Modelling and exploiting knowledge of the environment : A multi-agent approach to multi-goal pathfinding in ubiquitous environment

Oudom Kem

N°d'ordre NNT : 2018LYSEM023

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
l'Ecole des Mines de Saint-Etienne

Ecole Doctorale N° 488
Sciences, Ingénierie, Santé

Spécialité de doctorat : Informatique
Discipline : Intelligence artificielle

Soutenue publiquement le 29/10/2018, par :
Oudom KEM

Modélisation et Exploitation des Connaissances de l'Environnement : Une Approche Multi-Agents pour la Recherche d'Itinéraires Multi-Objectifs dans des Environnements Ubiquitaires

---

Devant le jury composé de :

MANDIAU, René	Professeur, Université de Valenciennes et du Hainaut-Cambrésis	Président
MONTICOLO, Davy	Maître de conférences HDR, Université de Lorraine	Rapporteur
OCCELLO, Michel	Professeur, Université Grenoble Alpes	Rapporteur
HERNANDEZ, Nathalie	Maître de conférences, Université de Toulouse	Examinatrice
BALBO, Flavien	Professeur, École des Mines de Saint-Étienne	Directeur de thèse
ZIMMERMANN, Antoine	Maître Assistant, École des Mines de Saint-Étienne	Co-encadrant

Spécialités doctorales	Responsables :	Spécialités doctorales	Responsables
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche	MATHEMATIQUES APPLIQUEES	O. Roustant, Maître-assistant
MECANIQUE ET INGENIERIE	S. Drapier, professeur	INFORMATIQUE	O. Boissier, Professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche	SCIENCES DES IMAGES ET DES FORMES	JC. Pinoli, Professeur
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche	GENIE INDUSTRIEL	N. Absi, Maitre de recherche
SCIENCES ET GENIE DE L'ENVIRONNEMENT	D. Graillot, Directeur de recherche	MICROELECTRONIQUE	Ph. Lalevée, Professeur

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

ABSI	Nabil	MR	Génie industriel	CMP
AUGUSTO	Vincent	CR	Image, Vision, Signal	CIS
AVRIL	Stéphane	PR2	Mécanique et ingénierie	CIS
BADEL	Pierre	MA(MDC)	Mécanique et ingénierie	CIS
BALBO	Flavien	PR2	Informatique	FAYOL
BASSEREAU	Jean-François	PR	Sciences et génie des matériaux	SMS
BATTON-HUBERT	Mireille	PR2	Sciences et génie de l'environnement	FAYOL
BEIGBEDER	Michel	MA(MDC)	Informatique	FAYOL
BLAYAC	Sylvain	MA(MDC)	Microélectronique	CMP
BOISSIER	Olivier	PR1	Informatique	FAYOL
BONNEFOY	Olivier	MA(MDC)	Génie des Procédés	SPIN
BORBELY	Andras	MR(DR2)	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR2	Génie Industriel	FAYOL
BRODHAG	Christian	DR	Sciences et génie de l'environnement	FAYOL
BRUCHON	Julien	MA(MDC)	Mécanique et ingénierie	SMS
CAMEIRAO	Ana	MA(MDC)	Génie des Procédés	SPIN
CHRISTIEN	Frédéric	PR	Science et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR1	Génie Industriel	CMP
DEBAYLE	Johan	MR	Sciences des Images et des Formes	SPIN
DEGEORGE	Jean-Michel	MA(MDC)	Génie industriel	Fayol
DELAFOSSE	David	PR0	Sciences et génie des matériaux	SMS
DELORME	Xavier	MA(MDC)	Génie industriel	FAYOL
DESRAYAUD	Christophe	PR1	Mécanique et ingénierie	SMS
DJENIZIAN	Thierry	PR	Science et génie des matériaux	CMP
DOUCE	Sandrine	PR2	Sciences de gestion	FAYOL
DRAPIER	Sylvain	PR1	Mécanique et ingénierie	SMS
FAUCHEU	Jenny	MA(MDC)	Sciences et génie des matériaux	SMS
FAVERGEON	Loïc	CR	Génie des Procédés	SPIN
FEILLET	Dominique	PR1	Génie Industriel	CMP
FOREST	Valérie	MA(MDC)	Génie des Procédés	CIS
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GARCIA	Daniel	MR(DR2)	Sciences de la Terre	SPIN
GAVET	Yann	MA(MDC)	Sciences des Images et des Formes	SPIN
GERINGER	Jean	MA(MDC)	Sciences et génie des matériaux	CIS
GOEURIOT	Dominique	DR	Sciences et génie des matériaux	SMS
GONDRAN	Natacha	MA(MDC)	Sciences et génie de l'environnement	FAYOL
GONZALEZ FELIU	Jesus	MA(MDC)	Sciences économiques	FAYOL
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR1	Génie des Procédés	SPIN
GUY	Bernard	DR	Sciences de la Terre	SPIN
HAN	Woo-Suck	MR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR1	Génie des Procédés	SPIN
KERMOUCHE	Guillaume	PR2	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	MR(DR2)	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	CR	Mécanique et ingénierie	FAYOL
MALLIARAS	Georges	PR1	Microélectronique	CMP
MOLIMARD	Jérôme	PR2	Mécanique et ingénierie	CIS
MOUTTE	Jacques	CR	Génie des Procédés	SPIN
NEUBERT	Gilles			FAYOL
NIKOLOVSKI	Jean-Pierre	Ingénieur de recherche	Mécanique et ingénierie	CMP
NORTIER	Patrice	PR1	Génie des Procédés	SPIN
O CONNOR	Rodney Philip	MA(MDC)	Microélectronique	CMP
OWENS	Rosin	MA(MDC)	Microélectronique	CMP
PERES	Véronique	MR	Génie des Procédés	SPIN
PICARD	Gauthier	MA(MDC)	Informatique	FAYOL
PIJOLAT	Christophe	PR0	Génie des Procédés	SPIN
PINOLI	Jean Charles	PR0	Sciences des Images et des Formes	SPIN
POURCHEZ	Jérémy	MR	Génie des Procédés	CIS
ROUSSY	Agnès	MA(MDC)	Microélectronique	CMP
ROUSTANT	Olivier	MA(MDC)	Mathématiques appliquées	FAYOL
SANAUR	Sébastien	MA(MDC)	Microélectronique	CMP
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
TRIA	Assia	Ingénieur de recherche	Microélectronique	CMP
VALDIVIESO	François	PR2	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	DR	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR0	Génie industriel	CIS
YUGMA	Gallian	CR	Génie industriel	CMP

Mise à jour : 03/02/2017

Dedicated to my mother and father

Acknowledgments

The last four years have been a challenging, yet rewarding experience. I would like to take this opportunity to express my gratitude to the people who have supported me throughout this journey.

I would like to express my sincere and utmost gratitude for my supervisors, Professor Flavien Balbo and Dr. Antoine Zimmermann, for their unlimited support, guidance, feedback, and constant trust in all my endeavours.

I wish to thank all my colleagues, Kamal Singh, Khadim Ndiaye, Radha Krishna Ayyalasomayajula, Omar Qawasmeh, Nicolas Cointe, El-Mehdi Khalfi, Dennis Diefenbach, José Miguel Giménez Garcia, Amro Najjar, Marie-Line Barneoud, Niloufare Sadr, Andrei Ciortea, Maxime Lefrançois, Gauthier Picard, Olivier Boissier, Xavier Serpaggi and everyone in the team for their support and friendship, making my stay in Saint-Étienne not only pleasant but memorable.

Many thanks go out to my family and friends for their support and understanding throughout this journey.

Contents

List of Figures

List of Tables

Introduction

In this information age, from intelligent artificial personal assistants to smart cities, we are experiencing the shifting towards Internet of Things (IoT), ubiquitous computing, and artificial intelligence. Cyber-physical entities are embedded in social environments of various scales from smart homes, to smart airports, to smart cities, and the list continues. This paradigm shift coupled with ceaseless expansion of the Web supplies us with tremendous amount of useful information and services, which creates opportunities for classical problems to be addressed in new, different, and potentially more efficient manners. Solutions to problems such as pathfinding, the problem of searching for a path between a start and a destination, and its variant multi-goal pathfinding in which a set of goals also need to be satisfied along the path, for instance, can be enriched and personalised by exploiting the available data and services.

Along with the new possibilities, we are, at the same time, presented with new constraints, problems, and challenges. Ubiquitous environments, the environments accommodating cybernetic, physical, and/or social entities such as sensors, connected objects, interactive devices, and people, are inherently dynamic and open. The state of the entities located in the environments may evolve over time. Some entities are mobile in the sense that their location may change. Entities may also dynamically enter or exit the environments. Consequently, the characteristics of the environments, namely the dynamics, the mobility, and the openness, need to be handled when solving problems in such environments. In this dissertation, we address the problem of multi-goal pathfinding in the context of ubiquitous environments. In this context, changes in the environment may influence the solution. A proposed path to a given problem may lose its validity when the state of the environment changes. In consequence, we address the aforementioned constraints in our approach to solving the problem.

In the literature, multi-goal pathfinding is commonly associated with the well-known Travelling Salesman Problem (TSP) whose objective is to determine the order in which the salesman should travel a set of cities to minimise the cost of travel. For this dissertation, however, rather than determining the order of places to travel, we aim at: (1) determining the places where each of the given goals can be satisfied and (2) finding the optimal path from a starting point to a destination in which all the goals can be satisfied in an order which we assume is given. The following scenario describes an example of a multi-goal pathfinding problem: A traveller, named Bob, arrives at an airport. Bob wants to find the path from his current location, which is the entrance of the airport, to the departure gate of his flight. Along the way to the gate, Bob has a set of goals he wants to achieve in the following order: get a trolley for his luggage, check in, buy a takeout for lunch, and find a waiting seat near a power socket to charge his laptop while waiting.

The characteristics of ubiquitous environments emphasise the need for up-to-date information. In such dynamic settings, it is impossible to have complete and accurate knowledge of the environments prior to the problem-solving process. Information about the environments and other knowledge required to solve a given problem need to be acquired from various sources including the cybernetic, physical, and/or social entities located in the environments during path computation to guarantee its validity. During path execution, more information needs to be retrieved to detect the changes that affect the current path and to adapt the path accordingly.

Let us refer to the previous example about the traveller Bob to convey the underlying motivation of using up-to-date information from various sources. Spatial information about the airport can be used to guide Bob to navigate within the airport. However, it is insufficient to determine which locations allow Bob to satisfy each of his goals or to determine the optimal path. Additional information about locations such as the type and the state of the locations in the airport is also needed. For instance, with such information, we are able to know that Bob can buy his take-out in the locations of type restaurant in the airport. Moreover, up-to-date information from sensors and smart objects is crucial in determining the optimal path for Bob. As an example, instead of suggesting Bob to go to the area where trolleys are stored, which is at the opposite direction of his gate, it is possible to locate an unoccupied trolley nearby that was left by other people thanks to the data from connected trolleys. We might suggest Bob to take an escalator instead of an elevator because we know that there are too many people in the queue waiting for the elevator or that the elevator is out of order, thanks to the feeds from the sensors of the elevators. In addition, information from the Web can be used to enhance Bob's travel experience. For example, reviews by travellers about the quality, the availability, and the food posted on the website of the restaurants enable us to choose the locations that are at Bob's best interests and preferences, rather than simply listing all the restaurants in the facility using static data.

Many algorithms have been proposed to solve pathfinding. The existing algorithms can be categorised into two groups: uninformed search algorithms and informed search algorithms. The former solve pathfinding by generating successor nodes and distinguishing a destination node from a non-destination node without using any additional knowledge of the search space. The latter exploit the knowledge of the problem to know if one node is more promising than the others in order to find solutions more efficiently. Each of these algorithms has its advantages and limitations from memory usage to optimality. Numerous attempts have been made to adapt the algorithms to address specific constraints such as reducing memory usage, supporting search in a partially known environment, or improving search efficiency. Multi-goal pathfinding, however, has been much less studied. Search algorithms for classical pathfinding find a path between two points, a start and a destination. They do not take into account the constraints on the points between the start and the destination. Therefore, they are not sufficient for solving multi-goal pathfinding problems as the solution to such problems consists of a path not only connecting the start and the destination, but also optimally passing by all the required points in a

given order. Existing approaches to multi-goal pathfinding are significantly limited in both numbers and efficiency. To the best of our knowledge, none address multi-goal pathfinding in the context of ubiquitous environments where the constraints such as dynamics, mobility, and openness need to be taken into account.

Therefore, in this dissertation, we propose an approach to solve multi-goal pathfinding that is able to handle the constraints of ubiquitous environments. In this approach, the information required for path computation is acquired from various *resources* such as the Web and cybernetic, physical, and/or social entities located in the environment, during the computation to provide up-to-date solutions. During path execution, changes in the environment that may influence the current solution are detected, and necessary actions performed to adapt the solution accordingly. To have a precise understanding of the context in which we aim at solving multi-goal pathfinding, in the following section, we provide a concise description of ubiquitous environments and their constraints.

Ubiquitous environments

Mark Weiser introduced ubiquitous computing as a method of enhancing the use of computers by rendering them available throughout the physical environment, but making them effectively invisible to the user [Weiser 1993]. The idea was further described by Weiser as "a field that speculates on a physical world richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives and connected through a continuous network" [Weiser 1999]. The concept of ubiquitous environments or smart environments has evolved from ubiquitous computing. Recent development and applications of Internet of Things and ubiquitous computing lead to an increasing number of ubiquitous environments. Such environments range from smart buildings, such as airports and train stations, to smart cities.

In the aim of solving multi-goal pathfinding in ubiquitous environments, we have identified the aspects of the environments that need to be taken into account when solving the problem. In the context of this work, we consider a ubiquitous environment as an environment that accommodates cybernetic, physical and/or social entities. We define different types of entities as follows:

- A physical entity refers to an entity that occupies a physical space. For instance, in a ubiquitous environment, a room, a restaurant, a shop, or an office can be considered a physical entity.

- A cyber-physical entity refers to a physical entity that is equipped with a cybernetic attribute. In other words, it is a physical entity that is at the same time a digital entity, and is connected to another entity or to other entities via some communication networks. For example, sensors, smart objects, or connected trolleys located in an environment can be considered as cyber-physical entities.

- A social entity refers to an entity that possesses social abilities, which are the abilities to interact with another entity or other entities. People are an example of social entities.

- A cyber-social entity refers to an entity that possesses cybernetic and social abilities, but does not have a physical form. An example of a cyber-social entity is a social software agent. The agent is able to communicate with other entities, but it remains a software entity, and thus has no physical form.

- A cyber-physical-social entity refers to a physical entity that possesses both cybernetic and social capabilities. An example of such an entity is a physical robot that is capable of communicating with other entities.

We can consider the smart airport mentioned in the previous example as a ubiquitous environment. A smart airport is a ubiquitous space as it accommodates cybernetic, physical and/or social entities. Cyber-physical entities such as sensors, information screens, and smart lights are installed. Social entities including travellers and airport personnel are actively participating in the environment. These entities are the prosumers (i.e., producer and consumer) of the information in the environment in the sense that they consume information from other entities, and also produce information back to the environment. For instance, a traveller, as a social entity, consumes information from sensors, information screens, and the airport personnel in order to facilitate their travel experiences. At the same time, they produce information back to the environment in terms of reviews on services, which can be useful for other travellers and the organisers of the airport.

