_{il}^2 \sqrt{\sum\limits_{l=1}^{r_j} k_{jl}^2}}}$$

where Pf_i represents the list of the keywords contained in the user profile i and r corresponds to the number of keywords that Pf_i and Pf_j have in common.

4.3.4 Probability of absorption and average first-passage time/cost

In this section, we review three basic quantities that can be computed from the definition of the Markov chain, that is, from its transition probability matrix: the probability of absorption, the average first-passage time, and the average commute time. We also introduce the average first-passage cost which generalizes the average first-passage time. Relationships allowing to compute these quantities are derived in a heuristic way (see, e.g., [48] or [35] for a more formal treatment).

The probability of absorption

The probability of absorption $u(k|i)$is the probability that a random walker starting from some initial state i enters for the first time state $k \in S^a (S^a$ is a subset of states) before reaching any other state belonging to S^a. The states of the set S^a are "absorbing states" in the sense that, once the random walker reaches one of them, it stops walking. Thus, once an absorbing state has been reached, the probability of staying in it is 1. A recurrence relation allowing to compute the probability of absorption can be obtained by elementary probability theory (a proof is provided in [20]):

$$\begin{aligned}
u(k|i) &= p_{ik} + \sum_{j=1, j\neq k}^{n} p_{ij} u(k|j), \text{ for } i \notin S^a \text{ and } k \in S^a \\
u(k|k) &= 1, \text{ for } k \in S^a \\
u(k|i) &= 0, \text{ for } i, k \in S^a \text{ and } i \neq k
\end{aligned}$$

This system of linear equations can be solved by iterating the relations (the relaxation method, see [48]). The probability of absorption can also be obtained by algorithms that were developed in the Markov-chain community (see for instance [48]), or by using the pseudoinverse of the Laplacian matrix of the graph (see [20]).

The average first-passage time and average first-passage cost

A similar relationship can be established for the average first-passage time $m(k|i)$, which is defined as the average number of steps that a random walker, starting in state $i \neq k$, will take to enter state k for the first time [35]. More precisely, we define the minimum time until hitting state k as $T_{ik} = min(t \geq 0 | s(t) = k \text{ and } s(0) = i)$ for one realization of the stochastic process. The average first-passage time is the expectation of this quantity, when starting from state i: $m(k|i) = E[T_{ik}|s(0) = i]$.

In a similar way, we define the average first-passage cost, $o(k|i)$, which is the average cost incurred by the random walker starting from state i to reach state k. The cost of each transition is given by $c(j|i)$ (a cost matrix) for any states i, j. Notice that $m(k|i)$ can be obtained as a special case where $c(j|i) = 1$ for all i, j.

In [19] we derive recurrence relations for computing $m(k|i)$ and $o(k|i)$ by first-step analysis:

$$\begin{cases} o(k|k) = 0 \\ o(k|i) = \sum_{j=1}^{n} p_{ij} c(j|i) + \sum_{j=1, j \neq k}^{n} p_{ij} o(k|j), \text{ for } i \neq k \end{cases}$$

For $m(k|i)$, we obtain

$$\begin{cases} m(k|k) = 0 \\ m(k|i) = 1 + \sum_{j=1, j \neq k}^{n} p_{ij} m(k|j), \text{ for } i \neq k \end{cases}$$

These equations can be used in order to iteratively compute the first-passage times [35] or first-passage costs. The meaning of these formulae is quite obvious: in order to go from state i to state k, one has to go to any adjacent state $j$and proceed from there. Once more, these quantities can be obtained by algorithms developed in the Markov-chain community (see for instance [48]) or by using the pseudoinverse of the Laplacian matrix of the graph, as shown in [20].

4.4 External search: Gosis

This search provides links to the community by using an external search engine, GOSIS [19] - an aGent-Oriented Search Information System. Its role is to supply the community experience with new knowledge. GOSIS is an entity extern to the system which launches a search using only the user keywords.

GOSIS is a typical search engine supporting the creation of information sources that facilitate the integration of information coming from different heterogeneous sources. The sources may be conventional databases or other types of information, such as collections of Web pages. Figure 4.5 shows the GOSIS architecture in joint-venture in which the Broker plays the role of the joint-venture's management public interface, the Mediator plays the role of the joint-venture's management private interface and the associated actors are Matchmaker, Wrapper, Monitor and Multi-criteria Analyzer.

The Borker interacts with users and provides them an interface for searching information. The Mediator coordinates the tasks and controls the resource's share between the associated actors. Each associate can control himself on a local dimension and interact directly with others to exchange resources, data and knowledge.

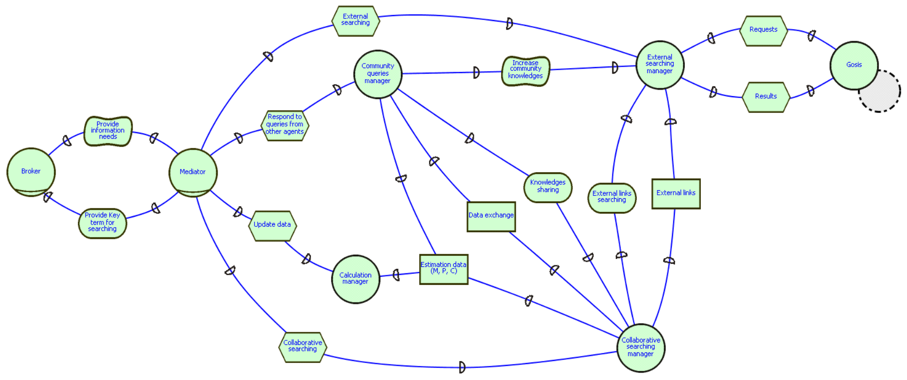

External searching
Community queries manager
Increase community knowledges
External searching manager
Requests
Gosis
Results
Provide information needs
Respond to queries from other agents
Broker
Mediator
Knowledges sharing
External links searching
External links
Provide Key term for searching
Update data
Data exchange
Calculation manager
Estimation data (M, P, C)
Collaborative searching
Collaborative searching manager

Figure 4.4: The complete architecture of personal agent

When a user wishes to send a request, it contacts the broker agent, which serves as an intermediary to select one or many mediator(s) that can satisfy the user information needs. Then, the selected mediator(s) decomposes the user's query into one or more subqueries to the sources, synthesizes the source answers and returns the answers to the broker.

If the mediator identifies repetitive user information needs, the information that may be of interest is extracted from each source, merged with relevant information from other sources, and stored as knowledge by the mediator. Each piece of information stored constitutes a materialized view that the mediator will have to maintain up-to-date.Connected to each information source is a wrapper and monitor agent. The wrapper is responsible for translating the subquery issued by the mediator in the native format of the source and translating the source response in the data model used by the mediator.

The monitor is responsible for detecting changes of interest (e.g, change which affects a materialized view) in the information source and reporting them to the mediator. Changes are then translated by the wrapper and sent to the mediator.

It may also be necessary for the mediator to obtain information concerning the localization of a source and its connected wrapper that are able to provide current or future relevant information. This kind of information is provided by the matchmaker agent, which then lets the mediator interact directly with the correspondent wrapper. The matchmaker plays the role of a "yellow-page" agent. Each wrapper advertises its abilities by subscribing to the yellow pages. The wrapper that no longuer wishes to be advertized can request to be unscribed.

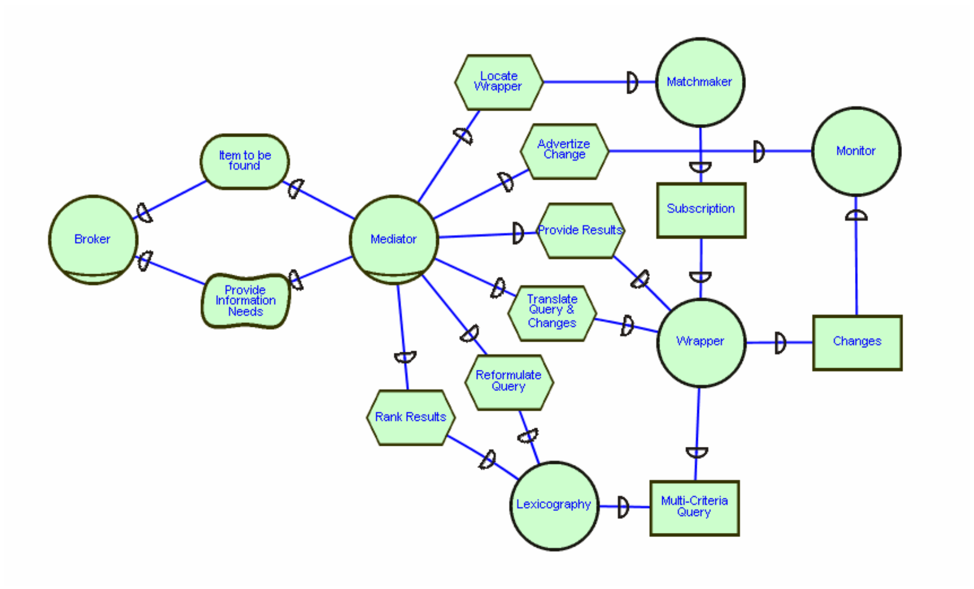


Figure 4.5: The GOSIS architecture in joint-venture

Finally, the multi-criteria analyzer can reformulate a subquery (sent by a mediator to a wrapper) through a set of criteria in order to express the user preferences in a more detailed way, and refine the possible domain of results.

We have developed WrapperSQLServer, WrapperText and WrapperGoogle for different types of information sources: SQL Server (structured information), text file (semi-structured information) and Internet (non-structured information, with the help of the Google search engine). By using this search engine, the external links are diversified and the community experience increases with these links. The recommendation system can collect the information from different and heterogeneous sources. That leads to a significant increase in the efficiency of the system.

Chapter 5

Reinforcement learning

This chapter introduces the Reinforcement Learning and the dilemma of the proportion of exploration and exploitation subjacent with this type of learning. Section 1 is an introduction to the Reinforcement Learning and presents the elements which compose it, its functioning and the type of problem which it solves. Section 2 exposes the dilemma between the exploitation and exploration.Section 3 introduces the notations, the standard stochastic shortest-path problem and the way we manage exploration. Section 4 describes our procedure for solving the stochastic shortest-path problem with exploration. The particular case where the graph of states is directed and acyclic is treated in Section 5. A short discussion of discounted problems is provided in Section 6, while the stochastic shortest path problem is discussed in Section 7. Some numerical examples are shown in Section 9.

5.1 Introduction

Reinforcement learning is an approach to artificial intelligence that emphasizes learning by the individual from its interaction with its environment. This contrasts with classical approaches to artificial intelligence and machine learning, which have downplayed learning from interaction, focusing instead on learning from a knowledgeable teacher, or on reasoning from a complete model of the environment. Modern reinforcement learning research is highly interdisciplinary; it includes researchers specializing in operations research, genetic algorithms, neural networks, psychology, and control engineering.