Thus, a ubiquitous environment is a complex socio-technical space that is both open and dynamic. Cybernetic, physical, and/or social entities in the environment can be mobile, in the sense that their location may change over time. Moreover, the state of the entities may also evolve over time. Referring back to the airport example, mobile entities such as trolleys and vehicles (e.g., indoor shuttles or cars) can move from one location to another inside the environment. Entities such as elevators may change their state from available at time t_0 to occupied at time t_1. Furthermore, in such an open environment, entities are able to freely enter and exit the environment. Actions performed by the entities may also influence the state of the environment. For example, a traveller carrying out an activity to satisfy their goal of using a check-in machine changes the state of the machine from available to occupied. Solving multi-goal pathfinding in such a dynamic environment necessitates up-to-date information, not only about the spatial dimension of the environment, but also the state of the entities situated in the environment. To solve multi-goal pathfinding and address the complexity of ubiquitous environments, in this dissertation, we propose a solution that responds to the research questions described in the following section.

Thesis statement and objectives

The aim of this dissertation is to propose a multi-agent approach that is capable of solving multi-goal pathfinding in ubiquitous environments by exploiting the knowledge from pertinent sources such as the Web and the cybernetic, physical, and/or social entities located in a given environment. The thesis statement of this dissertation is as follows:

Given the dynamic, mobile, and open characteristics inherent in ubiquitous environments as well as the needs for up-to-date information, exploiting information from pertinent sources such as cybernetic, physical and/or social entities as well as from the Web to use in solving multi-goal pathfinding can contribute to an improvement in the quality of the solutions in terms of coverage, adaptability and personalisation.

In the proposed approach, we address the following research questions:

Research question 1. *Obtaining pertinent information: How to acquire the information necessary for solving a multi-goal pathfinding problem?* Solving a multi-goal pathfinding problem requires various types of information such as the spatial structure of the environment and the up-to-date information about the state of the environment. Furthermore, in a given environment, information can be acquired from different entities. In the Web, data and services are distributed in various sources. Identifying the sources of information to use for solving a specific problem is one of the core challenges in our approach. To address this question, we propose a knowledge model for describing an environment by integrating the spatial dimension and the cyber-physical-social dimension of the environment with the knowledge regarding the information sources that are relevant to each part of the environment. Such an abstraction provides the knowledge necessary for solving multi-goal pathfinding problems, and enables us to determine which information sources should be used to retrieve the necessary data to solve a problem.

Research question 2. *Handling latency: How to address the latency of accessing data sources and transferring data from the sources?* Acquiring up-to-date information from relevant sources is essential to our approach due to the dynamics of the environment. However, accessing resources and transferring data from the resources creates latency that affects the efficiency of the problem-solving process. To address this overhead, we propose a collaborative multi-agent search model that can be applied to search algorithms in order to asynchronise the resource-accessing process, and thus mitigates the impact of latency on the search process. Furthermore, the proposed search model exploits the structure of the problem to improve search performance.

Research question 3. *Beyond the classical pathfinding: How to solve multi-goal pathfinding?* Many solutions have been developed to address the classical pathfinding. Pathfinding with an ordered set of goals to satisfy is much less investigated.

To address multi-goal pathfinding, we propose an approach that transforms a multi-goal pathfinding problem into 2 connected pathfinding problems, which creates a multi-layered search space. Then, a search algorithm is applied on each layer to find an optimal path that connects the start to the destination and that allows the goals to be satisfied in the given order.

Research question 4. *Dynamics, mobility, and openness: How to address the dynamic state of the environment during path computation and path execution?* The state of ubiquitous environments may evolve over time. The changes may affect the solution during path computation as well as during path execution in which the path is being followed. In addition, the action of satisfying a goal may influence the state of the environment. Therefore, the approach that attempts at solving multi-goal pathfinding in such environments needs to be capable of adapting the initial solution according to the changes. To address the dynamics, our approach continuously detects the changes in the environment that affect the solutions, and incrementally update the solutions accordingly.

Relevant issues
In addition to the aforementioned research questions, there are 2 other issues that need to be addressed. The first issue is concerned with the evolution of the environment: *How to detect the changes in the environment and update the description of the environment accordingly?* The description of the environment is an integral part of our approach to solving multi-goal pathfinding. Therefore, the description has to be an up-to-date representation of the environment. In the literature, there is a body of work that addresses the changes of the environment in the context of Internet of Things. For instance, in [Ciortea 2015], the authors propose a multi-agent-based socio-technical network to manage the complexity of cyber-physical-social entities. Building upon such work, we are able to detect the changes in the environment and update the description of the environment accordingly.

The second issue is the heterogeneity of data and resources: *How to access different types of resources and use data of heterogeneous formats and structures?* The mechanisms to access resources may vary from one resource to another. Data acquired from various resources are of heterogeneous formats and structures. This problem needs to be addressed for our approach to work. Currently, there is a body of work that addresses the heterogeneity problem. In our approach, we employ the existing solutions to abstract away the issue of data heterogeneity and interoperability.

Contributions

Our contributions in addressing multi-goal pathfinding in ubiquitous environments are as follows:

- We proposed a knowledge model for describing a ubiquitous environment by integrating its spatial dimension, the cyber-physical-social entities it contains,

and the information about the resources relevant to the environment. In addition, we incorporate the notion of goals in the proposed knowledge model. Such knowledge enables us to determine at which location a goal can be satisfied. (*Research question 1*)

- We developed a collaborative multi-agent search model for adapting search algorithms to asynchronise the process of accessing resources to mitigate the consequential overheads during path computation and path execution. Furthermore, the search model exploits the knowledge of the search space to improve the efficiency of the search process. (*Research question 2*)
- We proposed a multi-agent approach to solve multi-goal pathfinding, especially in the context of ubiquitous environments. To address the dynamics, mobility, and openness of ubiquitous environments, we propose a mechanism that continuously updates the solution according to the changes in the environments. (*Research question 3 & 4*)

Dissertation structure

This dissertation is divided into 4 parts:

- In Part I, we investigate the literature to establish the state of the art in multi-goal pathfinding as well as its application in traveller information systems in order to identify the gap in the literature and the related work that can be used or adapted to support our approach.
  - In Chapter 1, we study the existing approaches in the literature that address pathfinding and multi-goal pathfinding.
  - In Chapter 2, we investigate our field of application, which is smart mobility and intelligent traveller information systems, particularly in ubiquitous environments.
- In Part II, we present our contributions.
  - In Chapter 3, we describe the knowledge model for abstracting a ubiquitous environment to provide the necessary knowledge for solving multi-goal pathfinding by integrateing the spatial, cybernetic, physical, and social aspects of the environment with the notion of goals.
  - In Chapter 4, we present the collaborative multi-agent search model that aims at mitigating latency.
  - In Chapter 5, we describe our approach to solving multi-goal pathfinding in ubiquitous environments. Then, we present the mechanism for continuously updating the solution according to the changes in the environment during path computation and path execution.
- In Part III, we focus on the evaluation of the approach.

  - In Chapter 6, we show the empirical results of our experiments, and provide the evaluation of our approach.

- In Part IV, we conclude the dissertation.

  - In Chapter 7, we provide the summary of our work and present the directions for our future work.

Part I

STATE OF THE ART

Chapter 1

Pathfinding and Multi-goal Pathfinding

Contents

Multiple variants of pathfinding have been addressed in the literature. Such variation includes, but not limited to, single-agent pathfinding, multi-agent pathfinding, pathfinding in dynamic environments, and pathfinding with incomplete information. A significant number of search algorithms and strategies have been proposed to address the constraints specific to each variant. Multi-goal pathfinding, a variant of pathfinding, goes beyond the classical pathfinding problems to incorporate the constraints on goals to satisfy along the path. In this dissertation, we focus on multi-goal pathfinding, particularly in ubiquitous environments where the constraints including dynamics, mobility, and openness also need to be taken into account.

The aim of this chapter is to investigate the existing approaches to pathfinding and multi-goal pathfinding that address the aforementioned constraints. The purpose of such an investigation is to determine the gap in the literature as well as to identify the work that can be adapted to support our approach to solving multi-goal pathfinding in ubiquitous environments.

The rest of this chapter is organised as follows. First, we present the problem of pathfinding and its formalism. Second, we investigate multi-goal pathfinding problems in the literature. Third, we discuss the classical search algorithms. Fourth, we analyse the existing search algorithms for dynamic environments. Fifth, we review

parallel search algorithms with regard to the latency issue. Finally, we provide an overall analysis and a conclusion of the chapter.

1.1 Pathfinding

Pathfinding consists in searching through a given space to find a path between a starting point and a destination. A pathfinding problem can be abstracted as a search problem. To provide background for the rest of the chapter, we adapt the formal definitions of the weighted state space search defined in [Ghallab 2016] and [Fukunaga 2017] to formally describe a pathfinding problem.

Definition 1 (Pathfinding problem) *A pathfinding problem or a weighted state space problem $P = (S, s_o, T, A, w)$ is defined by a set of states S, an initial state $s_o \in S$, a set of goal states $T \subset S$, a finite set of actions $A = a_1, ..., a_m$ where each $a_i : S \to S$ transforms a state into another state, and a cost function $w : A \to [0, \infty)$.*

Although w is called the cost function, its meaning is arbitrary: it may represent monetary cost, time, or something else that one might want to minimise. The cost of a path consisting of actions $a_1, ..., a_n$ is defined as $\sum_{i=1}^{n} w(a_i)$.

Definition 2 (Solution) *A solution $\pi = (a_1, ..., a_k)$ is an ordered sequence of actions $a_i \in A, i \in 1, ..., k$ that transforms the initial state s_o into one of the goal states $t \in T$; that is, there exists a sequence of states $u_i \in S, i \in 0, ..., k$ with $u_o = s_o, u_k = t$, and u_i is the outcome of applying a_i to $u_{i-1}, i \in 1, ..., k$. A solution from s_o to a given goal state t is optimal if its weight is minimal among all paths between s_o and t.*

Definition 3 (Pathfinding problem graph) *A problem graph $G = (V, E, s_o, T, w)$ for the pathfinding problem $P = (S, A, s_o, T, w)$ is defined by $V = S$ as the set of nodes, $s_o \in S$ as the initial node, T as the set of goal nodes, $E \subset V \times V$ as the set of edges that connect nodes to nodes with $(u, v) \in E$ if and only if there exists an $a \in A$ with $a(u) = v$, and w is extended to $E \to [0, \infty)$. The graph is uniformly weighted if $w(u, v)$ is constant for all $(u, v) \in E$. The weight or cost of a path $\pi = (v_o, ..., v_k)$ is defined as $w(\pi) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$.*

The integral elements of an approach to solving pathfinding are a graph representation of the map or the environment, a search algorithm, and depending on the algorithm, a heuristic function. The generation of the representative graph of the environment is generally based on the spatial information of the environment [Algfoor 2015]. Grid maps are a popular way of discretising a map into a search graph. In that process, a map is partitioned into atomic square cells, also known as tiles. Depending on the topology of the map, a tile is marked as either traversable or blocked. Traversable tiles are nodes in the search graph. Graph edges connect adjacent traversable tiles. A search algorithm is used to search for the path in the

search graph. Search algorithms can be categorised into uninformed search algorithms and informed search algorithms. Uninformed search algorithms use only the information provided in the problem definition, while informed search algorithms employ additional knowledge of the search space to find the path more efficiently. Generally, an informed search algorithm uses a heuristic function to impart the additional knowledge to the search algorithm. In the following section, we investigate a variant of pathfinding, known as multi-goal pathfinding, which is the main focus of this dissertation.

1.2 Multi-goal pathfinding

Classical pathfinding consists in finding a path between 2 points, a start and a destination. Multi-goal pathfinding, however, aims at finding a path between 2 points, the start and the destination, which also connects a set of points along the path. Those points are the locations at which a given set of goals can be satisfied. In the literature, there are 2 common variants of multi-goal pathfinding. In the first variant, given a single start and multiple destinations, multi-goal pathfinding is defined as a problem of searching for a path between the start and each destination resulting in multiple paths [Lim 2014]. The second variant of multi-goal pathfinding is treated as a Travelling Salesman Problem in which the aim is to find a path from a start to a number of goals before reaching a destination. The multi-goal pathfinding addressed in this dissertation bares more resemblance to the second variant than the first in the sense that it has a start and a destination, and thus the solution is a single path from the start to the destination.

The classical Travelling Salesman Problem can be defined as: *Given a set of cities and the cost of travel between each possible pair, the Travelling Salesman Problem is to find the best possible way of visiting all the cities and returning to the starting point that minimises the travel cost* [Matai 2010]. Essentially, the objective of this problem is to determine the order in which the salesman should travel a set of cities to minimise the cost of travel. The main distinction between our problem and the classical Travelling Salesman Problem is that, in our problem, a goal may be satisfied at multiple locations. Rather than determining the order of goals, we are interested in determining the locations in which each given goal can be satisfied and finding the optimal path from the start to the destination in which all the given goals can be satisfied in an order which we assume is given. The other difference is that the solution to the classical Travelling Salesman Problem eventually leads the salesman back to the start. In our problem, the solution connects a start to a destination, which may not necessarily be the same as the start.

In [Werner 2011], the author addresses a variant of Travelling Salesman Problem in which there is a partial order constraint on the goals. The Partially Ordered Travelling Salesman Problem, as called by the author, is defined as: *Assume we are given a weighted Graph G and a partially ordered subset $P \subseteq V(G)$ of the vertices of G. Defining a tour to be given by a total ordering of the vertices in P and the*

length of such a tour as the sum of the length of the shortest ways interconnecting the vertices of P in this respective order, find the shortest tour. This problem still consists in determining the order of the goals that minimises the cost, which is not the case in our problem. Furthermore, as in the classical Travelling Salesman Problem, this Partially Ordered Travelling Salesman Problem also aims at finding a path that returns to the starting point.

The multi-goal pathfinding addressed in this dissertation consists of 2 main problems: the pathfinding problem and the goal satisfaction problem. Basically, it can be considered as a pathfinding problem with the addition of goals to be satisfied in a given order. Moreover, solving multi-goal pathfinding in ubiquitous environments necessitates an approach that is capable of handling the constraints specific to such environments, namely dynamics, mobility, and openness. In consequence, an approach to solving multi-goal pathfinding in ubiquitous environments needs to be capable of handling these 3 constraints. More precisely, these constraints impose the following issues:

- The needs for up-to-date information: In ubiquitous environments, the states of cybernetic, physical, and/or social entities may evolve over time. Such dynamics renders it impossible to have complete knowledge of the environment prior to path computation, and thus requires up-to-date information regarding the current state of the environment during path computation and path execution.

- The needs to consider the latency of accessing resources: Relevant and up-to-date information is required to find an optimal path in the current state of the environment. Such information is obtained from various resources including the cybernetic, physical, and/or social entities located in the environment. The process of retrieving information from resources results in latency caused by resource accesses and data transfers.