Reinforcement learning is learning what to do-how to map situations to actions-so as to maximize a scalar reward signal. The learner is not told which action to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward, but also the next situation, and through that all subsequent rewards. These two characteristics-trial-and-error search and delayed reward-are the two most important distinguishing features of reinforcement learning.

One of the challenges that arises in reinforcement learning and not in other kinds of learning is the tradeof between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover which actions these are it has to select actions that it has not tried before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploitation nor exploration can be pursued exclusively without failing at the task.

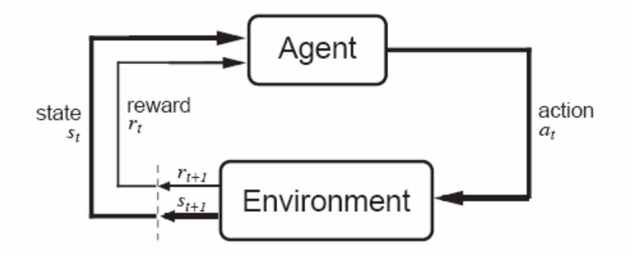


Figure 5.1: The Reinforcement Learning Framework

Modern reinforcement learning research uses the formal framework of Markov decision processes (MDPs). In this framework, the agent and environment interact in a sequence of discrete time steps, $t = 0, 1, 2, 3$,... On each step, the agent perceives the environment to be in a state, s_t, and selects an action, a_t. In response, the environment makes a stochastic transition to a new state, s_{t+1}, and stochastically emits a numerical reward, $r_{t+1} \in R$ (See Figure 5.1.) The agent seeks to maximize the reward it receives in the long run.

This framework is intended to capture in a simple way essential features of the problem of learning from interaction and thus of the overall problem of artificial intelligence. It includes sensation and action, cause and effect, and an explicit goal involving affecting the environment. There is uncertainty both in the environment (because it is stochastic) and about the environment (because the environment's transition probabilities may not be fully known). Simple extensions of this problem include incomplete perception of the state of the environment and computational limitations. Most theoretical results about reinforcement learning apply to the special case in which the state and action spaces are finite, in which case the process is called a finite MDP.

Reinforcement learning methods attempt to improve the agent's decision-making policy over time. Formally, a policy is a mapping from states to actions, or to probability distributions over actions. The policy is stored in a relatively explicit fashion so that appropriate responses can be generated quickly in response to unexpected states. The policy is thus what is sometimes called a "universal plan" in artificial intelligence, a "control law" in control theory, or a set of "stimulus-response associations" in psychology.

5.2 Balancing exploration and exploitation

Balancing **exploration** and **exploitation** is an important issue in reinforcement learning. Exploration aims to continually try new ways of solving the problem, while exploitation aims to capitalize on already well-established solutions. Exploration is of course especially important when the environment is changing, i.e. non-stationary. In this case, good solutions can deteriorate over time and better solutions can appear over time. Without exploration, the system only sends agents along the best path without exploring alternative paths. It would therefore not be aware of the changes, and its performances would inevitably deteriorate over time. One of the key features of reinforcement learning is that it explicitly considers the exploration/exploitation problem in an integrated way; this is one of its specificities [44], in contrast with, for instance, Markov decision processes [6], [7], [38].

When trying to solve a specific problem, exploration appears in essentially two different contexts

1. At the very first begining of the process, when everything is unknown, the agents start from some **initial state** and progressively discover the environment. As usual, we assume that the agents have some means to distinguish the states they actually visit and that they can label them and make an explicit graph, shared between agents, to represent the states and the transitions caused by actions. It is also assumed that if the agent does not know the costs of its actions, it learns this cost upon taking them. At this stage, no exploitation is performed since the agent has not reached any **goal** or **destination state** yet. The search is thus totaly exploratory and no exploitation occurs yet. Exploratory search strategies have been studied in the artificial intelligence litterature for years and are well-documented in standard artificial intelligence books, for instance [33], [42]. They differ on whether some prior knowledge of the problem is exploited (informed search) or not (uninformed search).

2. Once some goal states have been discovered, the agents begin to exploit these resources. At this stage, they may choose to exploit the resources without performing any exploration any more, or to balance exploitation and exploration. If the environment is stationary, there is of course no reason to continue exploration, since no better solution will ever be discovered. In this case, only exploitation is performed and the agents are routed only on the best path. On the other hand, if the environment is changing and no exploration is performed, the agents will only be aware

of the changes occuring on the shortest path to the goal state. Indeed, if no exploration occurs, the agent is always routed on the shortest path and will never perceive changes occuring outside of the shortest path. It might be, in this case, that other, better, solutions appear, which will be missed because of the lack of exploration. This problem is usually solved in an adhoc manner, by using some search heuristics. In this paper, we precisely propose a procedure that integrates both exploration and exploitation in a common framework by *optimaly balancing exploration and exploitation*, in this context. Notice however that we do not takle the problem of large search spaces in this work, while some ideas concerning this impmortant problem will be discussed in the conclusion.

Mathematically speaking, exploration corresponds to the association of a probability distribution to the set of admissible control actions in each state (choice randomization). If no exploration is performed, the agent is routed on the shortest path with probability one – he only exploits the solution. Exploration appears when this probability distribution is no more peaked on the best action: the agent is willing to sacrifice efficiency for exploration and the environment is continuously explored.

In this framework, we propose to measure exploitation from a given state by introducing the concept of **degree of exploration**, defined as the (Shannon) entropy [25] related to the probability distribution of the set of admissible actions on this state. This entropy value defines the degree of exploration linked to the state, provided a priori by the user. When the entropy is zero, no exploration is performed from this state, while when the entropy is maximal, a full, blind, exploration, with equal probability of choosing any action, is performed.

Then, we restate the exploration/exploitation dilemma as a global optimization problem: define the best exploration strategy (the probability distribution of choosing an action in a given state) that minimizes the expected cumulated cost from the initial state to the goal states while maintaining a fixed degree of exploration. This problem leads to a set of nonlinear equations defining the optimal solution. These equations, which are similar to Bellman's equations, can be solved by iterating them until convergence, which is proved for a stationary environment and a particular initialization strategy. They provide the action policy (the probability distribution of choosing an action in a given state) that minimizes the average cost from the initial state to the destination, or terminal, states, for a given degree of exploration.

Interestingly enough, when the degree of exploration is zero for all states, the nonlinear equations reduce to Bellman's equations for finding the shortest path from the initial state to the destination (solution) state. On the other hand, when the degree of exploration is maximum for all states, the nonlinear equations reduce to the linear equations allowing to compute the average cost for reaching the solution from the initial state in a Markov chain with transition probabilities equal to the inverse of the number of admissible actions for each state (full blind exploration).

The main drawback of this method is that it is computationally demanding since it relies on iterative algorithms like the value-iteration algorithm. We however show that if the graph of states is directed and acyclic, the equations can easily be solved by performing a single backward pass from the destination state. One way to obtain such a directed, acyclic, graph is to use Dial's procedure, popular in the transportation networks field [17].

Usually, one tackles this exploration/exploitation problem by readjusting the policy periodically and re-exploring up-to-now sub-optimal actions. Generally, the agent chooses a control action with a probability that is a function of the immediate cost associated with this action (usually, a softmax action selection; see [32], [44]). However, as shown in this paper, this strategy is sub-optimal because it does not integrate the exploitation issue within the exploration. Stated in other words, the exploration strategy does not take the exploitation into account.

In this document, for the sake of simplicity, we first concentrate on " *deterministic shortest-path problem*", as defined for instance in [7]. For this type of problem, any chosen control action drives deterministically the agent to a unique successor state. On the other hand, if the actions have uncertain effects, the resulting state is given by a probability distribution and one speaks about "*stochastic shortest-path problems*". In this case, a probability distribution on the successor states is introduced and must be estimated by the agents. Stochastic shortest-path problems are discussed in Section 5.7. We also briefly discuss "*discounted problems*" in Section 5.6 and we are currently working on "average cost per state" extensions.

5.3 Statement of the problem and notations

5.3.1 Statement of the problem

In this paper, we analyze reinforcement learning problems involving exploration. For the sake of simplicity, this paper will be focused on what is called "*deterministic shortest-path problems*"; stochastic shortest path problems and discounted problems are also briefly developped in Section 5.7 and 5.6.

In this kind of problem, it is assumed that during every state transition a bounded cost, given by $c(s_t, u(s_t)) < \infty$, is incurred where s_t is the state $\in \{1, 2, \ldots, n\}$ at time t and $u(s_t)$ is a control action selected from a set $U(s_t)$ of admissible actions, or choices, available in state s_t. The cost $c(s_t, u(s_t))$ can be viewed as the cost of performing action $u(s_t)$ given that the agent is in state s_t. It can be positive (penalty), negative (reward) or zero provided that no cycle exists whose total cost is negative. This is a standard requirement in shortest paths problems [14]; indeed, if such a cycle exists, then following this cycle an arbitrary large number of times will result in a path with an arbitrary small cost so that a best path is not properly defined. In particular, this implies that if the graph of the states is nondirected, all costs are nonnegative.

The **control action** $u(s_t)$ at any time t is determined according to a **policy** Π that maps every state k on the set $U(k)$ of admissible actions with a certain probability distribution, $\mathrm{P}(u(k)|s_t = k) = \pi_k(u(k))$, with $u(k) \in U(k)$. Thus the policy associates to each state k a probability distribution on the set of admissible actions $U(k)$: $\Pi \equiv \{\pi_k(u(k)), k = 1, 2, \ldots, n\}$ with each $u(k) \in U(k)$. For instance, if the admissible choices in state k are $U(k) = \{u_1^k, u_2^k, u_3^k\}$, the distribution $\pi_k(u(k))$ involves three values, $\pi_k(u_1^k)$, $\pi_k(u_2^k)$, and $\pi_k(u_3^k)$. Therefore the main difference with standard deterministic and stochastic shortest paths problems reside in the fact that we allow randomized choices in order to control exploration.

The action chosen by a policy is thus **randomized** in the sense that, for each state k, the agent chooses the next action with some probability $\pi_k(u(k))$. Randomization is introduced in order to guarantee a given **degree of exploration** that will be quantified by the entropy of the probability distribution; randomized choices are common in a variety of fields; for instance game theory (called mixed strategies in this context; see for instance [37]) or decision sciences [39] . We do not authorize, however, that an agent returns to the initial state; in other words, we do eliminate all the control actions that let us return to the initial state k_0.

Moreover, we assume (stochastic shortest path problems are discussed in Section 5.7) that once the action has been chosen, the next state s_{t+1} is known in a deterministic way, $s_{t+1} = f_{s_t}(u(s_t))$ where f is a one-to-one mapping between actions and resulting state. It provides the label of the next state s_{t+1}, given that we are in state s_t and we choose action $u(s_t)$. We assume that different actions lead to different states; otherwise just keep the action with lowest cost. This framework corresponds to a deterministic shortest path problem. Notice that, since each action automatically leads to a precise state, the introduction of control actions could be avoided by directly defining state transition probabilites. The more general formalism introduced here is justified by the fact that we will generalize this work to full stochastic problems for which, once a specific action is chosen, the transition to a state occurs according to some probability distribution (see Section 5.7). With this respect, notice that in the case of a deterministic shortest-path problems, only one family of probability distribution is introduced: the distribution of the control actions inside the states. In stochastic shortest path problems, an additional probability distribution will be introduced, governing the transition between the states after having chosen a particular action.

The goal is to minimize

$$V_\pi(k_0) = E_\pi \left[\sum_{t=0}^{\infty} c(s_t, u(s_t)) \, |s_0 = k_0 \right] \tag{5.1}$$

which is the **total expected cost** accumulated over an infinite horizon, given the initial (or source) state k_0. The expectation is taken on the policy, that is, on all the random variables $u(k)$ associated to the states k.

As in Bertsekas [7], we assume that there is a special cost-free **destination** or **goal state**; once the system has reached that state, it remains there at no further cost. We also consider a problem structure such that termination is inevitable, at least under an optimal policy. 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, linked to the fact that the destination state can be reached in a finite number of steps from any potential initial state; for a rigorous treatment, see [5], [7].

5.3.2 Computation of the total expected cost for a given policy

The essence of the problem is thus to reach the destination state $s = d$ with minimal expected cost from the initial state k_0; as already stated before, this problem is usually called the stochastic shortest-path problem. The deterministic shortest-path problem is obtained as a special case where, for each state, the control action is determined univocally.

This problem can be represented as a Markov chain where each original state and each action is a state. The destination state is then considered as an absorbing state with no outgoing link. In this framework, the problem of computing the expected cost (5.1) from any state k is closely related to the computation of the average first-passage time in the associated Markov chain [27]. 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. It can easily be generalized in order to take the cost of the transitions into account. By first-step analysis (see for instance [45]), we show in Appendix A.1 that, once the policy is fixed, $V_\pi(k)$ can be computed through the following equations

$$\begin{cases} V_\pi(k) = \sum\limits_{i \in U(k)} \pi_k(i)\,[c(k,i) + V_\pi(k'_i)], \\ \qquad \text{with } k'_i = f_k(i) \text{ and } k \neq d \\ V_\pi(d) = 0, \text{ where } d \text{ is the destination state} \end{cases} \tag{5.2}$$

Thus, in equation (5.2), k'_i is the state resulting from the application of control action $u(k) = i$ in state k.

(5.2) is a system of linear equations that can be solved by iterating the equations or by inverting the so-called fundamental matrix [27]; it is analogous to Bellman's equations in Markov decision processes.

Notice that the same framework can easily handle multiple destinations problems by defining one absorbing state for each destination. If the destination states are $d_1, d_2, \ldots, d_m$, we have

$$\begin{cases} V_\pi(k) = \sum\limits_{i \in U(k)} \pi_k(i)\,[c(k,i) + V_\pi(k'_i)], \\ \qquad \text{where } k'_i = f_k(i) \text{ and } k \neq d_1, d_2, ..., d_m \\ V_\pi(d_j) = 0 \text{ for all destination states } d_j,\ j = 1, ..., m \end{cases} \tag{5.3}$$

We now have to address two questions:

1. how do we control the randomized choices, i.e. exploration, and

2. how do we compute the optimal policy for a given degree of exploration.

5.3.3 Controlling exploration by fixing entropy at each state

Now that we have introduced the problem, we will explain how we manage exploration. At each state k, we define the **degree of exploration** E_k by

$$E_k = -\sum_{i \in U(k)} \pi_k(i) \log \pi_k(i) \tag{5.4}$$

which is simply the entropy of the probability distribution of the control actions in state k [16], [25]. E_k measures the uncertainty about the choice at state k. It is equal to zero when there is no uncertainty at all ($\pi_k(i)$ reduces to a Kronecker delta); it is equal to $\log(n_k)$, where n_k is the number of admissible choices at node k, in the case of maximum uncertainty, $\pi_k(i) = 1/n_k$ (a uniform distribution).

Furthermore, the **exploration rate** E_k^r at state k is defined as the ratio between E_k and its maximum value for state k, $E_k^r = E_k / \log(n_k)$. It takes its values in the interval $[0, 1]$. Fixing the entropy at a state sets the exploration level from this state; increasing the entropy increases exploration up to the maximal value, in which case there is no more exploitation since the next action is chosen completely at random, with a uniform distribution, without taking the costs into account.

5.4 Optimal policy under exploration constraints

5.4.1 Optimal policy and expected cost

We now turn to the problem of determining the optimal policy under exploration constraints. More precisely, we will seek the policy, that is, the probability distributions of the actions within the states, $\Pi \equiv \{\pi_k(u(k)), k = 1, 2, \ldots, n\}$, for which the expected cost $V_\pi(k_0)$ from initial state k_0 is minimal while maintaining a given degree of exploration on the states, E_k. Before going into the details, we make the following important preliminary observation: whatever the chosen policy, the system will remain a Markov chain and consequently Equation (5.2) remains valid for any admissible policy. The problem is thus to find the transition probability leading to the minimal expected cost. It can be formulated as a constrained optimization problem involving a Lagrange function.

In Appendix A.2, we derive the optimal probability distribution of control actions within a state k, which appears to be a logit distribution:

$$\pi_k(i) = \frac{\exp\left[-\theta_k \left(c(k,i) + V^*(k'_i)\right)\right]}{\sum_{j \in U(k)} \exp\left[-\theta_k \left(c(k,j) + V^*(k'_j)\right)\right]}, \tag{5.5}$$

where $k_i' = f_k(i)$ is a following state and V^* is the optimal (minimum) expected cost given by

$$\begin{cases} V^*(k) = \sum_{i \in U(k)} \pi_k(i) \left[c(k,i) + V^*(k_i')\right], \text{ with } k_i' = f_k(i) \text{ and } k \neq d \\ V^*(d) = 0, \text{ for the destination state } d \end{cases} \tag{5.6}$$

In Equation (5.5), θ_k must be chosen in order to satisfy

$$\sum_{i \in U(k)} \pi_k(i) \log \pi_k(i) = -E_k \tag{5.7}$$

for each state k and given E_k. It takes its values in $[0, \infty]$. This last equation guarantees a fixed exploration level (entropy) at each state. Of course if, for some state, the number of possible control actions reduces to one (no choice), no entropy constraint is introduced.

Equation (5.5) has a nice appealing interpretation: choose preferably (with highest probability) action i leading to state k_i' of lowest expected cost, including the cost of performing the action, $c(k,i) + V^*(k_i')$. Thus, the agent is routed preferably to the state which is nearest (in average) to the destination state.

Since Equation (5.7) has no analytical solution, θ_k must be computed numerically in terms of E_k. This is in fact quite easy since 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 [4]) or a simple binary search can efficiently find the θ_k corresponding to a E_k.

5.4.2 Computation of the optimal policy

Equations (5.5) and (5.6) suggest an iterative procedure very similar to the well-known value iteration algorithm for the computation of both the expected cost and the policy.

More concretely, we consider that agents are sent from the initial state and that they choose an action in each state k with probability distribution $\mathrm{P}(u(k)|s = k)$. The agent then performs the choosen action, say action i, and incurs the associated cost, $c(k,i)$ (which, in a non-stationary environment, may vary over time), 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 $\widehat{c}(k,i)$, $\widehat{\pi}_k(i)$ and $\widehat{V}(k)$.

1. **Initialization phase**:

    - Choose an initial policy, $\widehat{\pi}_k(i)$, $\forall i, k$, verifying the exploration rate constraints (5.7) and
    - Compute the corresponding expected cost until destination by using an iterative procedure for solving the set of linear equations (5.2). Any standard asynchronous iterative procedure (for instance, a Gauss-Seidel like algorithm) for computing the expected cost until absorption in a Markov chain could be used (see [27]).

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 $\widehat{\pi}_k(i)$, observe/update the current cost $\widehat{c}(k,i)$ for performing this action, and jump to the next state, k'_i.
   - Update the probability distribution for state k as:

   $$\widehat{\pi}_k(i) \leftarrow \frac{\exp\left[-\widehat{\theta}_k\left(\widehat{c}(k,i)+\widehat{V}(k'_i)\right)\right]}{\sum\limits_{j\in U(k)}\exp\left[-\widehat{\theta}_k\left(\widehat{c}(k,j)+\widehat{V}(k'_j)\right)\right]}, \tag{5.8}$$

     where $k'_i = f_k(i)$ and $\widehat{\theta}_k$ is set in order to respect the prescribed degree of entropy (see Equation (5.7)).
   - Update the expected cost asynchronously:

   $$\begin{cases} \widehat{V}(k) \leftarrow \sum\limits_{i\in U(k)} \widehat{\pi}_k(i)\,[\widehat{c}(k,i)+\widehat{V}(k'_i)], \text{ with } k'_i = f_k(i) \text{ and } k \neq d \\ \widehat{V}(d) \leftarrow 0, \text{ where } d \text{ is the destination state} \end{cases} \tag{5.9}$$

The convergence of these updating equations is proved in Appendix A.3. However, the described procedure is computationally demanding since it relies on iterative procedures like the value iteration algorithm in Markov decision processes.

Notice also that, while the initialization phase is necessary in our convergence proof, other simpler initialization schemes could also be applied. For instance, set initially $\widehat{c}(k,i) = 0$, $\widehat{\pi}_k(i) = 1/n_k$, $\widehat{V}(k) = 0$, where n_k is the number of admissible actions in state k; then proceed by directly applying updating rules (5.8) and (5.9). While convergence is not proved in this case, we observed that this updating rule works well in practice; in particular, we did not observe any convergence problem. This rule will be used in our experiments.

5.4.3 Some limit cases

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

Indeed, from Equations (5.8)-(5.9), if the parameter $\widehat{\theta}_k$ is very large, which corresponds to a near-zero entropy, the probability of choosing the action with the lowest value of $(\widehat{c}(k,i)+\widehat{V}(k'_i))$ dominates all the others. In other words, $\widehat{\pi}_k(j) \simeq 1$ for the action j corresponding to the lowest average cost (including the action cost), while $\widehat{\pi}_k(i) \simeq 0$ for the other alternatives $i \neq j$. Equations (5.9) can therefore be rewritten as

$$\begin{cases} \widehat{V}(k) \leftarrow \min\limits_{i \in U(k)} [\widehat{c}(k,i) + \widehat{V}(k'_i)], \text{ with } k'_i = f_k(i) \text{ and } k \neq d \\ \widehat{V}(d) \leftarrow 0, \text{ where } d \text{ is the destination state} \end{cases} \tag{5.10}$$

which are Bellman's updating equations for finding the shortest path to the destination state [6], [7].