The approaches to multi-goal pathfinding in the literature such as [Laporte 1992, Bektas 2006] focus on finding the optimal order of goals. They are, therefore, not compatible with our problem in which the order of goals is given and imposed. In the following sections, we present some of the most widely used search algorithms as they are fundamental to the more advanced algorithms as well as to the search algorithms designed to address the constraints such as dynamics and mobility.

1.3 Basic search algorithms

Search algorithms are an essential element in solving pathfinding problems. They can be categorised into 2 groups: uninformed or blind search algorithms and informed or heuristic search algorithms. Each group consists of a wide range of algorithms and variations. To measure the performance of a search algorithm, 4 criteria are used [Russell 2016]:

- Completeness: Is the algorithm guaranteed to find a solution if there is one?

- Optimality: Does the algorithm find the optimal solution?
- Time complexity: How long does it take to find a solution?
- Space complexity: How much memory is needed to perform the search?

In this section, we discuss the classical search algorithms of each category and assess them based on these 4 criteria.

1.3.1 Uninformed search algorithms

Uninformed search algorithms are the most fundamental search algorithms. Since no additional information about nodes in the search space is known, the algorithms are not able to determine whether a node is more promising than the others. The strategy employed in this type of algorithms consists in generating successors of a node and determining whether a node is a goal node. The key characteristic that makes an uninformed search algorithm different from the others of the same category is the order in which nodes are expanded [Russell 2016].

1.3.1.1 Breadth-first search

Breadth-first search starts the search process by expanding the root node. Then, it expands all the successors of the previously expanded node. Breadth-first search expands all the nodes at a given depth of the search tree before continuing to the next level. To achieve this, a simple use of FIFO queue is sufficient. New nodes, which are deeper, are placed at the back of the queue, while old nodes, which are shallower, are in the front of the queue. Thus, the shallower nodes get expanded first. Breadth-first search is complete if the search space is finite. It guarantees that the goal found is the shallowest goal node, which means that the goal is optimal if all the step costs are identical. However, space and time complexity of breadth-first search are exponential, making it unsuitable for solving complex or even normal-sized problems [Russell 2016].

1.3.1.2 Depth-first search

Depth-first search expands the deepest node in the search tree first. It maintains a LIFO queue so that the most recently generated node (i.e., the deepest node) is chosen for expansion. Depth-first search is complete if the search space is finite and a mechanism to avoid redundant paths and repeated nodes is used so that the algorithm will expand every node eventually. However, it is not optimal. Its time complexity is $O(b^m)$ where b is the branching factor (i.e., maximum number of successors of any node) of the search space and m is the maximum depth of the search tree, which can be much bigger than the depth of the shallowest solution or infinite. The advantage of depth-first search over breadth-first search is the space complexity. For a search space with branching factor b and maximum depth m, depth-first search needs $O(bm)$ [Russell 2016].

1.3.1.3 Uniform-cost search

Breadth-first search is optimal only if all the edge costs are equal because it expands the shallowest unexpanded node. Uniform-cost is optimal with any edge costs. In uniform-cost search, the order in which nodes are expanded is determined by the path cost from the starting node to the node being evaluated. The node with the lowest path cost is selected for expansion. To achieve this, uniform-cost search maintains a priority queue ordered by the path cost. When a node is expanded, the goal test is applied to determine whether the expanded node is the goal node. The pseudocode of uniform-cost search adapted from [Russell 2016, Chapter 3, Section 3.4, p. 84] is shown in Algorithm 1. Uniform-cost search guarantees completeness if the cost of every step exceeds some positive constant. It is also optimal provided that step costs are non-negative. The time and space complexity of uniform-cost search is $O(b^{1+\lfloor C^*/c \rfloor})$ where b is the branching factor, C^* the cost of the optimal path, and c the minimum cost of any step. Dijkstra's algorithm can be regarded as a variant of uniform-cost search. The difference is that Dijkstra's algorithm searches for the shortest path from the start node to every other node in a graph, whereas uniform-cost searches for the shortest path from the start node to only one node, the goal node.

1.3.2 Informed search algorithms

Contrary to uninformed search algorithms, informed search algorithms employ knowledge specific to the problem in order to find solutions more efficiently. In informed search algorithms, node expansion is based on an evaluation function, commonly denoted as f. The evaluation function determines the cost estimate of nodes. The node with the lowest f is expanded first. The choice of f distinguishes one informed search algorithm from another. In most cases, the evaluation function incorporates a heuristic function. The heuristic function estimates the cost of the cheapest path from a node to the goal node. The additional problem-specific knowledge is commonly used in heuristic functions [Russell 2016]. Many informed search algorithms have been developed such as greedy best-first search, recursive best-first search, and A* search. A* [Hart 1968] is probably the most widely used heuristic algorithm. Many variants of A* have been designed to address various aspects of the search such as improving search efficiency, reducing resource usage, and adapting to dynamic environments. Therefore, the rest of this section is dedicated to examining A* search and its variants.

A* search

A* is a best-first search algorithm that evaluates nodes by combining $g(n)$ value, the cost of the best known path from the start node s_0 to reach the node n, and $h(n)$ value, the estimated cost from the node n to the closest goal node. Therefore, the cost function of a node n is defined as $f(n) = g(n) + h(n)$. $f(n)$ estimates the cost of the cheapest path to the goal node through n. A* maintains an Open list and a Closed list. The Open list contains the nodes that have been generated, and

Algorithm 1 Uniform-cost search algorithm as described in [Russell 2016, Chapter 3, Section 3.4, p. 84]

```
1:  node ← initial node with path cost g = 0
2:  frontier ← a priority queue ordered by g , with node as the only element
3:  explored ← ∅ a list of explored nodes
4:  while true do
5:     if frontier is empty then
6:        return failure
7:     end if
8:     node ← pop(frontier)
9:     if node is goal node then
10:       return node
11:    end if
12:    add node to explored
13:    successors ← expand(n)
14:    for each successor succ in successors do
15:       if succ not in explored or frontier then
16:          frontier ← insert(succ)
17:       else if succ in frontier has a higher cost than the newly found succ then
18:          replace the succ in frontier with the newly found succ
19:       end if
20:    end for
21: end while
```

are waiting to be expanded. The Closed list is a set of expanded nodes. In each iteration, A* selects a node with the smallest path cost evaluated using f from the Open list to expand. A pseudocode of A* is illustrated in Algorithm 2.

A* is optimal if the heuristic function $h(n)$ is consistent [Russell 2016]. A heuristic function is consistent (or monotonic) if $h(n) \leq c(n, n') + h(n')$ for all nodes n and n' such that n' is a successor of n, $c(n, n')$ the cost from n to n', and $h(t) = 0$ for all goal nodes t. A* is also complete provided that there is a finite number of nodes with a cost less than or equal to the cost of the optimal path C^*. Another positive property of A* is that it is optimally efficient for any given heuristic. A* expands all the nodes with $f(n) < C^*$, and none with $f(n) > C^*$. No other optimal algorithm is guaranteed to expand fewer nodes than A* as they also have to expand all the nodes with $f(n) < C^*$ to avoid the risk of missing the optimal solution.

The main drawback of A* is its space complexity. A* stores all the generated nodes in memory. Therefore, it is not practical to use A* for large-scale problems. To overcome the space problem, some variants of A* have been developed such as iterative-deepening A* [Korf 1985] and simplified memory-bounded A* [Russell 1992]. Iterative-deepening A* is a variant of iterative-deepening depth-first search. The difference is that iterative-deepening depth-first search uses depth as the threshold for each iteration, while iterative-deepening A* uses the cost function $f(n) = g(n) + h(n)$. In each iteration, the algorithm performs a depth-first search and cuts off the branch with $f(n) > \mathit{threshold}$. At the start of the search, the threshold is cost of the initial node. For each iteration, the threshold is the minimum cost of all the costs exceeding the current threshold. Iterative-deepening A* combines the benefits of depth-first search and A* search. It uses less memory than A*, and focuses on exploring the most promising nodes, thanks to the cost function. Like A*, simplified memory-bounded A* expands the most promising nodes until the memory is full. In that case, it removes the worst node, which is the highest f-value node, from the search tree to be able to add a new node to the search tree. Then, it stores the value of the removed node to its parent. In this way, we know the quality (i.e., the cost) of the best path in the removed subtree.

1.3.3 Analysis of basic search algorithms

In this section, we have presented different search algorithms, and discussed them with regard to the 4 criteria, namely completeness, optimality, time complexity, and space complexity.

Uninformed search algorithms find a solution by blindly searching through the search space. Some uninformed search algorithms are complete and optimal. However, their time and memory requirements make them impractical for solving complex and large-scale problems. Informed search algorithms employ a heuristic function to search more efficiently than uninformed search algorithms. The most widely known informed search algorithm A* is optimal, complete, and efficiently optimal given certain conditions. However, its memory requirement renders it impractical for large problems.

Algorithm 2 Pseudocode of A* as described in [Fukunaga 2017]

```
1: Openlist ← s_0
2: while Openlist ≠ ∅ do
3:    Get and remove from Openlist a node n with the smallest f(n)
4:    Add n to Closedlist
5:    if n is a goal node then
6:       Return the path from s_0 to n as the solution
7:    end if
8:    for each successor n' of n do
9:       g_1 = g(n) + c(n, n')
10:      if n' ∈ Closedlist then
11:         if g_1 < g(n') then
12:            Remove n' from Closedlist and add it to Openlist
13:         else
14:            Continue
15:         end if
16:      else
17:         if n' ∉ Openlist then
18:            Add n' to Openlist
19:         else if g_1 ≥ g(n') then
20:            Continue
21:         end if
22:      end if
23:      Set g(n') = g_1
24:      Set f(n') = g(n') + h(n')
25:      Set parent(n') = n
26:   end for
27: end while
28: Return failure (no path exists)
```

These algorithms are serial. Each step is executed sequentially as the global state of the search is required. For example, in each iteration, A* selects a node with the lowest cost estimate f to expand. Serial search becomes impractical when the latency of computing edge costs is introduced as the algorithm is blocked while waiting for required data. Furthermore, with these basic search algorithms, when there are one or more edge cost changes during or after computation, the solution needs to be recomputed from scratch. Therefore, they are not applicable for solving pathfinding problems in ubiquitous environments, which are highly dynamic.

In the following sections, we review more advanced variants of informed search algorithms. First, we investigate the algorithms that aim at addressing the dynamics in the search space. Second, we look into the parallel and distributed algorithms for potential solutions to address the latency issue.

1.4 Search in dynamic environments

Ubiquitous environments are highly dynamic. The state of the environments may evolve over time. Searching in such dynamic settings needs to take into account the evolution of the environments during path computation and path execution to guarantee the validity of the solutions. Changes that may affect the solutions result from one of the following factors:

- mobility of the cybernetic, physical, and/or social entities located in the environment,
- changes of the state of the entities,
- and entries and exits of the entities.

Classical search algorithms are incapable of handling such dynamics. Every time changes occur, those algorithms need to recompute the solutions from scratch. This is impractical as the time between changes can be short, so the search may have to be restarted before it has finished and produced any usable solution. In the literature, there are incremental search algorithms that have been developed to handle the changes in dynamic environments. This type of algorithms corrects previous solutions based on updated information, and reuses data from previous searches to avoid recomputing from scratch. Another type of search algorithms that is of relevance to the dynamics is the anytime search algorithms. These algorithms trade optimality for speed, and can be useful in the context where computation time is critical. In this section, we discuss some of the most widely used algorithms in both categories with regard to the dynamics of the environment.

1.4.1 Incremental search

Search is often a repetitive process where one needs to solve a series of similar search tasks because the actual state is different from initially expected or the state evolves

over time [Koenig 2002b]. Therefore, the solution needs to be recomputed. The aim of incremental search techniques is to find solutions to a series of similar tasks faster than solving each search task independently from scratch. Such techniques are suitable for handling the dynamics and uncertainty of the environment.

D* algorithm

D* [Stentz 1994] is one of the first search algorithms that addresses pathfinding in unknown, partially known, and changing environments. It handles the dynamics in the sense that arc costs may change during the search process. It was named D* due to its resemblance to A*. Like A*, D* maintains an Open list which contains the states to be expanded. Each state is tagged with New, Open, or Closed. New states are the states that have never been on the Open list; Open states are the states currently on the Open list; Closed states are the states that are no longer on the Open list. Initially, all states are set to New. The goal state G is placed on the Open list because, in contrast to A*, D* searches backwards from the goal state. D* expands the nodes in Open list until the start state X is removed from the Open list (i.e., path found) or until there is no more state in the Open list (i.e., path not found).

For each state X, D* maintains an estimated cost from X to the goal state G determined by the cost function $h(G, X)$. For each state Y in the Open list, the key function $k(G, Y)$ is the minimum of $h(G, Y)$ since Y was placed on the Open list. When the path is being followed to reach the goal state, if a change in the arc cost is detected, the arc cost is updated and the affected states are placed on the Open list. States are classified either as a Raise state ($k(G, Y) < h(G, Y)$) or a Lower state ($k(G, Y) = h(G, Y)$). D* expands the states on the Open list to propagate the changes. Each time a state is removed from the Open list to expand, it passes the cost changes to its neighbours. The neighbours are in turn placed on the Open list to continue the process. D* is both complete and optimal. The drawback of D* is that it propagates cost changes through invalidated states without considering which expansions will benefit the moving subject at its current location.

Focussed D*

Focussed D* algorithm [Stentz 1995] computes an initial path from the goal state to the start state, and then modifies this path during path execution as arc costs change. It extends D* algorithm [Stentz 1994] by adding a heuristic focussing function, which makes it a complete generalisation of A* for dynamic environments.

D* algorithm maintains an estimated cost for each state X to the goal state G which is computed by a cost function $h(X)$. It propagates all cost changes without considering the current location of subject. All affected states become invalidated and are placed in the Open list to propagate the changes. Focussed D* introduces a focussing heuristic that takes into account the subject's location. Let the focussing heuristic $g(X, R)$ be the estimated cost of the path from the subject's current location to X. Focussed D* uses a function $f(X, R) = h(X) + g(X, R)$ to estimate path cost and to sort the Open list.

Focussed D* focuses the repairs to significantly reduce the total time required for the initial path computation and subsequent replanning operations. The focussing heuristic is used to focus the propagation of cost increases and cost reductions in the Open list. This enables the algorithm to focus on the direction of the moving subject and to reduce the total number of state expansions. The algorithm enables optimal path execution in the sense that an optimal path to the goal is followed at every state in the execution, assuming all known information at each step is accurate.

Lifelong planning A*

Lifelong planning A* (LPA*) [Koenig 2002b] is an incremental variant of A* algorithm. LPA* first searches the same way as A*, but when the arc costs change, the subsequent searches are much faster than using A* to recompute the path from scratch as LPA* reuses the parts of the previous search tree that are not affected by the changes. Like A*, for each state s, LPA* maintains a cost estimate from the start state $g(s)$. In addition, it also maintains another type of estimate from the start state $rhs(s)$, which is a one-step look ahead value based on $g(s)$, and thus makes it potentially more informed than $g(s)$. The two kinds of estimates always satisfy the following relationship:

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{start} \\ min_{s' \in Pred(s)}(g(s') + c(s', s)) & \text{otherwise} \end{cases}$$

A state s is locally consistent iff $g(s) = rhs(s)$. LPA* maintains a priority queue that always contains the locally inconsistent states, which are the states whose g value (i.e., cost estimate computed using the g function) needs to be updated to make them locally consistent. LPA* keeps expanding the states in the priority queue to update them until the goal state s_g is locally consistent and there are no states in the queue with a lower cost than that of s_g.

D*Lite

D*Lite [Koenig 2002a] addresses the problem of robot navigation in a unknown terrain. It is different from LPA* because D*Lite incrementally plans the shortest path from a robot's current position to the goal position as it moves rather than from a fixed starting point. D*Lite implements the same navigation strategy as Focussed D*, but it uses a different and simpler algorithm as it is based on LPA*. D*Lite starts by computing the shortest path from its current position to the goal. The robot follows this path until it reaches the goal. If some edge costs change during the path execution, D*Lite recomputes the shortest path from the current position to the goal. Based on LPA*, the algorithm repeatedly computes the shortest path as the edge costs change. However, LPA* searches from the start to the goal position. In D*Lite, the start position changes as the robot moves, so it adapts LPA* by reversing all edges and exchanging the start and the goal position. D*Lite is both complete and optimal. Compared to Focussed D*, D*Lite is simpler and more efficient.

1.4.2 Anytime search

The principle of anytime search algorithms is to provide an initial suboptimal solution very quickly, and then continuously improve the solution until the time runs out. This is particularly useful for solving problems in a complex environment where optimal search algorithms would require too much time to find a solution that they become impractical. A set of anytime search algorithms have been developed such as [Zilberstein 1995], [Dean 1988], [Zhou 2002], and [Likhachev 2004]. These algorithms start by computing an initial and potentially highly suboptimal solution, and then improve the solution as long as time permits. Most anytime algorithms that are based on A* gain speed-ups by inflating the heuristic values used by A*. However, they have no control over the suboptimality bound, while the initial solution is being improved.

Anytime Repairing A*

In [Likhachev 2004], the authors propose an anytime algorithm, entitled Anytime Repairing A* (ARA*), which provides suboptimality bounds for each successive search. ARA* exploits the fact that, in many domains, inflating the heuristic values used by A* tends to provide speed-ups at the cost of optimality. ARA* starts by performing an A* search with an initial inflation factor ε_0 to quickly produce an ε_0-bounded solution. Then, it executes a succession of A* searches, each with a decreasing inflation factor to improve the solution and reusing information from previous searches.

An expanded state of a particular search becomes inconsistent if the cost of one of its neighbouring states changes. The cost of a state may change when the inflation factor changes. In that case, the expanded state is added to INCONS list, which contains all inconsistent states. When the current search terminates, all the states in the INCONS list are inserted into a priority queue with a new inflation factor to be used in the next search. Considering only the inconsistent states for the previous search allows much of the information from the previous search to be reused. Therefore, minor computation is needed to find a new path when the inflation factor is reduced for each successive search. The limitation of ARA* is the fact that it is only applicable for static environments as it only takes into account the changes resulting from decreasing the value of the inflation factor. The algorithm does not consider any dynamic changes in the environments during path computation or path execution.

Anytime Dynamic A*

In [Likhachev 2005], the authors combine ARA* with D*Lite [Koenig 2002a] in order to solve search problems in dynamic environments and in an anytime manner. They propose an algorithm named Anytime Dynamic A* (AD*). Like ARA*, AD* performs a succession of searches with decreasing inflation factors to generate a series of solutions, each better than the previous one. When changes occur in the environment, states affected by the changes are inserted into Open queue. As in D*, the priority of the affected states is the minimum value of

their previous key value and their new key value. The key value of a state s is $key(s) = [min(g(s), rhs(s)) + h(s_{start}, s), min(g(s), rhs(s))]$ where $g(s)$ is the estimate of the cost from s to the goal, $rhs(s)$ is the one-step lookahead cost (as defined in LPA*), and $h(s_{start}, s)$ estimates the cost of the optimal path from s_{start} to s. The algorithm keeps expanding the states in the Open queue until the solution has reached a certain bound of suboptimality.

1.4.3 Analysis of search in dynamic environments

Searching in dynamic environments requires the algorithm to take into account changes during path computation and path execution as the subject is following the path. In this section, we have reviewed some of the most well-known and widely used search algorithms that were developed to handle the dynamics of the environments.

A number of algorithms, under the category of incremental search algorithms, have been proposed to cope with the dynamics and the uncertainty of the environments. These algorithms are efficient in correcting previous solutions based on updated information by using the efforts from previous searches. Furthermore, they can guarantee optimality of the solutions. The limitation of those algorithms with regard to our problem is that, given the latency required to determine the edge costs, the execution time would make them impractical as each step is performed sequentially and synchronously. In addition, the mechanism of repeatedly improving the solutions implies repeated accesses to resources, and thus more latency.

Anytime algorithms aim at reducing search time at the cost of optimality. There have been attempts to integrate incremental algorithms and anytime algorithms to address the time constraint as well as the dynamics of the environments, at the same time. The general drawback of this kind of algorithms is that the solution is suboptimal. As in the case of incremental algorithms, anytime algorithms would be rendered impractical when latency is introduced.

1.5 Parallel search algorithms

Most search algorithms were developed for single-agent problem solving. Therefore, they perform in a serial and sequential manner. Serial search algorithms are not compatible with our context where edge costs are computed by using information from resources because each access to a resource creates some latency. Our intuition is to investigate parallel and distributed search algorithms in which search processes can be executed in parallel, and thus may mitigate the impact of latency as the computation of different edge costs can be in parallel. In this section, we discuss some potential parallel and distributed search algorithms in the literature by keeping in mind the latency issue.

Generally, effective parallelisation of search algorithms offers two main benefits. First, it enables speedup on multi-core processors. Second, it offers an increased aggregate memory when running on a cluster. Best-first search algorithms maintain an Open list, which stores the set of states to be expanded. The authors

of [Kumar 1988] identify two broad approaches to parallelising best-first search algorithms based on how the maintenance of the Open list is parallelised, namely centralised parallel search and decentralised parallel search.

Centralised parallel search
In centralised parallel search, the Open list is shared among processors. Simple Parallel A* (SPA*) [IRANI 1986], Parallel K-Best First Search [Vidal 2010], and PA*SE [Phillips 2014] are instances of centralised parallel search. The advantage of such algorithms is that each processor expands one of the current best nodes from the shared Open list and generates its successors, thus computing edge costs in parallel. The drawback of the centralised parallel search is the synchronisation overhead which refers to the time wasted when some processors have to wait for the other processors to reach synchronisation points. For instance, concurrent accesses to the shared Open list becomes a bottleneck, even if lock-free data structures are used.

Decentralised parallel search
With the decentralised approach, each processor maintains its own Open list. Initially, a root processor generates some nodes and distributes them among the processors. Each processor, then, locally performs the search. This enables parallel node expansions, thus parallel edge cost computations, and avoids the concurrency overhead associated with the Open list. However, decentralised parallel search suffers from the search overhead as it expands more states than its serial counterpart. This is owing to the fact that the search space is distributed among processors. Hence, each processor has limited knowledge of the state of the overall search. Another drawback of this approach is the communication overhead that results from the exchanges of information among processors. Some examples of decentralised parallel search algorithms include HDA* [Kishimoto 2009, Kishimoto 2013] and PRA* [Evett 1995].

Distributed and parallel search
A parallel search algorithm that is also suitable for distributed search was proposed in [Nissim 2012]. The algorithm is a multi-agent version of A* that is available in 2 flavours: parallel and distributed. The parallel version of the algorithm, as in the case of other parallel search algorithms, exploits the parallel hardware to obtain speed-up. The distributed version of the algorithm was designed to be compatible with the search in a distributed setting by different agents possessing different search capabilities. The aim is to provide fast and distributed optimal search, while respecting agent privacy.

Discussion
Parallelisation is a promising way to address latency as it enables parallel computation of edge costs. However, one has to deal with search, synchronisation, and communication overheads to achieve good speed-ups in parallel search. Moreover, existing parallel search algorithms assume the information needed to compute the

edge costs is immediately available, so each processor executes a search algorithm in a serial and synchronous manner. With latency, the efficiency achieved by parallelisation could be reduced significantly.

1.6 Conclusion and discussion

In this chapter, we have presented three main categories of search algorithms, namely basic search algorithms, search algorithms for dynamic environments, and parallel search algorithms. Basic search algorithms cannot handle changes in the environments, so every time a change occurs, a new path needs to be recomputed from scratch. Furthermore, they execute in a synchronous and serial manner, which is impractical when dealing with latency. Incremental and anytime search algorithms address the dynamics of the environments, but cannot handle the latency as they are serial algorithms. Parallel search algorithms enable parallel computation of edge costs, but fail to take into account the changes in the search space.

It appears that individually, the existing algorithms are not sufficient to handle our problem where both dynamics and latency need to be handled. Attempts to combine different types of algorithms to address multiple constraints can be seen in Anytime Dynamic A* [Likhachev 2005]. Integrating the algorithms for dynamic environments with parallel algorithms is a promising way to address our problem, which we would like to explore.

Chapter 2

Application: Intelligent Traveller Information Systems

Contents

Pathfinding is a problem that is fundamental in numerous fields such as transportation, robotics, and video games. Similarly, multi-goal pathfinding can be seen in various contexts including, but not limited to, trip planning, robotics and logistics. The relation between multi-goal pathfinding and trip planning can be evidently seen. We can refer to our previous example about the traveller named Bob who wants to plan a trip in an airport, presented in the introduction, as an instance of addressing multi-goal pathfinding in the context of trip planning. In this dissertation, we aim at proposing a generic approach to solving multi-goal pathfinding that can function across different fields. Nevertheless, our field of focus is transportation and smart mobility, more specifically, solving multi-goal pathfinding in trip planning which is generally required in traveller information systems.

Efforts have been invested to apply information and communication technologies in smart mobility such as intelligent transport information systems. The aim of such systems is to support transportation of humans and goods in order to safely and efficiently use transportation means and infrastructures [ETSI 2011]. More specifically, one of the most essential elements of intelligent transport information systems is the traveller information systems. The attempt to design traveller information systems dates back to the late 1960s, during which the systems were employed to merely inform travellers about congestion via one-way communication means including Variable Message Signs and Highway Advisory Radio [Adler 1998]. Various research and development have been conducted to study and improve traveller information systems. They have evolved into a more mature version, currently referred to as advanced traveller information systems or even intelligent traveller information systems. They aim at providing all kinds of travellers with multimodal (e.g., cars, train, ferry, bus) trip planning information and assistances before and during trips, route guidance services and information (e.g., directions, travel time), and travel-related advices such as incident warnings and parking information [U.S. Department of Transportation 1998].

In contemporary society where travel plays an important role in our daily lives, intelligent traveller information systems can be very beneficial. In this respect, travelling with personal vehicle and/or public transportation can be complicated and stressful, especially to unfamiliar destinations. For instance, in the case of using public transportation, choosing an optimised itinerary requires sophisticated knowledge about public transportation networks [Chiu 2005]. Similarly, travelling on private vehicles necessitates familiarity with roadways and road conditions. Through intelligent traveller information systems, the provision of useful travel information and assistances to travellers could lead to diminishing stress related to trip planning and navigation, more efficient trip choices, reducing travel time [Adler 1998], and avoiding congestion as well as dangerous driving conditions [Kumar 2003].

Though numerous traveller information systems are in operation, very few attempt to support multiple travel modes on a global geographical coverage. Some traveller information systems are designed to support only particular modes of trans-

portation and geographical areas, while others may be restricted by technical barriers. In addition, only a small number of those systems take travellers' preferences into account. Hardly any, or possibly none, dynamically use different sources of information adapted to travellers' requests. Owing to the functional and technical limitations of the existing traveller information systems, travellers are often required to use multiple systems in combination and to seek for further information from other sources such as weather applications and websites for traffic information in order to acquire enough information for their trips.

In this information age, smart and connected buildings and infrastructures provide a significant amount of information that can be used to make more informed decisions in travelling. They have redefined the term point-to-point travel guidance, as such guidance should also assist travellers in indoor navigation of various facilities, such as train stations, airports, and shopping complexes just to name a few. Most of the research in the field of indoor navigation concentrates on indoor positioning, rather than pathfinding. In the past, this made sense as indoor pathfinding could be simply done by using static information such as a static map of the environments. With the emergence of ubiquitous environments, such approach to indoor pathfinding can no longer be efficient due to the dynamics of the environments. Constraints such as dynamics and openness need to be taken into account. Up-to-date information of the environment is required during path computation to provide solutions coherent with the actual state of the environment.

The purpose of this chapter is to investigate our field of application which is the intelligent traveller information systems. The motivation behind this investigation is to study the existing systems and to identify their benefits and limitations in trip planning, both indoor and outdoor. The rest of the chapter is organised as follows. First, we review the existing systems for outdoor trip planning. Second, we discuss various solutions used in existing traveller information systems to solve indoor trip planning. Third, we discuss the data management models employed in existing traveller information systems. Fourth, we discuss the existing solutions for modelling transport and traveller information to identify the extent to which they are able to model different aspects of ubiquitous environments.

2.1 Outdoor trip planning and navigation

Most of the research and development on traveller information systems focus on outdoor trip planning and navigation. Diverse research has attempted to identify the important functionalities that should be included in traveller information systems. According to [U.S. Department of Transportation 1998], the key functionalities that a traveller information system should provide consist of multi-modal trip planning, route guidance services, and advisory functions. A common way to classify traveller information systems is via the transport services for which the systems provide assistances. In this way, we can categorise them into route guidance systems and transit information systems.

Route guidance systems assist drivers in making travel decisions by providing them with travel recommendations and traffic information [Herbert 2008]. The assistance provided by such systems may include decision aids in pre-trip planning, which involve selecting route and departure time as well as making trip or no-trip decision, and en-route support for route adaptation [Khanjary 2012]. Examples of route guidance systems are Centrally Determined Route Guidance [Yamaguchi 1999], *PersianGulf* [Khanjary 2011], and *STRG* [Chen 1993]. The features provided by route guidance systems are limited to private vehicles.

Transit information systems, however, assist public transport passengers in their trips using public transport services such as buses, railways, subways, and ferries. Information provided by transit information systems varies significantly. Types of information offered by such systems may include static information (e.g., transit routes, service schedules, fares), itinerary planning, and real-time information such as delays or incidents [Peng 2000]. For example, *TCL*[1] and *Tisséo*[2] provide information about public transport services in Lyon and Toulouse, respectively. The drawback of transit information systems is that they target only public transport.

Up to the present, a significant number of traveller information systems have been developed. The assistance provided to travellers vary from one system to another. In this section, we present and compare different types of traveller information systems. To this end, we classify them based on geographical coverage they support as it allows us to highlight the benefits and limitations of each type of the systems. Based on geographical coverage, traveller information systems can be classified into operator-specific, region-specific, and independent systems.