On the other hand, when $\widehat{\theta}_k = 0$, the choice probabilities reduce to $\widehat{\pi}_k(i) = 1/n_k$, and the degree of exploration is maximum for all states. In this case, the nonlinear equations reduce to the 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, we then perform a "blind" exploration, for the choice probabilities.

Any intermediary 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.

5.5 Optimal policy for directed, acyclic, graphs

5.5.1 Definition of a directed acyclic graph (Dial's procedure)

In the case where the states form a directed acyclic graph, the procedure described in the previous section simplifies greatly. We will first describe a procedure, proposed by Dial [17], which allows to simplify the model in order to obtain an acyclic process. An acyclic process is a process for which there is no cycle, that is, one can never return to the same state. Dial [17] proposed the following procedure allowing to compute an acyclic process from the original one:

- First, compute the minimum cost from the initial state to any other state (by using for instance Bellman-Ford's algorithm [14]). The minimum cost to reach any state k from the initial state k_0 will be denoted as $d(k_0, k)$.
- Then, consider only actions leading to a state with a higher minimum cost from the initial state; all the other actions being labeled as "inefficient" and eliminated. Indeed, it is natural to consider that "efficient paths" should lead away from the origin. Thus, all the actions leading to a state transition $k \to k'$ for which $d(k_0, k') \leq d(k_0, k)$ are prohibited; the others are "efficient actions" and are allowed.

This results in an acyclic graph since all the states can be ordered by increasing $d(k_0, k)$ and only transitions from a lower $d(k_0, k)$ to a higher $d(k_0, k')$

are allowed. We consequently relabel the states by increasing $d(k_0, k)$, from 0 (initial state) to n (ties are ordered arbitrarily). The label of the destination state is still 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 $d(k, d)$ and eliminate actions according to this criterion. Other measures, different from the minimum cost, could be used as well.

5.5.2 Computation of the optimal policy

In this case, if $U(k)$ now represents the set of "efficient actions" for each state, the algorithm (5.8)-(5.9) reduces to

1. **Initialization phase**: $\widehat{V}(d) = 0$, 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:

$$\begin{cases} \widehat{\pi}_k(i) = \dfrac{\exp\left[-\widehat{\theta}_k\left(\widehat{c}(k,i) + \widehat{V}(k'_i)\right)\right]}{\displaystyle\sum_{j\in U(k)} \exp\left[-\widehat{\theta}_k\left(\widehat{c}(k,j) + \widehat{V}(k'_j)\right)\right]}, \\ \widehat{V}(k) = \displaystyle\sum_{i\in U(k)} \widehat{\pi}_k(i)\,[\widehat{c}(k,i) + \widehat{V}(k'_i)], \text{ with } k \neq d \end{cases} \tag{5.11}$$

   where $k'_i = f_k(i)$ and $\widehat{\theta}_k$ is set in order to respect the prescribed degree of entropy at each state (see Equation (5.7)).

The fact that the states have been relabeled by increasing $d(k_0, k)$ and the efficient actions always lead to a state with higher label guarantees that the values of the expected cost at states k'_i are known when using formula (5.11).

5.6 Discounted problems

In this section, instead of Equation (5.1), we consider **discounted problems**, for which there is a discount factor $0 < \alpha < 1$,

$$V_\pi(k_0) = E_\pi\left[\sum_{t=0}^{\infty} \alpha^t c(s_t, u(s_t))\,|s_0 = k_0\right] \tag{5.12}$$

The meaning of α is that future costs matter to us less than the same costs incurred at the present time. In this situation, we will see that there is no need to assume the existence of a destination state d.

Indeed, this problem can be converted to a stochastic shortest-path problem for which the analysis of previous sections holds (see [7]). Let $s = 1, \ldots, n$ be the states and consider an associated stochastic shortest-path problem involving, as for the original problem, the states $s = 1, \ldots, n$ plus an extra *termination, absorbing, state* t, with state transitions and costs obtained as follows: from a state k, when a 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 α, or t with probability $(1-\alpha)$. Once the agent has reached state t, it remains in this state forever with no additional cost. Here, absorbing state t plays the same role as destination state d in previous section and our previous stochastic shortest path model can easily be adapted to this framework, leading to the following optimality condition

$$V^*(k) = \alpha \sum_{i \in U(k)} \pi_k(i)\, [c(k,i) + V^*(k'_i)], \text{ with } k'_i = f_k(i),\, 0 < \alpha < 1 \tag{5.13}$$

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

$$\pi_k(i) = \frac{\exp\left[-\theta_k\left(c(k,i) + V^*(k'_i)\right)\right]}{\sum\limits_{j \in U(k)} \exp\left[-\theta_k\left(c(k,j) + V^*(k'_j)\right)\right]} \tag{5.14}$$

which are the corresponding optimality conditions for discounted problems.

5.7 Stochastic shortest path problems

We now consider **stochastic shortest path problems** for which, once an action has been performed, the transition to the next state is no longer deterministic but stochastic [7]. More precisely, when an agent chooses action i in state k, it jumps to state k' with a probability $\mathrm{P}(s = k' | u(k) = i, s = k) = p_{kk'}(i)$. Since there are now two different probability distributions, we will redefine our notations as

- The probability of choosing an action $u(k) = i$ within the state k, $\pi_k(i)$;
- The probability of jumping to a state $s = k'$ after having chosen the action $u(k) = i$ within the state k, $p_{kk'}(i)$.

By first-step analysis (see Appendix A.3), we easily find the recurence relations allowing to compute the expected cost $V_\pi(k)$, for policy Π:

$$\begin{cases} V_\pi(k) = \sum\limits_{i \in U(k)} \pi_k(i)\, [c(k,i) + \sum\limits_{k'=1}^{n} p_{kk'}(i)\, V_\pi(k')], \\ V_\pi(d) = 0, \text{ where } d \text{ is the destination state} \end{cases} \tag{5.15}$$

Furthermore, by defining the average cost when having chosen control action $u(k) = i$ in state k by $\overline{V}_\pi(k,i) = \sum_{k'} p_{kk'}(i) V_\pi(k')$, Equation (5.15) can be rewritten as

$$\begin{cases} V_\pi(k) = \displaystyle\sum_{i \in U(k)} \pi_k(i) \left[c(k,i) + \overline{V}_\pi(k,i)\right], \\ V_\pi(d) = 0, \text{ where } d \text{ is the destination state} \end{cases} \tag{5.16}$$

When proceeding as in Appendix A.2, the result is obtained by substituting $V_\pi(k')$ by $\overline{V}^*(k,i)$ in both 5.5 and 5.6), the optimal policy can easily be derived:

$$\pi_k(i) = \frac{\exp\left[-\theta_k\left(c(k,i) + \overline{V}^*(k,i)\right)\right]}{\displaystyle\sum_{j \in U(k)} \exp\left[-\theta_k\left(c(k,j) + \overline{V}^*(k,j)\right)\right]} \tag{5.17}$$

The additional difficulty here, in comparison with a deterministic problem, is that the probability distributions $p_{kk'}(i)$ have to be estimated on-line, together with the costs and the distribution of the randomized control actions [44].

5.8 Deterministic discounted problems

In this section, instead of Equation (5.1), we consider **discounted problems**, for which there is a discount factor $0 < \alpha < 1$,

$$V_\pi(k_0) = E_\pi\left[\sum_{t=0}^{\infty} \alpha^t c(s_t, u(s_t))\,|s_0 = k_0\right] \tag{5.18}$$

The meaning of α is that future costs matter to us less than the same costs incurred at the present time. In this situation, we will see that there is no need to assume the existence of a destination state d.

Indeed, this problem can be converted to a stochastic shortest-path problem for which the analysis of previous sections holds (see [7]). Let $s = 1, \ldots, n$ be the states and consider an associated stochastic shortest-path problem involving, as for the original problem, the states $s = 1, \ldots, n$ plus an extra *termination, absorbing, state* t, with state transitions and costs obtained as follows: from a state k, when a 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 α, or t with probability $(1-\alpha)$. Once the agent has reached state t, it remains in this state forever with no additional cost. Here, absorbing state t plays the same role as destination state d in our previous analysis, and the stochastic shortest path model can easily be adapted to this framework, leading to the following optimality condition

$$V^*(k) = \sum_{i \in U(k)} \pi_k(i) \left[c(k,i) + \alpha V^*(k'_i)\right], \text{ with } k'_i = f_k(i),\ 0 < \alpha < 1 \tag{5.19}$$

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

$$\pi_k(i) = \frac{\exp\left[-\theta_k\left(c(k,i) + V^*(k'_i)\right)\right]}{\displaystyle\sum_{j \in U(k)} \exp\left[-\theta_k\left(c(k,j) + V^*(k'_j)\right)\right]} \tag{5.20}$$

which are the corresponding optimality conditions for discounted problems.

5.9 Simulation results

Our experiments were performed on the graph shown in Figure 5.2. It is composed of 14 numbered nodes connected by edges of different weights, representing costs.

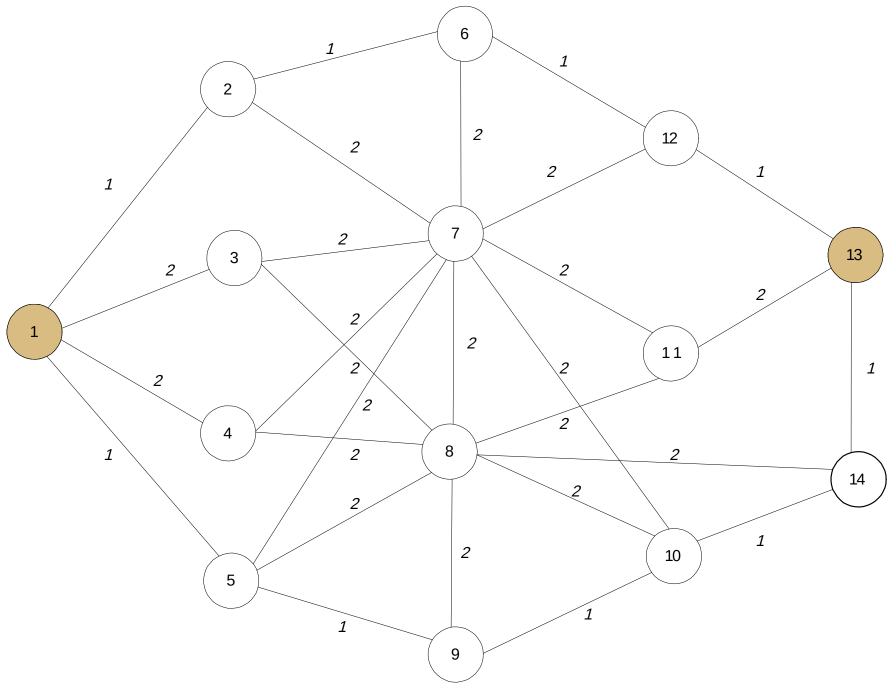


Figure 5.2: The graph used in our experiments. The initial (source) state is state 1 while the destination state is state 13. The initial costs are indicated on the edges.