2.1.1 Operator-specific traveller information systems

An operator-specific traveller information system is dedicated to a particular transport operator. Hence, the spatial coverage of such a system is limited to the areas for which their operator provides the actual transportation services. Many transport operators have developed their own systems to provide information about their transportation services. *RATP*[3], designed for the public transport operator in Paris, and *VINCI AUTOROUTES*[4] for a highway operator in France are examples of operator-specific traveller information systems. For this type of systems, the spatial coverage, the travel mode(s), and the supported transportation services are limited to those of their operators. Travellers may need to use multiple systems of different operators to get the necessary information and assistance for their trips.

2.1.2 Region-specific traveller information systems

A region-specific traveller information system is designed to provide assistance for a specific geographical space. Commonly, such a system is a system developed by the

[1] http://www.tcl.fr/
[2] http://www.tisseo.fr/
[3] http://www.ratp.fr/
[4] http://www.vinci-autoroutes.com/

authority of a region or an area to serve within that particular region or area. Information provided by region-specific traveller information systems are not restricted to a single transport operator. It could be a combination of transportation services from different operators serving the region. For example, Moovizy[5] is a traveller information system that provides travel-related information and trip planning feature for the city of Saint-Étienne. The assistance provided by Moovizy is not limited to the transport operator of Saint-Étienne, namely STAS[6], but also includes other transport operators that operate in the region of Rhône-Alpes from Saint-Étienne to Lyon such as TIL coaches and TCL network operating in Lyon. Compared to operator-specific traveller information systems, region-specific traveller information systems often support a larger geographical coverage, more travel modes, and wider choices of transportation services. Nonetheless, they remain limited in terms of geographical coverage.

2.1.3 Independent traveller information systems

We refer independent traveller information systems to the systems that are independent of any specific transport operator or geographical areas. *Google Transit*[7], *iTransports*[8], and Rome2rio[9] are examples of independent traveller information systems. Currently, *iTransports* covers only cities in France and certain areas in Europe. *Rome2rio* and *Google Transit* are among the very few that attempt to support a global geographical coverage. This kind of systems has no restrictions in terms of transport operators or geographical areas. Their limitations are often the result of having insufficient data to provide the assistance for some transport services or geographical areas. The absence of deliberate restriction signifies the possibilities for further extension and development of the system.

2.1.4 Discussion

Existing traveller information systems for outdoor trip planning provide various features, types of assistance, and information for assisting travellers plan their trip. However, the path planning feature available in those systems address mainly the classical pathfinding problems, and to the best of our knowledge, none addresses multi-goal pathfinding.

2.2 Indoor trip planning and navigation

Compared to outdoor trip planning, indoor trip planning and navigation have been much less studied. In the field of indoor mobility, significant efforts have been invested to address indoor positioning rather than planning and navigation. One of

[5] https://www.reseau-stas.fr/en/moovizy-mobile-app/9
[6] https://www.reseau-stas.fr/
[7] http://www.google.com/landing/transit/
[8] http://www.itransports.fr/
[9] https://www.rome2rio.com/

the main reasons is that indoor planning requires indoor positioning capability. Another plausible reason is that enclosed spaces used to be simple and of small size, and thus easy to navigate. These days, enclosed spaces have grown considerably in sizes, complexity, and numbers. Furthermore, the emergence of smart environments that are highly dynamic such as smart buildings, smart campuses and smart transit stations emphasises the needs of indoor planning systems. In this section, we investigate various existing indoor trip planning systems to determine the extent to which they can be used to address multi-goal pathfinding in ubiquitous environments.

2.2.1 Trip planning and navigation in transit complexes

In [Czogalla 2015a], [Czogalla 2015b], and [Czogalla 2016], the authors present an approach to address indoor navigation in public transport facilities such as train stations and airports. In their work, indoor planning and navigation involves 2 steps: building or facility modelling and route search. The model of a facility incorporates only the spatial information of the facility. A building is considered multi-level, i.e., having multiple floors. Each individual floor is modelled as a grid. The entire building is modelled as a graph where each platform represents a node, and a stair, an escalator, or an elevator connecting between the platforms is an edge. For pathfinding, they propose an algorithm that combines breadth-first search with A* algorithm to perform the search within the graph of building. In terms of positioning, they employ various techniques including Bluetooth and Wifi cell positioning and QR-Codes.

2.2.2 Trip planning and navigation in cyber-physical environments

The work presented in [Subakti 2016] addresses indoor guidance systems in cyber-physical environment such as smart campuses. The authors proposed a system called a marker-based cyber-physical interaction system which collects sensor data from the physical environment and links them to different sources of information to provide guidance information to travellers as well as to act on the environment dynamically as part of the guidance. The example scenario provided in the paper is to guide a student who has an appointment with a professor from the gate of the building until the professor's office. The system detects the position of the student and calls the elevator automatically for the student. Once the student has arrived on the right floor, the light above the professor's office is turned on to allow the student to find the office easily.

2.2.3 Multi-goal trip planning

Most of the existing indoor trip planning and navigation systems address the classical pathfinding problem. Hardly any or possibly none addresses multi-goal pathfiding. The work in [Werner 2011] aims at solving indoor trip planning that has goals to satisfy, in an airport. The problem addressed in that work is as follows: *"Assume we are given a weighted graph G and a partially ordered subset $P \subseteq V(G)$*

of the vertices of G. Defining a tour to be given by a total ordering of the vertices in P and the length of such a tour as the sum of the length of the shortest ways interconnecting the vertices of P in this respective order, find the shortest tour." In his approach, the author models the environment as a grid to generate the search graph of the environment. He proposed two algorithms based on exhaustive search and genetic search to solve or approximate the problem.

The problem addressed in that paper is a Partially Ordered Travelling Salesman Problem. Even though it imposes some constraints on the goals to satisfy, the problem is still to find the total order of the goals, which is not the case in our problem of multi-goal pathfinding where the order of goals to satisfy is given and imposed.

2.2.4 Discussion

In the literature, there is a body of work that addresses indoor trip planning. Some work addresses the problem in cyber-physical environments. However, they address mainly the problem of classical pathfinding, and not multi-goal pathfinding. In addition, the work in the context of cyber-physical systems focuses on exploiting the information coming from the environment and acting on the environment, but does not consider the dynamic changes of the environment, nor the latency that could result from accessing the information about the environment.

2.3 Data management models

The quality of services provided by traveller information systems relies heavily on the quality of data that the systems use. For example, to perform multi-modal trip planning, information related to the trip using private vehicles and public transportation information from different operators is needed to propose travel plans for travellers. During the trip, information that may influence the travel such as traffic, road condition, delay and accidents are essential for adapting the trip. Therefore, in this section, we discuss different models that existing traveller information systems use to acquire necessary data to support their services.

2.3.1 Using data from other systems

One of the common methods for collecting data is by using the data from other systems. This approach is commonly implemented in traveller information systems of the authority owing to it requiring authorised permission to access other government-possessed systems. As an example, the *Beijing traveller information systems* [Hu 2002] collect real-time traffic flow data from many systems such as urban traffic signal control system for data of traffic lights control intersections, travel time estimation system for travel time and speed information, traffic monitoring system for current traffic condition images, and parking guidance system for parking

space information. Other examples are the *traveller information system for Edinburgh* [Lovicsek 1998], *Oklahoma's ATIS* [Campbell 2011], *ATIS for the Bay area* [Asad J. Khattak 1994], and *ONLYMOOV'*[10] for Grand Lyon region in France.

2.3.2 Using pre-collected data

Another approach consists in pre-collecting data from different sources during the construction of the system. With this approach, traveller information systems are built upon data acquired during the development of the systems. For instance, in the case of *ATIS for Hyderabad city* [Kumar 2005], geographic data, information of one-way road segments, speed limits, road names, city bus routes, and time tables of inter city bus, train and air services were collected to build a database of the system [Kumar 2003]. *Rome2rio* constructs a large repository of transportation data collected from many sources including thousands of transportation operators and OpenStreetMap for driving and walking direction. This approach is suitable for systems that use static data.

2.3.3 Collaboration-based model

This model depends on the contribution of various data owners to obtain data. Such a model is notably adopted by *Google Transit*. Providing the standard data format entitled General Transit Feed Specification (GTFS), *Google Transit* requires transportation operators or data owners to publish their data respecting the standard and to provide Google the location of the published data so that it can periodically fetch the data. Considering the influence of Google, such approach appears feasible, but compared to other approaches, it requires and depends on much more efforts from data owners.

2.3.4 Active approach to data collection

The active approach to data acquisition has also been employed in various traveller information systems. In this approach, a set of data sources is actively and periodically accessed to collect the necessary data and to detect the updates. Examples of traveler information systems that employ this approach are Multi-modal Intelligent Route Advisory System (MIRAS) [Chiu 2005] and Intelligent Transportation Web Services (ITWS) [Wu 2003]. MIRAS uses WebScript Tool [Chiu 2001] to gather transportation data from web pages of transportation companies. Similarly, ITWS uses a crawler to collect real time traffic information from the TANFB[11] website.

2.3.5 Analysis of the approaches to acquiring data

Each of the models presented has its own benefits, and is more compatible with certain cases than with the others. The common shortcoming of the discussed

[10] http://www.onlymoov.com/
[11] Taiwan Area National Freeway Bureau (http://www.freeway.gov.tw/)

models is in the fact that they are based on a static set of sources. Hence, the data available in the systems for providing various features remain limited, and may not be able to provide optimised solutions for travellers, especially in a dynamic setting. Furthermore, human intervention is often required, at different extents depending on the approach, for adding and updating data and/or data sources.

2.4 Data models for transport and traveller information

Data is indispensable to traveller information systems as well as to any other transport-related systems. Therefore, various data models have been proposed to provide a standard for modelling transport and traveller information such as transportation network, traffic information, and schedules. In this section, we investigate the existing data models and identify the extent to which they can be used to model various aspects of travels, especially in ubiquitous environments.

2.4.1 Transmodel

Transmodel[12] provides a conceptual data model reference for public transport, taking into account both multi-operator and multimodal aspects. The rationale behind the development of this reference is to offer a solution to the interoperability of the applications, and thus enabling a convenient integration of applications developed by different suppliers into a single system. This can be done by standardising the data structures used by different applications allowing the implementation of integrated information systems [Tra 2012].

2.4.2 Datex II

Datex II[13], a European Technical Specification, aims at providing a standard way for modelling and exchanging traffic data. According to [Group 2012], the notion of publication is used to exchange the information such as situation publication, elaborated publication, measured data publication, and traffic view publication. Supported mechanisms for data exchange include publisher push on occurrence, publisher push periodic, and client pull. The data modelling approach is based on UML, and the current implementation uses XML schema. The UML DATEX II data model is converted into an XML schema by a developed conversion tool. The resulting XML schema is used for development of real data exchanges by software developers. The uses of the DATEX II can be seen in the National Traffic Control Centre in England, French Ministry of Transport, Swedish Road Administration, Spanish Ministry of Transport, and two traffic centres in Germany (Frankfurt and Koblenz).

[12] http://www.billettique.fr/spip.php?article310
[13] http://www.datex2.eu/

2.4.3 GTFS

Providing a common format for public transport schedules and related geographic information, GTFS allows public transport agencies to publish their transit data via GTFS feeds, which can be used interoperably by user applications. A GTFS feed is a set of text files, with extension ".txt", each of which represents a particular aspect of transit [GTF 2015].

2.4.4 IFOPT

Identification of Fixed Objects in Public Transport (IFOPT) [transport 2007], a standard built on Transmodel by extending the concepts of location, defines a model and identification principles for fixed objects related to public access to public transport such as stop points, stop areas, stations, connection links, and entrances. Based on the Transmodel, four related sub models are defined: Stop Place Model, Point of Interest Model, Gazetteer Topographical, and Administrative Model.

2.4.5 GDF

GDF (Geographic Data Files) [GDF 2015] is a standard that provides both a model and a file interchange format for road network data. The GDF conceptual data model consists of three entities: levels (level 0 - Geometry , level 1 - Routing, level 2 - Driver instruction/mapping), attributes (e.g., junctions, route elements), and relationships.

2.4.6 NeTEx

Network Timetable Exchange[14] (NeTEx) provides means to exchange between different computer systems the timetable-related data for public transport services which include network topologies, timetables, data to support real-time operations, and basic fare data [transport 2009]. NeTEx describes data as XML documents that can be easily exchanged by many communication protocols.

2.4.7 SIRI

Service Interface for Real Time Information[15] (SIRI) is used to exchange information between servers including the control centres of transport operators and information systems, which contains real-time public transport vehicle or journey time data.

2.4.8 Analysis of the data models

The discussed models aim at providing a uniform and interoperable way for modelling and exchanging a predefined set of types of transport-related data. However, they address mainly the aspects related to outdoor navigation with public transport.

[14] http://www.netex-cen.eu/
[15] http://www.user47094.vs.easily.co.uk/siri/

To navigate in ubiquitous environments where the state of the environments evolves over time and where data of different structures are acquired from various sources that are not completely predefined, these models remain limited. In addition, in multi-goal pathfinding, by introducing the notion of goals, data of various fields other than transport-related fields also need to be modelled in an integrated fashion with the transport data. The previously discussed models are not able to address this requirement.

2.5 Conclusion

In this chapter, we investigated the existing traveller information systems for both indoor and outdoor trip planning, the approaches they use to acquire and manage data, and the existing data models in the field. The existing systems provide solutions to the classical pathfinding, and none addresses multi-goal pathfinding. Furthermore, the existing systems are built upon a static set of data and/or data sources. This property of the system is not compatible with ubiquitous environments of which dynamics, mobility, and openness are inherent characteristics. In such environments, the ability to dynamically use the data sources adapted to the current state of the environment and to dynamically retrieve up-to-date information is necessary. Regarding the data models, many models have been proposed to capture different transport and travel-related aspects. However, those models are not extensible to incorporate other types of data which are related to goals when solving multi-goal pathfinding.

Part II

CONTRIBUTIONS

Chapter 3

Semantic Representation of the Environment

Contents

Common approaches to classical pathfinding generate a search graph as an abstraction of the environment by using spatial information of the environment. A search algorithm is then employed to search for the path in the search graph. However, such an abstraction is insufficient for solving multi-goal pathfinding, especially in the context of ubiquitous environments. First, spatial information alone is not enough. Knowledge about each location in the environment is necessary to determine at which location a goal can be satisfied. For example, information pertaining to each restaurant such as the availability or the quality of the restaurant based on the reviews by other travellers acquired from its website enables us to choose a restaurant that is best suited for the traveller. Second, in a ubiquitous environment where the state of the environment evolves over time and the cybernetic, physical, and/or social entities located in the environment are dynamic and mobile, it is impossible to generate a static representation of the environment that is accurate and complete. We need an abstraction that enables dynamic accesses to up-to-date information about the environment as well as the entities. For instance, to get a trolley to transport luggage, instead of suggesting travellers to go to a trolley area that is at the opposite direction of the gate, it is possible to locate an available trolley nearby that was left by another person, thanks to the data from connected trolleys.

In this chapter, we present our solution to the environment abstraction problem. We propose a knowledge model for describing a ubiquitous environment that integrates all the elements necessary for solving multi-goal pathfinding which include

the spatial dimension of the environment, the entities located in the environment, the relevant resources that can be accessed to retrieve useful information about the environment and the entities, and the relationships between the entities and goals that can be used to determine via which entity a goal can be satisfied. The rest of the chapter is organised as follows. First, we describe our conceptual knowledge model for abstracting a ubiquitous environment for multi-goal pathfinding. Second, we provide an ontology for formalising our knowledge model. Third, we present the Smart Airport Activity Ontology which is an instance of our model specifically customised for abstracting a smart airport for multi-goal pathfinding. Finally, we summarise our proposal and conclude the chapter.

3.1 Knowledge Model of the Environment

Knowledge about the environment is indispensable for solving multi-goal pathfinding problems. In a dynamic setting as a ubiquitous environment, various types of information is needed. In this section, we present our conceptual knowledge model designed for describing the aspects of ubiquitous environments that are pertinent for solving multi-goal pathfinding. It enables us to model the following elements: the *spatial topology* of the environment, the *organisational structure* of the environment, the *cybernetic, physical, and/or social entities* located in the environment, the notion of *goals*, and the *resources* providing information related to the environment.

Definition 4 ***Knowledge model of the environment***
Let G be a set of goals and Act a set of activities. Let further $K : Act \rightarrow \mathcal{P}(G)$ be a function that associates an activity $act \in Act$ to a set of goals $AG \subset \mathcal{P}$ where act can satisfy any goal $g \in AG$. Then, in our model, we abstract an environment E at a given time t as a tuple $E_t = (L_t, C_t, OS_t, CPSE_t, Sit_t, Act_t, R_t)$ where:

- *E_t is the state of the environment E at time t;*
- *L_t is a finite set of nodes representing the locations in E_t;*
- *$C_t \subseteq L_t \times L_t$ is a set of arcs representing the connections between locations;*
- *OS_t represents the organisational structure of E_t;*
- *$CPSE_t$ is a finite set of cybernetic, physical, and/or social entities located in E_t;*
- *Sit_t is a set of relations between $CPSE_t$ and L_t. An entity $cpse_t \in CPSE_t$ is situated in a location $l \in L_t$ at time t;*
- *$Act_t = (A_n)_{n \in CPSE_t}$ is a finite set of activities that can be carried out through $CPSE_t$. We say that an entity $cpse \in CPSE_t$ located in $l \in L_t$ satisfies a goal g if the activity $act \in Act_t$ required to satisfy the goal g can be carried out through $cpse$;*

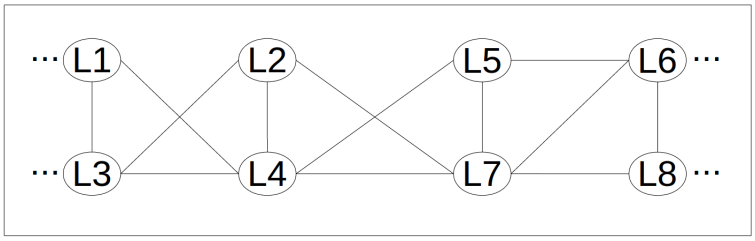

Figure 3.1: An example of a search graph

- $R_t = (R_n)_{n \in CPSE_t \cup C_t}$ *is a finite set of resources providing information about a cyber-physical-social entity or giving information on how to move between locations.*

Spatial topology

In this conceptual model, we abstract an environment in function of time as the state of the environment evolves over time. The spatial topology of an environment at time t is abstracted as a graph where nodes L_t represent the locations in the environment. Taking an airport as an example, a location can be a shop, a lounge, a gate, or a restroom. An arc $c \in C_t$ that connects a node $l \in L_t$ to another node $l' \in L_t$ is defined if, in the given environment, the location represented by l' is directly accessible from that by l. The arcs are dynamic due to the fact that the connections between nodes may evolve over time. At time t, it may be possible to go from l to l'; however, at time t', the path between l and l' may be blocked for some reason (e.g., accidents or congestion), so it is impossible to go to l' from l. Figure 3.1 demonstrates an example of a graph representing the spatial topology of an environment. L1 to L8 are nodes, and the arcs connect the nodes together. This graph is integral to solving multi-goal pathfinding. It serves as the search graph on which search algorithms are to be applied to find the path.

Organisational structure

An organisational structure of an environment is a way in which the locations in an environment are grouped together. The structure can be based on various criteria depending on the environment and the purpose for which the environment is modelled. For example, based on the types of locations, we can build an organisational structure by grouping together the locations of the same type. By using distance as the criterion, we can form groups of locations that are situated close to one another.

A common way to construct an organisational structure is by grouping locations in a hierarchy. An airport, for instance, may be organised as a hierarchy consisting of terminals and zones. In such a hierarchy, the locations of the airport are grouped into zones, and the zones are in turn grouped into terminals. Figure 3.2 illustrates an organisational structure that is based on the hierarchy of an airport. In this example, the hierarchy is comprised of Airport, Terminals, and Zones. Locations (L1, L2, ...) are grouped into Zones (Zone 1, Zone 2, ...). The Zones are grouped into

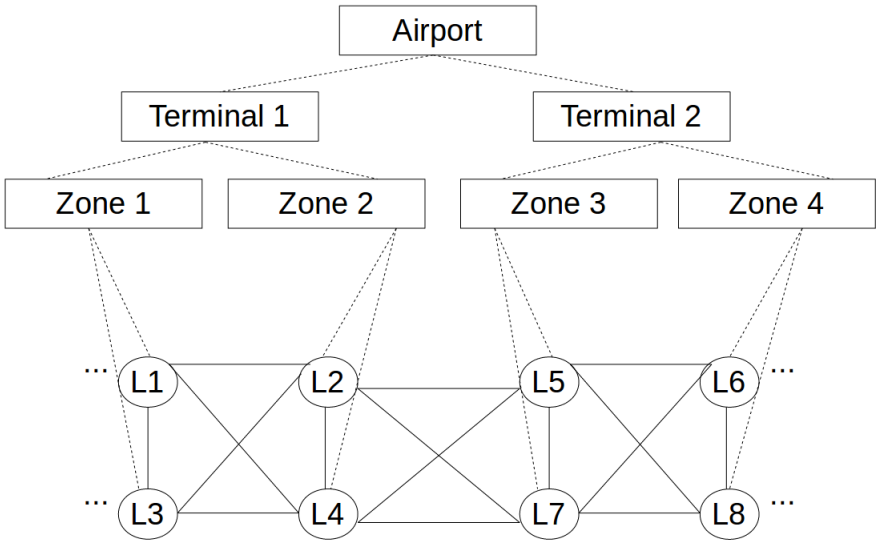

Figure 3.2: A hierarchy-based organisational structure of an airport

Terminals (Terminal 1 and Terminal 2). At the top of the hierarchy, the Terminals are grouped under the Airport. This hierarchy-based organisational structure can be viewed as a tree whose elements correspond to the hierarchy entities such as Terminals and Zones. The child relations indicate sub-hierarchy entities (e.g., Zones within a Terminal). The leafs of the tree are directly connected to the locations.

The search graph describes the connections amongst the locations of an environment, whereas the organisational structure describes *how* the locations are related to one another (e.g., located near one another, having the same type, or in the same area). The organisational structure is an essential component of our knowledge model as it allows us to impart the knowledge specific to the environment to our approach. We use such knowledge to organise the search process (as described in Chapter 4) and to focus the search process in the form of heuristics (as described in Section 5.5.2) in order to improve search efficiency.

Cybernetic, physical, and/or social entities

A ubiquitous environment may accommodate various types of entities including physical entities, cyber-physical entities, social entities, cyber-social entities, and cyber-physical-social entities. For example, an airport may contain different entities such as trolleys, ATM, help desks, restaurants, and waiting seats, just to name a few. It is noteworthy that some entities are mobile in the sense that their positions may change, and the state of the entities may also change over time. For instance, a trolley may be moved from one location to another by the traveller; an elevator may be available at time t, but at time t', it may become out of order. Therefore, in our knowledge model, we define the entities in function of time. The relation between the entities and the locations is that, at a given point in time, an entity is

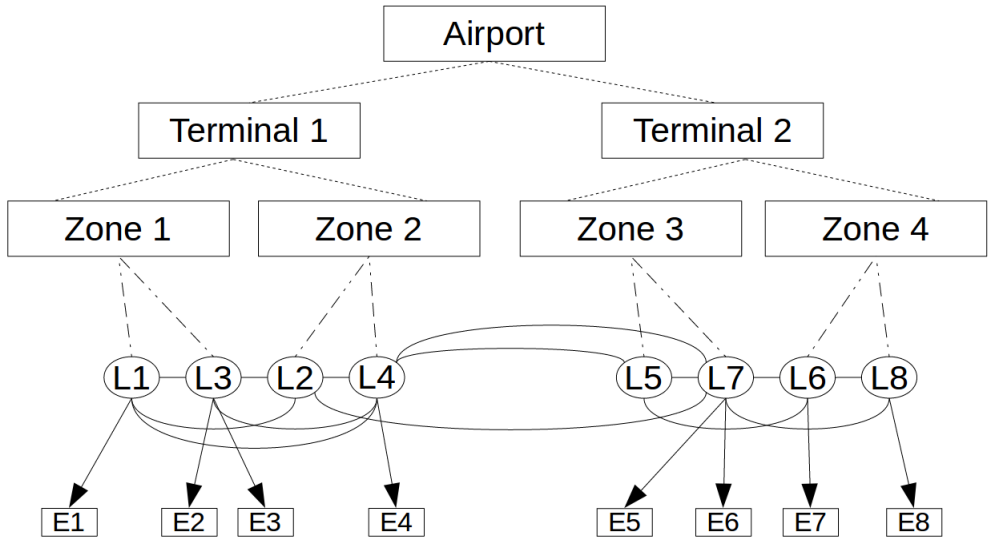

Figure 3.3: An example of the relations between locations and cyber, physical, and/or social entities

located at some location in the environment. Figure 3.3 depicts an example of the relations between the entities and the locations in an airport at a give time t. In this example, the entity E1 is located at the location L1, E2 and E3 at L3, E4 at L4, E5 and E6 at L7, E7 at L6, and E8 at L8.

These entities are included in our knowledge model because they are crucial for solving multi-goal pathfinding. Through them, travellers are able to satisfy their goals. A goal is satisfied by carrying out an activity. An activity could be carried out via a set of entities. As an example, suppose a traveller has a goal to achieve in an airport, which is to *have lunch*. Having lunch can be associated with an activity *Eat*. The entities such as restaurants and bakeries enable the activity *Eat* to be carried out, and thus are able to satisfy the goal *have lunch*. Furthermore, the relations between the entities, in this case restaurants and bakeries, and the locations in the environment allow us to locate those entities by searching through the search graph.

Moreover, to determine if an entity can satisfy a goal, we need to know what are the activities the entity supports. Therefore, the activities supported by the entities are incorporated in our model. In this way, we can exploit this knowledge to find the entities for satisfying the given goals during the problem-solving process. For example, for each goal to satisfy, we determine the activity that satisfies the goal via the function $K : Act \rightarrow \mathcal{P}(G)$ as defined in Definition 4. Then, based on the activity, we are able to find the suitable entities for satisfying the goal.

Resources

Ubiquitous environments are inherently dynamic. Providing a complete and accurate description of the environments is impossible. To adapt to the dynamics of the environments, we incorporate the notion of *resources* in our model. In this con-

text, a resource refers to a source of data from which relevant information can be retrieved. In this way, instead of including static information in the description of the environment, relevant resources can be accessed dynamically during the search to retrieve up-to-date information. The notion of resource in our knowledge model is conceptual and generic. In a concrete instantiation of the model, resources may be of various forms such as RESTful interfaces or APIs to different data sources.

We distinguish resources into: *connecting resources* and *entity resources*. Connecting resources provide information related to the path that connect two locations together (i.e., the arc between two nodes in the search graph). For instance, the path $c_{l-l'}$ between a location l and another location l' requires travellers to use an escalator and an elevator. Connecting resources for $c_{l-l'}$ can be an API to retrieve the information about the escalator and the elevator collected from the sensors installed on the escalator and in the elevator. Such information is necessary for evaluating the path, and thus deciding if the travellers should follow the path or choose other alternatives. Entity resources are the sources of information relevant to the entities. Through entity resources, information about the quality and the state of the entities can be acquired. Such information can be used to determine if an entity should be chosen for carrying out the activities to satisfy a given goal.

As an example, suppose a traveller wants to satisfy his goal which is to have lunch in an airport. Entity resources of different restaurants in the airport are accessed to retrieve information about the restaurants such as their availability and the ratings. The acquired information can be used to choose the restaurant that suits the condition of the traveller the most. Figure 3.4 shows an example of how resources are incorporated into our model. The entity resource ER1 is associated to the entity E1 as it provides information about E1. The same goes for ER2 with E2, ER3 with E3, and other entity resources with other entities. The connecting resource CR1 is assigned to the path between the location L1 and L4, which means that CR1 can be accessed to retrieve the information related to the path. This is also the case for CR2 with L1 and L2, CR3 with L3 and L4, and the other connecting resources with the other paths.

Modularity of the knowledge model

A complete example of an abstracted environment is shown in Figure 3.5. It comprises the following elements:

- nodes representing the locations in the environment such as L1, L2, L3, L4, and L5;
- arcs representing the connections between locations such as the arc between L1 and L2 (L1-L2), L3 and L4 (L3-L4), and L4 and L7 (L4-L7);
- the organisational structure of the environment in this example is a hierarchy that is comprised of different organisational entities such as Airport, Terminal 1, Terminal 2, and Zone 1;

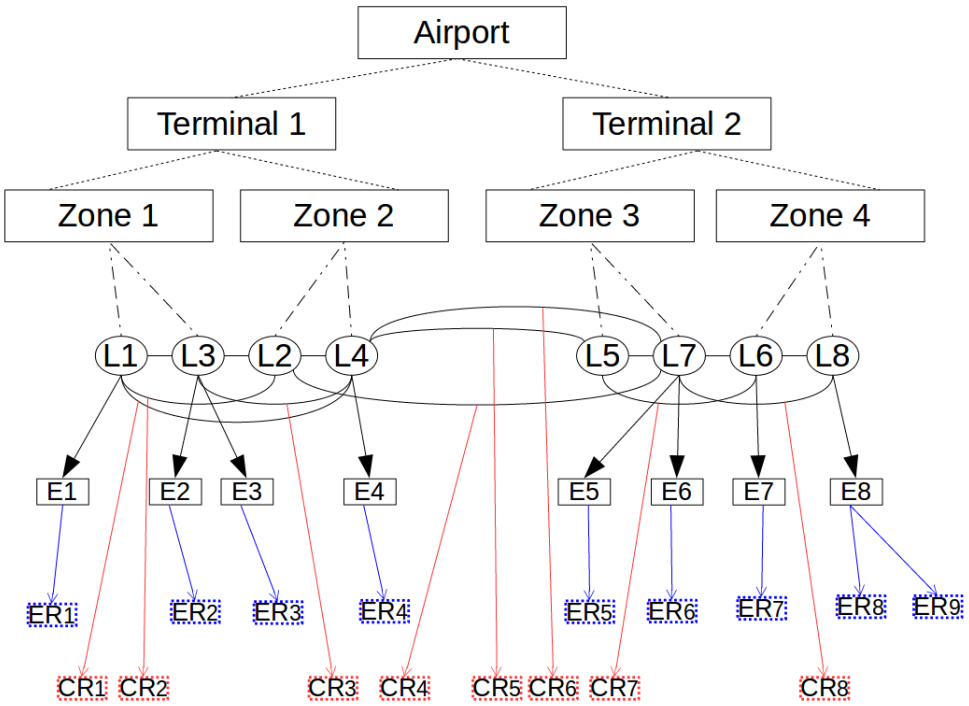

Figure 3.4: Incorporating resources in the model

- cybernetic, physical, and/or social entities located in the environment such as E1, E2, E6, and E8. Activities can be carried out via cybernetic, physical, and/or social entity which are provided in the description of each entity;
- resources providing information about cyber-physical-social entities such as ER1, ER2, ER5, and ER8;
- resources giving information on how to move between locations such as CR1 for L1-L4 and CR7 for L5-L6.