The algorithm described in this paper was used in each experiment. It searches an optimal path in a given graph, by going from a starting 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.

Two simple experiments were performed, testing the algorithm's capacity to find the optimal paths in two different settings:

1. a static environment (i.e. an environment where the weight of the edges does not change over time);

2. a dynamic (non-stationary) environment.

In each experiment, the matrices of expected costs and transition probabilities are updated at each transition according to Equations (5.8) and (5.9) while a step represents the complete routing of a agent from its source to its destination. At the beginning of each simulation, the $\widehat{V}(k)$ and $\widehat{\pi}_k(i)$ are initialized according to the procedure introduced at the end of Section 5.4.2. The exploration rate is fixed to a specified value common to each state.

5.9.1 First experiment

Description

This experiment sends 15,000 agents through the network shown in Figure 5.2. The initial and destination nodes are node 1 and node 13, respectively. The costs of the external edges (i.e., the edges on the paths [1,2,6,12,13] and [1,5,9,10,14,13]) are initialized to 1, while the costs of all the other edges are initialized to 2.

The goal of this simulation is to observe the paths followed by the routed agents, in function of various values for the entropy. We simulate the agents transfer by assigning the same value of the exploration rate at each node. Each node corresponds to a state and the actions correspond to the choice of the next edge to follow. We repeat the experiment four times with, for each of them, a fixed value of the entropy corresponding to a percentage of the maximum entropy (i.e. exploration rates of 0%, 30%, 60% and 90%).

Results

The following graphs, displayed in Table 5.1, show the behaviour of the algorithm when varying the exploration degree. For each tested value, a graph (with the same topology as described above) is used to represent the results: the more an edge is used in the routing, the more its grey level and width are important.

The first graph shows the results for an exploration degree 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 used.

The second graph shows the followed paths for an exploration degree of 30%. The path [1,2,6,12,13], corresponding to the shortest path, is still the most used, while other edges not belonging to the shortest path are now also investigated. Moreover, we observe that the second shortest path, i.e., path [1,5,9,10,14,13], also routes a significant traffic here. Exploitation thus remains a priority while exploration is nevertheless present.

The third graph shows the paths used for an exploration rate of 60%. As for the previous experiment, the algorithm uses more and more edges that are not part of the shortest path, but still prefers the shortest path.

The last graph shows the result with an exploration rate of 90%. With such a value, the exploitation of the shortest path is no more the priority; exploration is now clearly favored over exploitation.

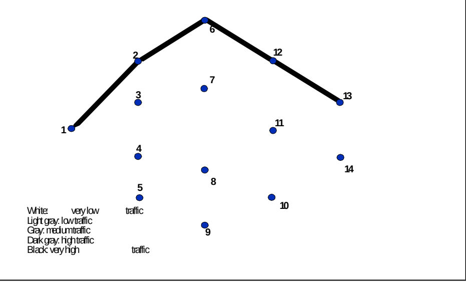
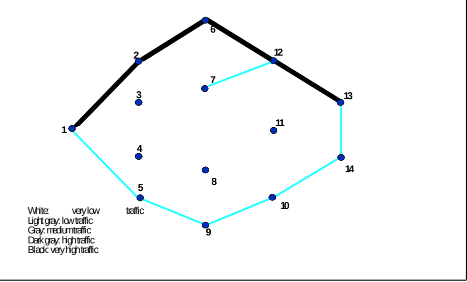
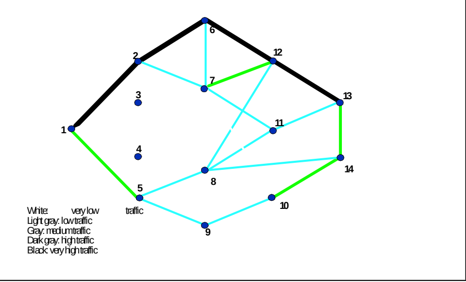
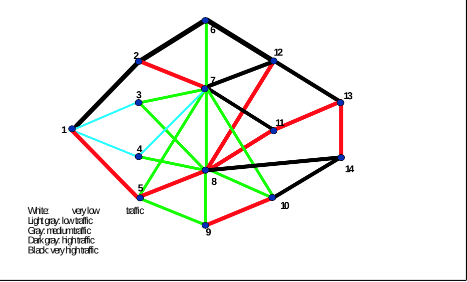
0% exploration rate	20% exploration rate
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 White: very low traffic Light gray: low traffic Gray: medium traffic Dark gray: high traffic Black: very high traffic	 1 2 3 4 5 6 7 8 9 10 11 12 13 14 White: very low traffic Light gray: low traffic Gray: medium traffic Dark gray: high traffic Black: very high traffic
40% exploration rate	**60% exploration rate**
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 White: very low traffic Light gray: low traffic Gray: medium traffic Dark gray: high traffic Black: very high traffic	 1 2 3 4 5 6 7 8 9 10 11 12 13 14 White: very low traffic Light gray: low traffic Gray: medium traffic Dark gray: high traffic Black: very high traffic

Table 5.1: Traffic through the edges for the four different exploration rates.

5.9.2 Second experiment

Description

The second experiment repeats the previous simulations while varying the environment, i.e., the cost of some edges.

Except the change in the environment, the context remains the same as in the first experiment, namely to convey 15,000 agents from a source node (1) to a destination node (13) by using the suggested algorithm. Thus, at each time step, an agent is sent from node 1 to reach node 13. At the beginning, the costs of the external edges (i.e. the edges on the paths [1,2,6,12,13] and [1,5,9,10,14,13]) are initialized to 3, while the cost of all the others (internal edges) are 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), the cost of all internal edges was set to 1, all other things being 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].

Results

Figure 5.3 contains five graphs representing the total costs of the transfer of all agents from their source to their destination, averaged on the first 7500 agents.

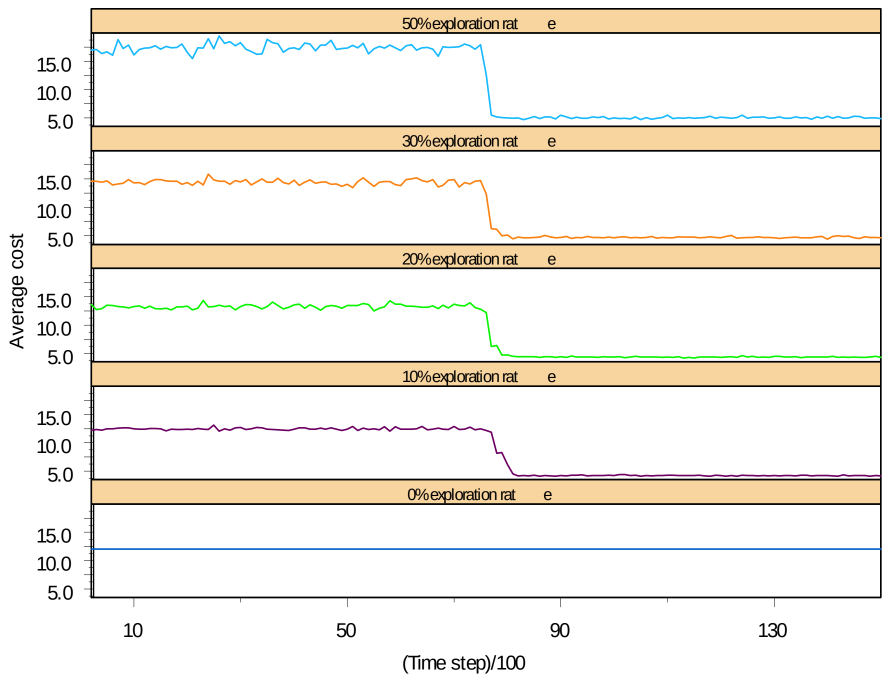


Figure 5.3: Average cost needed to reach destination node (13) from the initial node (1) in terms of time step for various exploration degrees.

We display the results for five levels of exploration rate: 0%, 10%, 20%, 30% and 50%.

The first graph (at the bottom) shows the total cost for an exploration degree of 0%. We observe that, despite the environment change (which leads to the creation of shorter paths with a total cost of 4), the cost remains equal to 12 throughout simulation. Once the algorithm has discovered the shortest path, it does not explore anymore. The system misses a path that has become (at time step 7500) much cheaper (in terms of cost) than the previous optimal path.

The four remaining 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 in environment – the total cost is updated from about 12 to 4. This is due to the fact that the algorithm carries out exploration and exploitation in parallel. This also explains the slight rise in total cost when increasing the value of the exploration degree.

Figure 5.4 shows the evolution of the average cost, the maximal cost as well as the minimal cost (computed on the first 7,500 agents) in terms of the exploration rate. Increasing the entropy induces a growth in the average total costs, which is natural since the rise in this entropy causes an enhancement in exploration, therefore producing a reduction in the exploitation and increasing the average

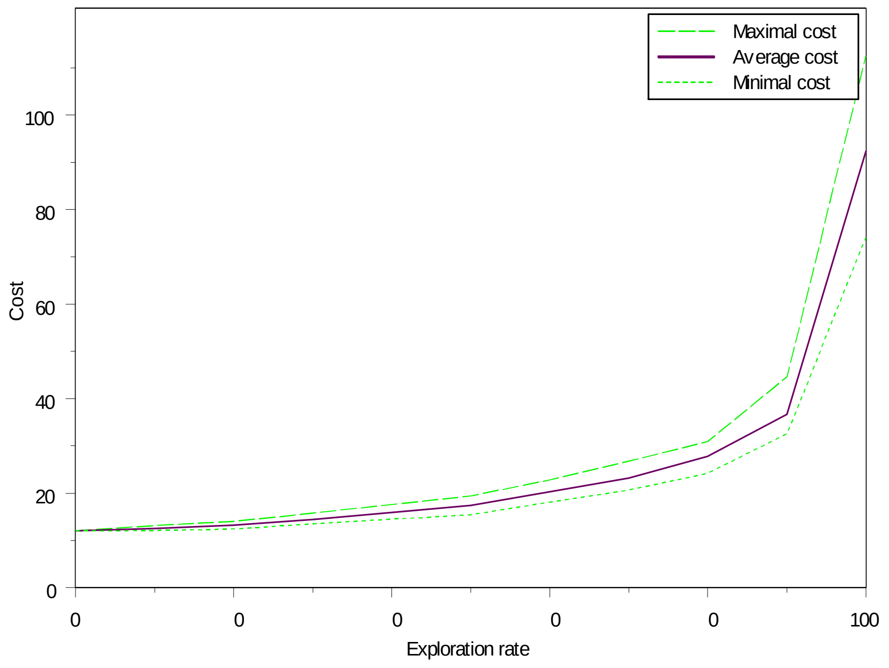


Figure 5.4: Average, maximal and minimal costs needed to reach the destination node (13) from the initial node (1) in terms of exploration rate.

total costs. We also notice that the difference between the minimum and the maximum total cost grows with the increase in entropy but remains quite weak.