Figure 3.5 demonstrates the composition of different layers of the model. The top layer is the search graph which is the spatial representation of the environment. The bottom layer represents the resource space which contains the resources relevant to the environment. Our knowledge model, as shown in the middle layer, connects the top and the bottom layers together, and integrates them with other components that are necessary for solving multi-goal pathfinding.

The conceptual model enables environments to be modelled in a modular manner, which allows different subproblems to be solved independently when necessary. Pathfinding problem is concerned with the search graph in the top layer, while the problem of determining the entities to satisfy goals is concerned with the resource space in the bottom layer. Each problem can be addressed independently. Depending on the available information and the structure of the given environment, various search algorithms can be employed to solve pathfinding. Different mechanisms can

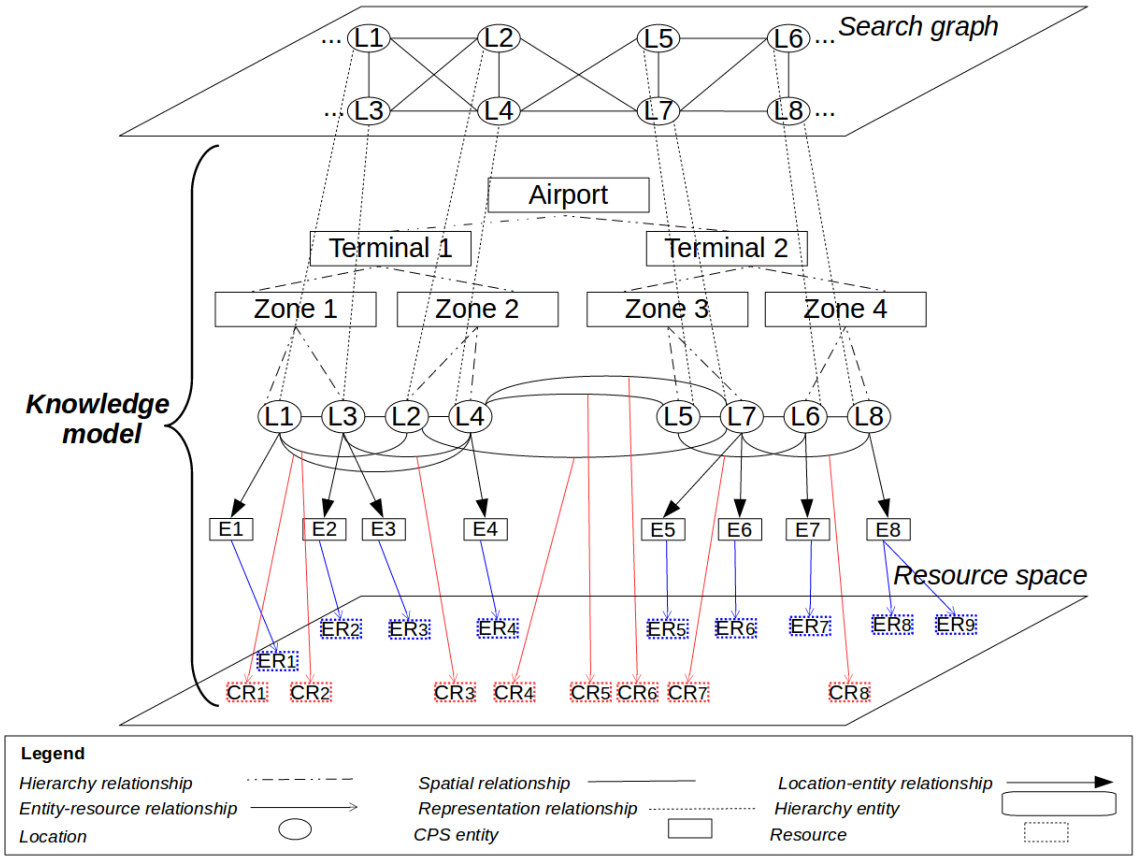

Figure 3.5: An example of the conceptual model of an environment

be used to access resources to retrieve the necessary information to a solve goal satisfaction problem. The elements and their relationships modelled in the middle layer serve as the necessary knowledge that is used to solve both pathfinding and goal satisfaction problems in an integrated and more efficient manner. For example, an uninformed search algorithm such as uniform-cost search can be used when only the information of the search space is available. An informed search algorithm such as A* search can be employed when additional information of the environment is available to be used in the heuristic function of the algorithm to improve the search process.

To formalise our knowledge model, we propose an ontology for describing the components of the knowledge model and their relationships, which is presented in the following section.

3.2 Ubiquitous Environment Abstraction Ontology

The knowledge model presented in the previous section is a conceptual model. To provide a concrete implementation of the model, we employ semantic web principles and technologies. More precisely, we propose an ontology entitled Ubiquitous Environment Abstraction Ontology, abbreviated as `ueao`, to capture the knowledge incorporated in the model. To create this ontology, we use the Web Ontology Language [Hitzler 2012] (OWL). The ontology is employed to provide a description of a ubiquitous environment in Resource Description Framework [Cyganiak 2014] (RDF). Figure 3.6[1] illustrates the classes and properties defined in the ontology.

3.2.1 Classes and properties

This ontology consists of 11 classes and 14 object properties that capture the aspects of ubiquitous environments as defined in our knowledge model (Definition 4).

Classes

The classes defined in the `ueao` ontology are as follows:

- Class `ueao:Location` represents the locations in an environment (defined as L_t in Definition 4);
- Class `ueao:Connection` describes the paths (i.e., the arcs in a search graph) between locations in an environment at a given point in time (defined as C_t in Definition 4);
- Class `ueao:OrganisationalEntity` is defined as the class of the organisational entities that belong to the organisational structure of an environment (defined as OS_t in Definition 4);
- Class `ueao:ConnectingPoint` represents the points where the organisational entities are connected to one another;

[1] The arrow indicates the direction of the property.

- Class `ueao:CPSEntity` is the class of cybernetic, physical, and/or social entities located in an environment at a given point in time (defined as $CPSE_t$ in Definition 4);
- Class `ueao:PhysicalEntity` is a subclass of `ueao:CPSEntity`. It represents physical entities which are the entities that occupy a physical space;
- Class `ueao:CyberPhysicalEntity` is a subclass of `ueao:CPSEntity` and defined as a class of physical entities that are also equipped with cybernetic abilities;
- Class `ueao:SocialEntity` is the subclass of `ueao:CPSEntity` that possesses social capabilities – the ability to interact with other entities;
- Class `ueao:CyberPhysicalSocialEntity`, a subclass of `ueao:CPSEntity`, represents physical entities that have both cybernetic and social abilities;
- Class `ueao:PotentialActivity` represents the activities that can be carried out in an environment at a given point in time (defined as Act_t in Definition 4);
- Class `ueao:Resource` represents the resources that can be accessed to retrieve information about cybernetic, physical, and/or social entities in an environment or about the paths between locations in the environment (defined as R_t in Definition 4).

The concept of *Activity* is widely used in various domains and fields of research. Existing activity ontologies such as [Catarci 2006] and [Fox 1993] model activities for their respective domains, while [Abdalla 2014] attempts to provide a generic ontology that captures the common core of activities. The concept *PotentialActivity* defined in our ontology does not represent the actual activities as addressed in [Abdalla 2014], [Catarci 2006] or [Fox 1993]. Potential activities stress on the *ability* to be performed, independent of whether they are actually carried out. For example, an instance of `ueao:PotentialActivity` is *eating a pizza*, while the actual activity would be *eating a pizza in a restaurant X at time Y*. The relationship between an actual activity *act* and its potential activity act' can be viewed as *act realises* act' at a given point in time.

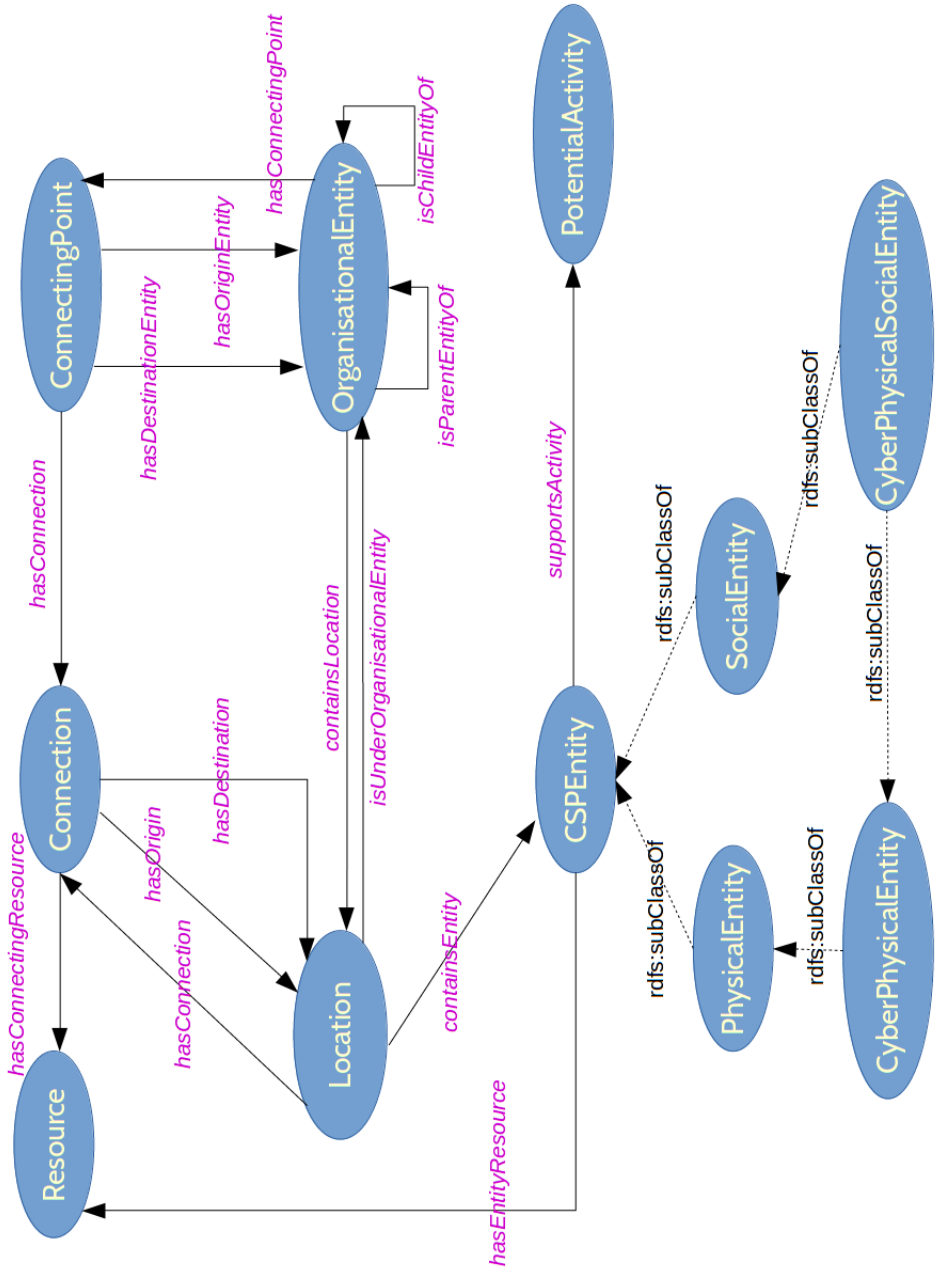

Figure 3.6: Ubiquitous Environment Abstraction Ontology

Properties
In the `ueao` ontology, we define the following properties:

- Property `ueao:hasOrigin` describes the relationship between a connection and a location. A connection represents an arc from a location (the origin location) to another location (the destination location). This property relates a connection to a location that is the origin location of the connection;

- Property `ueao:hasDestination` relates a connection to a location that is the destination location of the connection;

- Property `ueao:hasConnectingResource` relates a connection to a resource that provides information about the path between the origin location and the destination location of the connection;

- Property `ueao:containsLocation`: An organisational entity may contain a set of locations. The property `ueao:containsLocation` relates an organisational entity to a location that it contains;

- Property `ueao:isUnderOrganisationalEntity` relates a location to the organisational entity that contains the location;

- Property `ueao:isParentEntityOf`: An organisational entity may contain a set of other organisational entities (the child organisational entities). We use the property `ueao:isParentEntityOf` to relate an organisational entity to a child organisational entity;

- Property `ueao:isChildEntityOf` relates an organisational entity to its parent organisational entity;

- Property `ueao:hasConnectingPoint`: A connecting point (an instance of the class `ueao:ConnectingPoint`) represents a point where 2 organisational entities are connected. It is a unidirectional connection from an organisational entity (the origin entity) to another organisational entity (the destination entity). The property `ueao:hasConnectingPoint` relates an origin organisational entity to a connecting point;

- Property `ueao:hasOriginEntity` describes the relationship between a connecting point to an organisational entity. It relates a connecting point to an organisational entity that is the origin entity of the connecting point;

- Property `ueao:hasDestinationEntity` relates a connecting point to an organisational entity that is the destination entity of the connecting point;

- Property `ueao:hasConnection` relates a location or a connecting point to a connection;

  - If a location $l' \in L_t$ is accessible from another location $l \in L_t$ at time t, we say that there is a *connection* from l to l'. To capture this type of relationship, we use the property `ueao:hasConnection` to relate the location l to the connection from l to l' where l is the origin location and l' is the destination location of the connection;
  - The organisational entities of an organisational structure are connected though connecting points. In each connecting point, the origin entity *oe* is connected to a destination entity *de* via a connection from a location l to another location l' where l is contained in *oe* and l' in *de*. In other words, l is the exit point of *oe*, and it is connected to l' which is the entry point of *de*. We use the property `ueao:hasConnection` to relate a connecting point to a connection.