5.9.3 Third experiment

Description

In this experiment, we simulate a very simple load balancing problem, and we will observe how the agents behave when using our algorithm. Load balancing is a technique used to spread work between many processes, computers, disks or other resources. The specification of the load balancing problem for this experiment is as follows. First, notice that we still use the graph defined previously, with the same cost on the edges (see Figure 5.2).

In this experiment, only two destination nodes are defined for illustration and ease of interpretation, but the approach generalizes to several nodes. Each destination node is a special node able to provide some resources (such as food) coveted by the agents crossing the network. The amount of resources present on each destination node is equal and limited, and the agents have to retrieve these resources in some optimal way (i.e., the agents must take a path of minimum cost and avoid the congestion).

We also take into consideration the number of remaining resources in the calculation of the average cost. Indeed, the number of resources of each destination node being limited, we introduce a "warning threshold" indicating that the resources are becoming scarce, and which will be translated into a decrease in reward. For the sake of simplicity, this critical threshold is equal for each destination node and is worth 10% of the starting resources. When an agent reaches a destination node, a checking of its remaining resources is carried out. Two cases can arise. In the first case, if the number of remaining resources is higher or equal to the threshold, the agent receives a positive reward (a negative cost) equal to the value of the threshold. In the second case, if the number of resources is lower than the threshold, the agent will perceive a lower reward, equal to the numbers of remaining resources (the cost is still negative). The introduction of this threshold results in a gradual decrease of the reward of a destination node when the number of its resources is below the fixed threshold.

Values of the simulation parameters were defined in the following way. Destinations nodes chosen for the experiment are nodes 11 and 14 with a number of resources being worth 5000 and a threshold equal to 500. 10,000 agents are sent from node 1 with the objective of gathering resources with minimal average cost.

Results

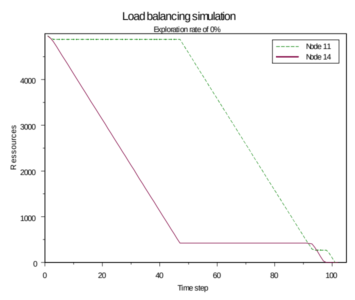
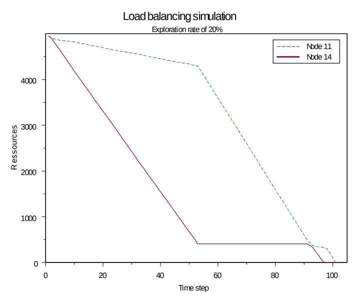
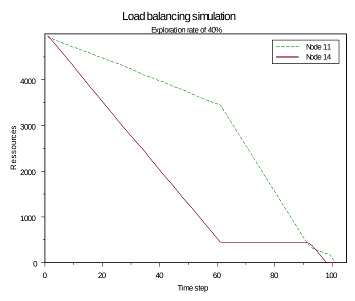
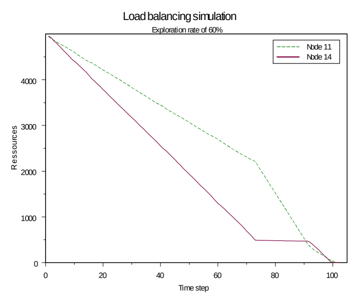
0% exploration rate	20% exploration rate
	
40% exploration rate	**60% exploration rate**
	

Table 5.2: Results of the load balancing problem for different exploration rate.

For this experiment, we use five exploration rates, precisely 0, 20, 40, 60 and 80%; Table 5.2 shows the resulting graphs for each rate (expected for 80%).

According to the fixed destination nodes (i.e., nodes 11 and 14) and the topography of the network, the destination with the shortest path is the node 14.

The first graph is the result of the simulation with an exploration rate of 0%. We observe that when no exploration is present, the algorithm proposes only the destination node 14 (i.e., the shortest path). Node 11 is exploited only when the resources of node 14 reach the critical point and when the cost of exploiting node 11 becomes less than exploiting node 14. In turn, the node 14 will be exploited only when the resources of node 11 reach the threshold. Notice that in the case of a zero-entropy, the two destination nodes are never exploited simultaneously, so that no "load balancing" really occurs. This could generate congestion on the shortest path if the resource is not transfered immediately.

The other graphs show the results of simulation with entropies of 20, 40, 60 and 80%. When exploration rate is raised, the two destination nodes are more and more exploited simultaneously and in proportions increasing with the entropy. The degree of load balancing is a function of the exploration rate.

Chapter 6

Conclusion

6.1 Recommendation system

We described an agent-based recommendation system dealing with the extraction of implicit knowledge from user behavior during web search. The knowledge produced from observations is used in order to suggest links or agents to group of people and to their personal agents. The main idea is that we do not express this knowledge in explicit form but use it for improving quality of further search sessions, including searches performed by new users. Personal agents produce results by asking another personal agent about links. Each agent has the learning capabilities that help to produce results even without interaction. With interaction, when the user performs a search already done by one of the community members, he/she does not do it \from scratch" but exploits his/her and the others' experience. This feature increases the search quality.

Our system is a solution to the problem of finding necessary information on the web. One of the main advantages of our approach is represented by the use of both search engine results and suggestions produced by community members. The multi-agent system mimics natural user behavior of asking someone who probably knows the answer. Finally, the process of producing suggestions is completely hidden from the user and therefore does not force him/her to perform additional actions.

There are possibilities of user profile improvements. Although for the present time it contains information only about acceptance and rejection of links obtained from an external search engine, it is possible to have rules for acceptance of links from the other agents. Finally, one of the further directions is to analyze the social relations and interactions between the users.

6.2 Trade-off between exploitation and exploration

In this work, we presented a model integrating both exploration and exploitation in a common framework. The exploration rate is controlled by the entropy of the choice probability distribution defined on the states of the system. 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 present approach is that it is computationally demanding since it relies on iterative procedures such as the value iteration algorithm.

Further work will aim to investigate alternative cost formulations, such as the "average cost per step". Moreover, since the optimal average cost is obtained by taking the minimum among all the potential policies, it can be shown that it defines a distance measure between the states of the process. Further work will investigate the properties of this distance, which generalizes the Euclidean commute time distance between nodes of a graph, as introduced and investigated in [43], [21].

6.3 Future works

In the first part of the document, we studied a system of recommendation allowing users to carry out research on the basis of key word. In the last part, we presented the reinforcement learning and especially a model integrating both exploration and the exploitation in a common framework. We now will analyse how the system of recommendation could be extended by using this model. The possibilities of integration presented in this part are for the moment only ideas that it will be necessary tested and analyzed thereafter.

A first use of the model with the system of recommendation could be considered on the level of collaborative filtering. Indeed, for collaborative filtering we use a calculation method to compute the transition probabilities varying not the degree of exploration and exploitation. Thus a first solution of integration would be the use of our calculation method, and the concept of entropy, to allow a rate of exploration which could vary according to the value of entropy.

A second use of our model with the system of recommendation would be to extend the system to make a Peer-to-Peer system. In this case, our model could be to use for the routing of information (information more significant than of simple links) solicited by the users. In a system peer-to-peer, the nodes having the resources required by the user can be numerous. Moreover, it is possible that certain nodes have suddenly disappeared or appeared, or that certain nodes progressively become more interesting to exploit than others (e.g., paths to reach these nodes are longer at the beginning, and become faster in the course of time considering the weak overload). The size and dynamic character of such a network could return the use of our model very interesting.

In this document we especially studied these two aspects in a very detailed way but independently, by giving only simple intuitions as for their integration. Other solutions of integration will be studied and analyzed in our future work.

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering vol. 17, no. 6*, 2005.

[2] W. R. Ashby. Principles of the self-organization system. *Principles of self-organization. Pergamon Press*, pages 255–278, 1962.

[3] M. Balabanovic and Y. Shoham. Fab: Content-based, collaborative recommendation. *ACM, vol. 40, no. 3*, pages 66–72, 1997.

[4] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming: Theory and algorithms.* John Wiley and Sons, 1993.

[5] D. P. Bertsekas. *Neuro-dynamic programming.* Athena Scientific, 1996.

[6] D. P. Bertsekas. *Network optimization: continuous and discrete models.* Athena Scientific, 1998.

[7] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2000.

[8] E. Blanzieri, P. Giorgini, P. Massa, and S. Recla. Implicit culture for multi-agent interaction support. In *Proceedings of Sixth International Conference on Cooperative Information Systems*, 2001.

[9] M. Bratman. *Intentions, Plans and Practical Reasoning.* Hardvard University Press, 1988.

[10] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, 1998.

[11] P. Bremaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues.* Springer-Verlag, 1999.

[12] M. Chau, D. Zeng, H. Chen, M. Huang, and D. Hendriawan. Design and evaluation of a multi-agent collaborativeweb mining system. *Decision Support Systems 35*, pages 167–183, 2003.

[13] L. Chen and K. Sycara. Webmate: A personal agent for browsingandsearch. In *Proceedings of the2ndInternational Conference on Autonomous Agents, ACM Press, New York, NY*, pages pp.132–139, 1998.

[14] N. Christofides. *Graph theory: An algorithmic approach.* Academic Press, 1975.

[15] M. Condliff, D. Lewis, D. Madigan, and C. Posse. Bayesian mixed-effects models for recommender systems. In *Proc. ACM SIGIR - Workshop Recommender Systems: Algorithms and Evaluation*, 1999.

[16] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley and Sons, 1991.

[17] R. Dial. A probabilistic multipath assignment model that obviates path enumeration. *Transportation Research*, 5:83–111, 1971.

[18] N. Dixon. *Common Knowledge.* Harvard Business School Press, 2000.

[19] S. Faulkner, M. Kolp, T. Nguyen, and A. Coyette. A multi-agent perspective on data integration architectural design. *Knowledge-Based Information and Engineering Systems (KES)*, 2004.

[20] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. A novel way of computing similarities between nodes of a graph, with application to collaborative recommendation. *Proceedings of the 2005 IEEE / WIC / ACM International Joint Conference on Web Intelligence*, pages 550–556.

[21] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Markov-chain computation of similarities between nodes of a graph, with application to collaborative filtering. *Submitted for publication, http://www.isys.ucl.ac.be/staff/francois/Articles/Saerens2005a.pdf*, 2005.

[22] F. Heylighen. The science of self-organization and adaptivity. *Knowledge Management Organizational Intelligence and Learning, and Complexity (in The Encyclopedia of Life Support Systems)*, 2003.

[23] F. Heylighen and C. Gershenson. The meaning of self-organization in computing. *IEEE Intelligent Systems, section Trends and Controversies - Self-organization and Information Systems*, 2003.

[24] iProspect. Search engine user attitudes survey. *Technical report, http://www.iprospect.com/premiumPDFs/iProspectSurveyComplete.pdf*, 2004.

[25] J. N. Kapur and H. K. Kesavan. *Entropy optimization principles with applications.* Academic Press, 1992.

[26] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time.

[27] J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* Springer-Verlag, 1976.

[28] M. Kolp, T. T. Do, and S. Faulkner. Introspecting agent-oriented design patterns. *S. K. Chang (Ed.) Advances in Software Engineering and Knowledge Engineering, World Scientific*, 2004.

[29] M. Kolp, S. Faulkner, and T. T. Do. Analysis styles for requirements enginnering: an organizational perspective. *S. K. Chang (Ed.) Advances in Software Engineering and Knowledge Engineering, World Scientific*, 2004.

[30] H. Lieberman. Letizia: An agent that assists web browsing. *International Joint Conference of Artificial Intelligent, Montreal*, 1995.

[31] F. Menczer. Complementing search engines with onlineweb mining agents. *Decision Support Systems 35*, pages 195–212, 2002.

[32] T. M. Mitchell. *Machine learning.* McGraw-Hill Compagnies, 1997.

[33] N. Nilsson. *Artificial intelligence: A new synthesis.* Morgan Kaufmann, 1998.

[34] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation.* Oxford University Press, 1995.

[35] J. Norris. *Markov Chains.* Cambridge University Press, 1997.

[36] J. Norris. *Markov Chains.* Cambridge University Press, 1997.

[37] M. J. Osborne. *An introduction to Game Theory.* Oxford University Press, 2004.

[38] M. Puterman. *Markov decision processes: discrete stochastic programming.* John Wiley and Sons, 1994.

[39] H. Raiffa. *Decision analysis.* Addison-Wesley, 1970.

[40] B. Ratitch and D. Precup. Using mdp characteristics to guide exploration in reinforcement learning. In *Machine Learning: ECML 2003, 14th European Conference on Machine Learning*, pages 313–324. Springer, 2003.

[41] S. Ross. *Stochastic Processes, 2nd Ed.* John Wiley and Sons, 1996.

[42] S. Russell and P. Norvig. *Artificial intelligence: A modern approach, 2nd Ed.* Prentice-Hall, 2003.

[43] M. Saerens, F. Fouss, L. Yen, and P. Dupont. The principal components analysis of a graph, and its relationships to spectral clustering. *Proceedings of the 15th European Conference on Machine Learning (ECML 2004). Lecture Notes in Artificial Intelligence, Vol. 3201, Springer-Verlag, Berlin*, pages 371–383, 2004.

[44] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, 1998.

[45] H. M. Taylor and S. Karlin. *An Introduction to Stochastic Modeling, 3th Ed.* Academic Press, 1998.

[46] S. Thrun. Efficient exploration in reinforcement learning. *Technical report,School of Computer Science, Carnegie Mellon University*, 1992.

[47] R. Turner, E. Turner, T. Wagner, T. Wheeler, and N. Ogle. Using explicit,apriori contextual knowledge in an intelligent web search agent. *Lecture Notes in Artificial Intelligence 2116*, pages 343–352, 2001.

[48] Y. Wei, L. Moreau, and N. Jennings. Recommender systems: A market-based design. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, ACMPress, Melbourn*, pages 600–607, 2003.

[49] M. Wooldridge and N. Jennings. Agent theories, architectures, and languages: A survey. *Artificial Intelligence 890. ECAI-94, Springer-Verlag*, 1994.

[50] B. Yu and M. Singh. Recommender systems: A market-based design. In *Proceedings of Eleventh International Conference on Information and Knowledge Management (CIKM02), SAIC Headquarters, McLean, Virginia, USA*, 2002.

Appendix A

Proof of the main results

A.1 Computation of the expected cost for a deterministic shortest path problem and a fixed policy

The goal is to compute the expectation of the cumulated cost over an infinite number of steps, under the policy Π:

$$V_\pi(k_0) = E_\pi \left[\sum_{t=0}^{\infty} c(s_t, u(s_t)) \, | s_0 = k_0 \right], \tag{A.1}$$

which thus represents the expected cost accumulated over an infinite horizon, when starting from some initial state k_0, before reaching the destination state d. The expectation is taken on the policy, that is, on all the random variables $u(k)$ associated to the states k.

We now derive the recurrence relation for computing $V_\pi(k_0)$ by first-step analysis (for a rigorous proof, see, for instance, [27]). The cumulated cost, $c_\pi(k_0)$, incurred during one particular walk (one realization of the random process), starting from state k_0 and entering for the first time destination state d, is

$$c_\pi(k_0) = \sum_{t=0}^{T_{k_0}} c(s_t, u(s_t)) \tag{A.2}$$

where T_{k_0} is the time (number of steps) taken to reach destination node. Now, without changing the problem, we can assume that the destination state d is an absorbing state, so that $\mathrm{P}(s_{t+1} = k | s_t = d) = \delta(d, k)$, where δ is the delta of Kronecker – just as if the process stops once state d has been reached. If we further assume $c(d, u) = 0$, $c_\pi(k_0)$ in Equation (A.2) can be rewritten as

$$c_\pi(k_0) = \sum_{t=0}^{\infty} c(s_t, u(s_t))$$

since once we have reached state k, we remain in k forever and $c(d,d) = 0$. Furthermore, we immediately observe that $c_\pi(d) = 0$.

Let us now compute the quantity of interest, $E[c_\pi(k_0)|s_0 = k_0]$, for $k_0 \neq d$:

$$V_\pi(k_0) = E[c_\pi(k_0)|s_0 = k_0] = E_\pi \left[\sum_{t=0}^{\infty} c(s_t, u(s_t)) \, |s_0 = k_0 \right] \tag{A.3}$$

$$= \sum_{i_0, i_1, \ldots} \mathrm{P}(u(s_0) = i_0, u(s_1) = i_1, \ldots |s_0 = k_0) \left[\sum_{t=0}^{\infty} c(s_t, u(s_t)) \right] \tag{A.4}$$

$$= \sum_{i_0, i_1, \ldots} \mathrm{P}(u(s_0) = i_0, u(s_1) = i_1, \ldots |s_0 = k_0)$$

$$\times \left[c(s_0, u(s_0)) + \sum_{t=1}^{\infty} c(s_t, u(s_t)) \right] \tag{A.5}$$

$$= \sum_{i_0} \mathrm{P}(u(s_0) = i_0 |s_0 = k_0)$$

$$\times \left\{ \sum_{i_1, i_2, \ldots} \mathrm{P}(u(s_1) = i_1, u(s_2) = i_2, \ldots |u(s_0) = i_0, s_0 = k_0) \right.$$

$$\left. \times \left[c(s_0, u(s_0)) + \sum_{t=1}^{\infty} c(s_t, u(s_t)) \right] \right\} \tag{A.6}$$

$$= \sum_{i_0} \mathrm{P}(u(s_0) = i_0 |s_0 = k_0) \left\{ \sum_{i_1, i_2, \ldots} \mathrm{P}(u(s_1) = i_1, u(s_2) = i_2, \ldots |s_1 = k_1) \right.$$

$$\left. \times \left[c(s_0, u(s_0)) + \sum_{t=1}^{\infty} c(s_t, u(s_t)) \right] \right\} \tag{A.7}$$

$$= \sum_{i_0} \mathrm{P}(u(s_0) = i_0 |s_0 = k_0) \left[c(s_0, u(s_0)) + V_\pi(k_1) \right] \tag{A.8}$$

where we used the Markov property (the future behaviour only depends on the knowledge of the last state) and $k_1 = f_{k_0}(i_0)$.

Therefore, since $V_\pi(d) = 0$, we can compute the expected cost at state k in terms of the expected cost at each state that can be reached from state k:

$$\begin{cases} V_\pi(k) = \displaystyle\sum_{i \in U(k)} \pi_k(i) \left[c(k,i) + V_\pi(k'_i) \right], \text{ with } k'_i = f_k(i) \text{ and } k \neq d \\ V_\pi(d) = 0 \end{cases} \tag{A.9}$$

These equations are very similar to the relations allowing to compute the average first-passage times in Markov chains [36]. Their meaning is quite obvious: in order to compute the total expected cost from state k to state d, one has to go to any reachable state k'_i and proceed from there.

A.2 Appendix: Determination of the optimal policy

We now turn to the problem of finding the optimal policy under entropy constraints. We therefore consider an agent starting from state k_0 and trying to reach the destination state by choosing an action according to a policy $\Pi \equiv \{\pi_1(u(1)), \pi_2(u(2)), \pi_3(u(3)), \ldots\}$. We already know (5.2) that the expected cost from any state, $V_\pi(k)$, is provided by the following equations

$$\begin{cases} V_\pi(k) = \sum\limits_{i \in U(k)} \pi_k(i)\,[c(k,i) + V_\pi(k'_i)], \\ \quad \text{with } k'_i = f_k(i) \text{ and } k \neq d \\ V_\pi(d) = 0, \text{ where } d \text{ is the destination state} \end{cases} \tag{A.10}$$

The goal here is to determine the policy that minimizes this expected cost, when starting from state k_0 and under entropy constraints (5.4). We therefore introduce the following Lagrange function, taking all the constraints into account,

$$L = V(k_0) + \sum_{k \neq d} \lambda_k \left(V(k) - \sum_{i \in U(k)} \pi_k(i)\,[c(k,i) + V(k'_i)] \right) \tag{A.11}$$

$$+ \lambda_d (V(d) - 0) + \sum_{k \neq d} \mu_k \left(\sum_{i \in U(k)} \pi_k(i) - 1 \right) \tag{A.12}$$

$$+ \sum_{k \neq d} \eta_k \left(\sum_{i \in U(k)} \pi_k(i) \log \pi_k(i) + E_k \right) \tag{A.13}$$

with $k'_i = f_k(i)$. Differentiating this Lagrange function in terms of the choice probabilities, $\partial L / \partial \pi_l(j)$, and equating to zero gives

$$-\lambda_l (c(l,j) + V(k'_j)) + \mu_l + \eta_l (\log \pi_l(j) + 1) = 0 \tag{A.14}$$

with $k'_j = f_k(i)$.

By multiplying Equation (A.14) by $\pi_l(j)$ and summing over $j \in U(l)$, we easily obtain

$$-\lambda_l V(l) + \mu_l + \eta_l (E_l + 1) = 0 \tag{A.15}$$

from which we deduce $\mu_l = \lambda_l V(l) - \eta_l (E_l + 1)$. Replacing μ_l by its expression in (A.14) provides

$$-\lambda_l (c(l,j) + V(k'_j) - V(l)) + \eta_l (\log \pi_l(j) - E_l) = 0 \tag{A.16}$$

Defining $\theta_l = -\lambda_l / \eta_l$ and extracting $\log \pi_l(j)$ from (A.16) gives

$$\log \pi_l(j) = -\theta_l (c(l,j) + V(k'_j) - V(l)) + E_l \tag{A.17}$$

or, equivalently,

$$\pi_l(j) = \exp\left[-\theta_l(c(l,j) + V(k'_j))\right] \exp\left[-\theta_l V(l) + E_l\right] \tag{A.18}$$

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 in the right-hand side of Equation (A.18):

$$\exp\left[-\theta_l V(l) + E_l\right] = \left[\sum_{j \in U(l)} \exp\left[-\theta_l(c(l,j) + V(k'_j))\right]\right]^{-1} \tag{A.19}$$

By replacing (A.19) in Equation (A.18), we finally obtain

$$\pi_l(i) = \frac{\exp\left[-\theta_l(c(l,i) + V(k'_i))\right]}{\sum\limits_{j \in U(l)} \exp\left[-\theta_l(c(l,j) + V(k'_j))\right]} \tag{A.20}$$

Finally, expressing the fact that each probability distribution has a given entropy,

$$-\sum_{i \in U(k)} \pi_l(i) \log \pi_l(i) = E_l,$$

allows us to compute the value of θ_l in terms of E_l. This defines the optimal policy; it can be shown that it is indeed a minimum.

Notice that differentiating the Lagrange function in terms of the $V(k)$ provides dual equations allowing to compute the Lagrange multipliers.

A.3 Convergence of the iterative updating rule

Before proving the convergence of the iterative updating rule (5.8)-(5.9), 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:

$$p_i = \frac{\exp[-\theta x_i]}{\sum\limits_j \exp[-\theta x_j]} \tag{A.21}$$

where θ 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 (5.8), in the proposed algorithm. This lemma shows that using this distribution (A.21) automatically reduces the linear form $\sum_i p_i x_i$.

To this end, let us introduce the Lagrange function

$$L = \sum_i p_i x_i + \lambda(\sum_i p_i \log p_i + E) + \mu(\sum_i p_i - 1) \tag{A.22}$$

and compute the differential of L in term of p_j, $\partial L/\partial p_j = 0$,

$$x_j + \lambda \log p_j + \lambda + \mu = 0 \tag{A.23}$$

By multiplying Equation (A.23) by p_j and summing the result, we obtain

$$\sum_j p_j(x_j + \lambda \log p_j + \lambda + \mu) = \sum_j p_j x_j - \lambda E + \lambda + \mu = 0 \tag{A.24}$$

Thus we have $(\lambda + \mu) = -\sum_j p_j x_j + \lambda E$. By substituting $(\lambda + \mu)$ by its value in (A.23), we find

$$\lambda \log p_i = -x_i + \sum_j p_j x_j - \lambda E \tag{A.25}$$

Now, by defining $\theta = \lambda^{-1}$, Equation (A.25) can be rewritten as

$$p_i = \exp[-\theta x_i] \exp[\theta \sum_j p_j x_j - E] \tag{A.26}$$

Summing Equation (A.26) over i and using the fact that $\sum_i p_i = 1$ allows us to compute the value of the second factor of Equation (A.26),

$$\exp[\theta \sum_j p_j x_j - E] = \left[\sum_j \exp[-\theta x_j] \right]^{-1} \tag{A.27}$$

Thus, we obtain the expected result from (A.26)-(A.27):

$$p_i = \frac{\exp[-\theta x_i]}{\sum_j \exp[-\theta x_j]} \tag{A.28}$$

where θ 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 (5.8)-(5.9). We will prove it by induction. First, we show that if the $\widehat{V}(k)$ have been decreasing (less than or equal to) up to the current time step, then the next update will also decrease $\widehat{V}(k)$. Then, since the first update necesarily decreases $\widehat{V}(k)$, the following updates will also decrease the value of $\widehat{V}(k)$. Now, since $\widehat{V}(k)$ is decreasing at each update and cannot be negative, this sequence must converge.

Thus, we first show that if all the $\widehat{V}(k)$ are decreasing up to the current time step t, then the next update will also decrease $\widehat{V}(k)$. Suppose the agent does visit state k at time step t, so that an update of the expected cost is computed. If we denote by $\widehat{V}^t(k)$ the estimate of the expected cost for state k and time step t, the update of $\widehat{V}(k)$ is given by

$$\begin{cases} \widehat{\pi}_k^t(i) = \dfrac{\exp\left[-\widehat{\theta}_k\left(\widehat{c}(k,i)+\widehat{V}_t(k'_i)\right)\right]}{\sum\limits_{j\in U(k)}\exp\left[-\theta_k\left(\widehat{c}(k,j)+\widehat{V}_t(k'_j)\right)\right]} \\ \widehat{V}^t(k) = \sum\limits_{i\in U(k)} \widehat{\pi}_k^t(i)\,[\widehat{c}(k,i)+\widehat{V}^t(k'_i)], \text{ with } k'_i = f_k(i) \text{ and } k \neq d \end{cases} \tag{A.29}$$

Now, denote by $\tau < t$ the time of the last visit of the agent at state k , and consequently the last update of $\widehat{V}(k)$ before time t. We easily find

$$\widehat{V}^t(k) = \sum_{i\in U(k)} \widehat{\pi}_k^t(i)\,[\widehat{c}(k,i)+\widehat{V}^t(k'_i)] \tag{A.30}$$

$$\leq \sum_{i\in U(k)} \widehat{\pi}_k^\tau(i)\;[\widehat{c}(k,i)+\widehat{V}^t(k'_i)] \tag{A.31}$$

$$\leq \sum_{i\in U(k)} \widehat{\pi}_k^\tau(i)\,[\widehat{c}(k,i)+\widehat{V}^\tau(k'_i)] = \widehat{V}^\tau(k) \tag{A.32}$$

The passage from Equation (A.30) to (A.31) is due to the preliminary lemma, while the passage from (A.31) to (A.32) results from the assumption that all the $\widehat{V}(k)$ are decreasing up to the current time step t.

Now, at the beginning of the exploration process ($t = 1$), the agent starts at node k_0 and updates $\widehat{V}^1(k_0)$ according to

$$\widehat{V}^1(k) = \sum_{i\in U(k)} \widehat{\pi}_k^1(i)\;[\widehat{c}(k,i)+\widehat{V}^1(k'_i)] \tag{A.33}$$

$$\leq \sum_{i\in U(k)} \widehat{\pi}_k^0(i)\,[\widehat{c}(k,i)+\widehat{V}^0(k'_i)] \tag{A.34}$$

since for every node $k'_i \neq k_0$, $\widehat{V}^1(k'_i) = \widehat{V}^0(k'_i)$ (no update has yet occurred). The assumption is thus clearly verified: all the $\widehat{V}(k)$ are decreasing up to the current time step $t = 1$. Consequently, the next update will certainly decrease $\widehat{V}(k)$, as well as all the subsequent updates.

Thus, if we define the decrease of $\widehat{V}(k)$ as $\Delta\widehat{V}^t(k) = \widehat{V}^{t-1}(k) - \widehat{V}^t(k) \geq 0$, since $\widehat{V}(k)$ cannot become less than the cost of the shortest path C, we have $\widehat{V}^\infty(k) = \widehat{V}^0(k) - \sum_{t=1}^{\infty}\Delta\widehat{V}^t(k) \leq C$ so that $\Delta\widehat{V}^t(k) \to 0$ as $t \to \infty$.

A.4 Computation of the expected cost for a stochastic shortest path problem and a fixed policy

Let us now compute the quantity of interest, $E[c_\pi(k_0)|s_0 = k_0]$, in the case of a stochastic shortest path problem, for $k_0 \neq d$:

$$\begin{aligned}
V_\pi(k_0) = E[c_\pi(k_0)|s_0 = k_0] &= E_\pi\left[\sum_{t=0}^{\infty} c(s_t, u(s_t))\,|s_0 = k_0\right] && \text{(A.35)}\\
&= \sum_{i_0,i_1,\ldots}\ \sum_{k_1,k_2,\ldots} \mathrm{P}(u(s_0) = i_0, u(s_1) = i_1, \ldots, s_1 = k_1, \ldots |s_0 = k_0)\\
&\times \left[\sum_{t=0}^{\infty} c(s_t, u(s_t))\right] && \text{(A.36)}\\
&= \sum_{i_0,i_1,\ldots}\ \sum_{k_1,k_2,\ldots} \mathrm{P}(u(s_0) = i_0, u(s_1) = i_1, \ldots, s_1 = k_1, \ldots |s_0 = k_0)\\
&\times \left[c(s_0, u(s_0)) + \sum_{t=1}^{\infty} c(s_t, u(s_t))\right] && \text{(A.37)}\\
&= \sum_{i_0}\sum_{k_1} \mathrm{P}_u(u(s_0) = i_0|s_0 = k_0)\ \mathrm{P}_s(s_1 = k_1|s_0 = k_0, u(s_0) = i_0)\\
&\times \left\{\sum_{i_1,i_2,\ldots}\ \sum_{k_2,k_3,\ldots} \mathrm{P}(u(s_1) = i_1, u(s_2) = i_2, \ldots, s_1 = k_1, \ldots |s_1 = k_1)\right.\\
&\times \left.\left[c(s_0, u(s_0)) + \sum_{t=1}^{\infty} c(s_t, u(s_t))\right]\right\} && \text{(A.38)}\\
&= \sum_{i_0,k_1} \mathrm{P}_u(u(s_0) = i_0|s_0 = k_0)\ \mathrm{P}_s(s_1 = k_1|s_0 = k_0, u(s_0) = i_0)\\
&\times [c(s_0, u(s_0)) + V_\pi(k_1)] && \text{(A.39)}
\end{aligned}$$

where we used the Markov property (the future behaviour only depends on the knowledge of the present state).

Therefore, since $V_\pi(d) = 0$, we can compute the expected cost at state k in terms of the expected cost at each state that can be reached from state k:

$$\begin{cases} V_\pi(k) = \displaystyle\sum_{i \in U(k)} \sum_{k'} \pi_k(i) p_{kk'}(i)\,[c(k,i) + V_\pi(k')],\ k \neq d \\ V_\pi(d) = 0 \end{cases} \tag{A.40}$$

which is the expected result.