- Property `ueao:containsEntity`: relates a location to a cybernetic, physical, and/or social entity that it contains;
- Property `ueao:supportsActivity`: relates a cybernetic, physical, and/or social entity to a potential activity that it supports;
- Property `ueao:hasEntityResource`: relates a cybernetic, physical, and/or social entity to a resource that provides information about the entity.

3.2.2 Usage of the ontology

The classes and properties of this ontology are used to describe the necessary aspects for addressing multi-goal pathfinding, namely *spatial topology*, *organisational structure*, *cybernetic, physical, and/or social entities*, *goals*, and *resources*. In the rest of this section, the examples provided for each of the aspects are based on Figure 3.5. The prefixes used in the examples are:

- `@base <http://www.example.org/environment/description/>` which is the base URI for the description of the environment;
- `@prefix ueao: <http://www.semanticweb.org/ontologies/2018/1/ueao#>` which is the URI of the `ueao` ontology.

Spatial topology

To describe the spatial topology of a ubiquitous environment, we use the class `ueao:Location` to describe the locations in the environment, and the class `ueao:Connection` to describe the connections between the locations.

For example, using the ontology, the connection from location L3 to L2 can described in RDF as shown in the following description. In this description, the connection from L3 to L2 is `<Connection-L3-L2>`. The property `ueao:hasOrigin` is used to specify its origin location which is `<L3>`, and `ueao:hasDestination` to specify its destination location `<L2>`.

```
<Connection-L3-L2> a ueao:Connection ;
                   ueao:hasOrigin <L3> ;
                   ueao:hasDestination <L2> .
```

Below is an excerpt of the description of location L2. From location L2 (`<L2>`), it is possible to access to L1, L3, L4, and L7. This knowledge is described using the property `ueao:hasConnection`, as in the example.

```
<L2> a ueao:Location ;
    ueao:hasConnection <Connection-L2-L1> ;
    ueao:hasConnection <Connection-L2-L3> ;
    ueao:hasConnection <Connection-L2-L4> ;
    ueao:hasConnection <Connection-L2-L7> .
```

Below is an excerpt of the description of location L3. Location L3 (`<L3>`) is connected to L2, L1, and L4.

```
<L3> a ueao:Location ;
    ueao:hasConnection <Connection-L3-L2> ;
    ueao:hasConnection <Connection-L3-L1> ;
    ueao:hasConnection <Connection-L3-L4> .
```

Organisational structure

An organisational structure of an environment is composed of a set of organisational entities. We use the class `ueao:OrganisationalEntity` to represent the organisational entities and `ueao:ConnectingPoint` to describe how the entities are connected to one another. The organisational structure of the example in Figure 3.5 is a hierarchy that has Airport as the root. Airport has 2 direct child organisational entities, namely Terminal 1 and Terminal 2, which in turn also have child entities (the Zones).

In the excerpt below, we state that Terminal 1 (`<Terminal1>`) is an organisational entity by describing it as an instance of class `ueao:OrganisationalEntity`. The property `ueao:isChildEntityOf` and `ueao:isParentEntityOf` are used to describe the parent entity (Airport) and the child entities (Zone 1 and Zone 2) of Terminal 1, respectively. Terminal 1 is connected to Terminal 2 via a connecting point (`<ConnectingPoint-Terminal1-Terminal2>`) that is the connection from location L4 to L5.

```
<Terminal1>
   a ueao:OrganisationalEntity ;
   ueao:isChildEntityOf <Airport> ;
   ueao:isParentEntityOf <Zone1>, <Zone2> ;
   ueao:hasConnectingPoint <ConnectingPoint-Terminal1-Terminal2>.

<ConnectingPoint-Terminal1-Terminal2>
   a ueao:ConnectingPoint ;
   ueao:hasOriginEntity <Terminal1> ;
   ueao:hasDestinationEntity <Terminal2> ;
   ueao:hasConnection <Connection-L4-L5> .
```

Zone 1 is at the lowest level of the hierarchy, and thus has no child entity. However, it directly contains locations (L1 and L3), which is described using the property `ueao:containsLocation`.

```
<Zone1>
   a ueao:OrganisationalEntity ;
   ueao:isChildEntityOf <Terminal1> ;
   ueao:containsLocation <L1>, <L3> ;
   ueao:hasConnectingPoint <ConnectingPoint-Zone1-Zone2> .

<ConnectingPoint-Zone1-Zone2>
   a ueao:ConnectingPoint ;
   ueao:hasOriginEntity <Zone1> ;
   ueao:hasDestinationEntity <Zone2> ;
   ueao:hasConnection <Connection-L3-L2> .
```

Cybernetic, physical, and/or social entities

Cybernetic, physical, and/or social entities are represented by the class `ueao:CPSEntity`. At a given point in time, each entity is located at a location in the environment. For example, a lounge in an airport contains some trolley, ATMs, and toilets. To capture this knowledge, we use the property `ueao:containsEntity` to relate a location to an entity.

In the following description, we describe that the entity E1 (`<CPSEntity1>`) is located at location L1 (`<L1>`) and E2 (`<CPSEntity2>`) and E3 (`<CPSEntity3>`) at location L3 (`<L3>`), as in Figure 3.5.

```
<CPSEntity1> a ueao:CPSEntity .
<CPSEntity2> a ueao:CPSEntity .
<CPSEntity3> a ueao:CPSEntity .

<L1> ueao:containsEntity <CPSEntity1> .
<L3> ueao:containsEntity <CPSEntity2>, <CPSEntity3> .
```

Goals

A goal is satisfied by carrying out a set of activities. An activity is carried out through the use of a cybernetic, physical, and/or social entity. Therefore, to determine which entities allow an activity to be carried out, we describe the relationship between the entities and the activities. We use the class `ueao:PotentialActivity` to describe the activities and the property `ueao:supportsActivity` to describe the relationship.

Suppose that, in Figure 3.5, the entity E1 is a restaurant where one can eat, E2 a help desk where one can inquire for information, and E3 an ATM. We can describe the activities supported by each entity as follows:

```
<EatVegetarian> a ueao:PotentialActivity .
<SeekInformation> a ueao:PotentialActivity .
<WithdrawMoney> a ueao:PotentialActivity .

<Veggie-restaurant> ueao:supportsActivity <EatVegetarian> .
<Help-desk> ueao:supportsActivity <SeekInformation> .
<ATM> ueao:supportsActivity <WithdrawMoney> .
```

In `ueao` ontology, we describe the generic relationships between cybernetic, physical, and/or social entities and potential activities. The entities and activities vary from one environment to another. It is possible to extend the ontology to capture the activities specific to the environment.

Resources

To incorporate resources into the description of an environment, we use the class `ueao:Resource`. We distinguish 2 types of resources: entity resources and connecting resources. An entity resource provides information about a cybernetic, physical, and/or social entity. For example, a resource relevant to a restaurant can be the restaurant's website. A resource of an elevator can be an API to retrieve data from the sensors installed in the elevator. We use the property `ueao:hasEntityResource` to express the relationship between an entity and a resource as shown in the excerpt of the description below:

```
<EntityResource1> a ueao:Resource .
<EntityResource2> a ueao:Resource .
<EntityResource3> a ueao:Resource .

<CPSEntity1> ueao:hasEntityResource <EntityResource1> .
<CPSEntity2> ueao:hasEntityResource <EntityResource2> .
<CPSEntity3> ueao:hasEntityResource <EntityResource3> .
```

A connecting resource provides information related to the path between 2 locations. Such a path is represented by the class `ueao:connection` in our ontology. To attribute a connecting resource to a connection, we use the property `ueao:hasConnectingResource` as shown in the description below:

```
<ConnectingResource1> a ueao:Resource .
<ConnectingResource2> a ueao:Resource .
<ConnectingResource3> a ueao:Resource .

<Connection-L1-L4>
             ueao:hasConnectingResource <ConnectingResource1> .
<Connection-L1-L2>
             ueao:hasConnectingResource <ConnectingResource2> .
<Connection-L3-L4>
             ueao:hasConnectingResource <ConnectingResource3> .
```

3.2.3 Discussion

The first motivation behind using RDF, ontology, and OWL is the use of HTTP URIs to identify resources[2]. An HTTP URI can be dereferenced to access the representation of the resource identified by the URI. Furthermore, it also enables us to store an environment description in a distributed manner as each component of the environment (e.g., locations, organisational entities) is addressed by a URI, and thus can be looked up using its URI. Needless to say, managing the description in a centralised fashion is also possible.

The second motivation is the ability to integrate various kinds of information in an environment description. The schemaless property of RDF makes it possible to combine data of arbitrary kinds. In this way, different ontologies can be used to provide additional information about the environment. An environment may accommodate different types of cybernetic, physical, and/or social entities. Depending on the types of the entities, specific vocabularies can be used to provide more information about the entities. In real use-cases, this allows people that are in charge of each entity (e.g., the owner of a restaurant in an airport) in an environment to provide a description for their entity. They can incorporate additional knowledge specific to their entities. Such knowledge allows us to associate entities to more specific activities. For example, an owner of a restaurant may describe their restaurant, in addition to supporting eating, as a place that serves vegetarian food. Following this, it would be possible to address more specific goals such as eating vegetarian food.

The third motivation is owing to the fact that using ontologies in OWL enables reasoning. The ability to reason upon the knowledge of the environment is essential. In practice, we can expect the cases where the description of the environment is partial or incomplete. In such cases, reasoning upon the existing knowledge of the environment may discover additional knowledge to complete or improve the description.

To demonstrate a specialisation of the `ueao` ontology for a specific environment, in the following section, we present its adaptation to model a smart airport.

[2]In an RDF context, a resource can be anything an RDF graph describes. It does not necessarily refer to the resources defined in our ontology.

3.3 Ontology for smart airports

Ubiquitous Environment Abstraction Ontology `ueao` provides generic classes and properties for describing ubiquitous environments. Each environment may contain different cybernetic, physical, and/or social entities, and thus supports different types of activities. To adapt to each environment, `ueao` ontology can be extended to capture the knowledge specific to the environment. In this section, we present an extension of `ueao` ontology for a smart airport, entitled Smart Airport Activity ontology and abbreviated to `saa`.

3.3.1 Smart Airport Activity ontology

The aim of designing `saa` ontology is to extend `ueao` ontology to incorporate the types of cybernetic, physical, and/or social entities that are commonly present in a smart airport and the common activities supported by a smart airport. To determine the common activities that are carried out in an airport, we use the analysis of airport travellers' activities presented in [Liu 2014]. This analysis provides a classification of the common activities that travellers do in an airport. Based on this classification, we determine the cybernetic, physical, and/or social entities that support each activity.

In `saa` ontology, we add other classes of entities that were identified based on the classification of the common activities such as `saa:InteractiveHologramGuide`, `saa:HelpDeskGuide`, `saa:Restaurant`, and `saa:Check-inKiosk`. Figure 3.7 illustrates an excerpt of the ontology. Since we have the knowledge of which types of activities are supported by which types of entities, we can make explicit relations between the classes of entities, which are the subclasses of subclasses of `ueao:CPSEntity`, and the classes of activities, which are the subclasses of subclasses of `ueao:PotentialActivity`. These relations are captured by using existential restrictions on the property `ueao:supportsActivity`. For example, to state that instances of the class `saa:HelpDeskGuide` are those that support instances of `saa:SeekInformation`, we make use of the following axiom:

$$HelpDeskGuide \equiv \exists supportsActivity.SeekInformation$$

These explicit relations facilitate the process of identifying the activities supported by a given entity, which is crucial for solving the goal satisfaction problem of multi-goal pathfinding. For instance, given the following description of a kiosk, we are able to know that check-in can be carried out at the kiosk because the kiosk is an instance of class `saa:Check-inKiosk` which supports the activities of type `saa:Check-in`.

```
<Kiosk-ID-252> a saa:Check-inKiosk .
```

It is essential to note that the classes representing cyber-physical-social entities and activities in this ontology are by no means exhaustive nor compatible with all kinds of airports. It was, however, defined in a manner such that it can be reused and extended to capture the specifics of a particular airport as well as the desired level of granularity of potential activities.

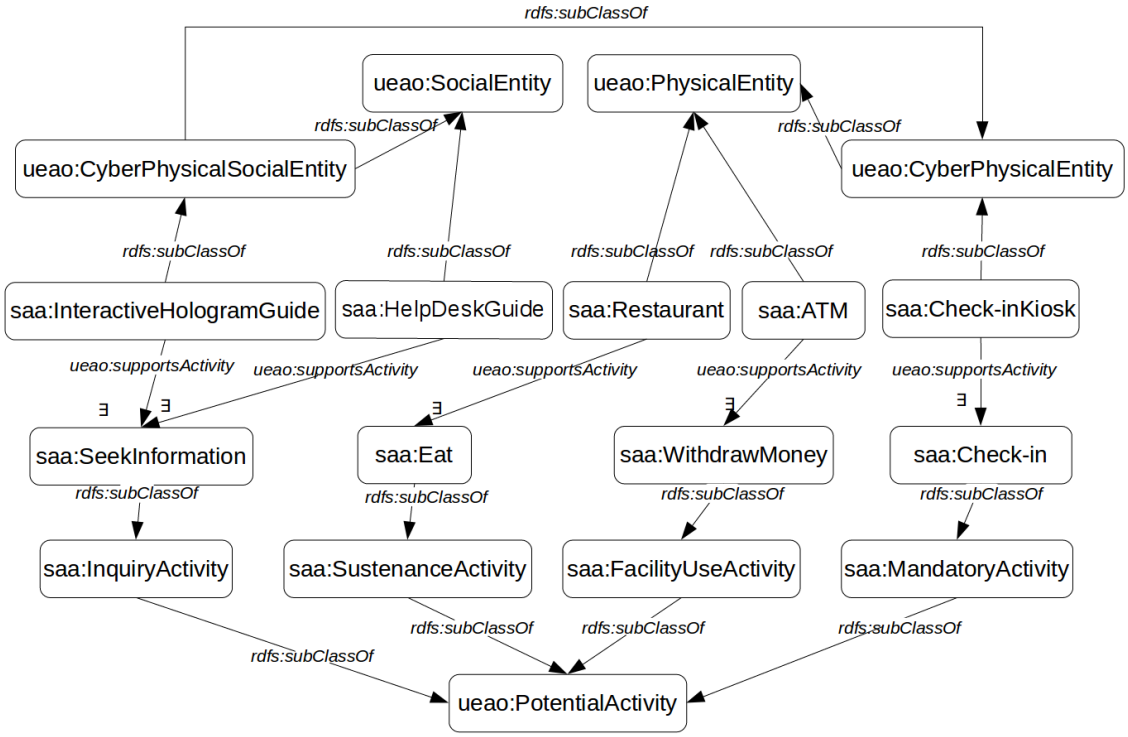

Figure 3.7: A partial view of Smart Airport Activity Ontology

3.3.2 Demonstration

For the purpose of demonstration, in this section, we use the proposed model and ontologies to describe partly a terminal of an airport as shown in Figure 3.8[3].

[3]Photo credits: http://www.klia2.info/about-klia2/klia2-layout-plan/klia2-departure-hall

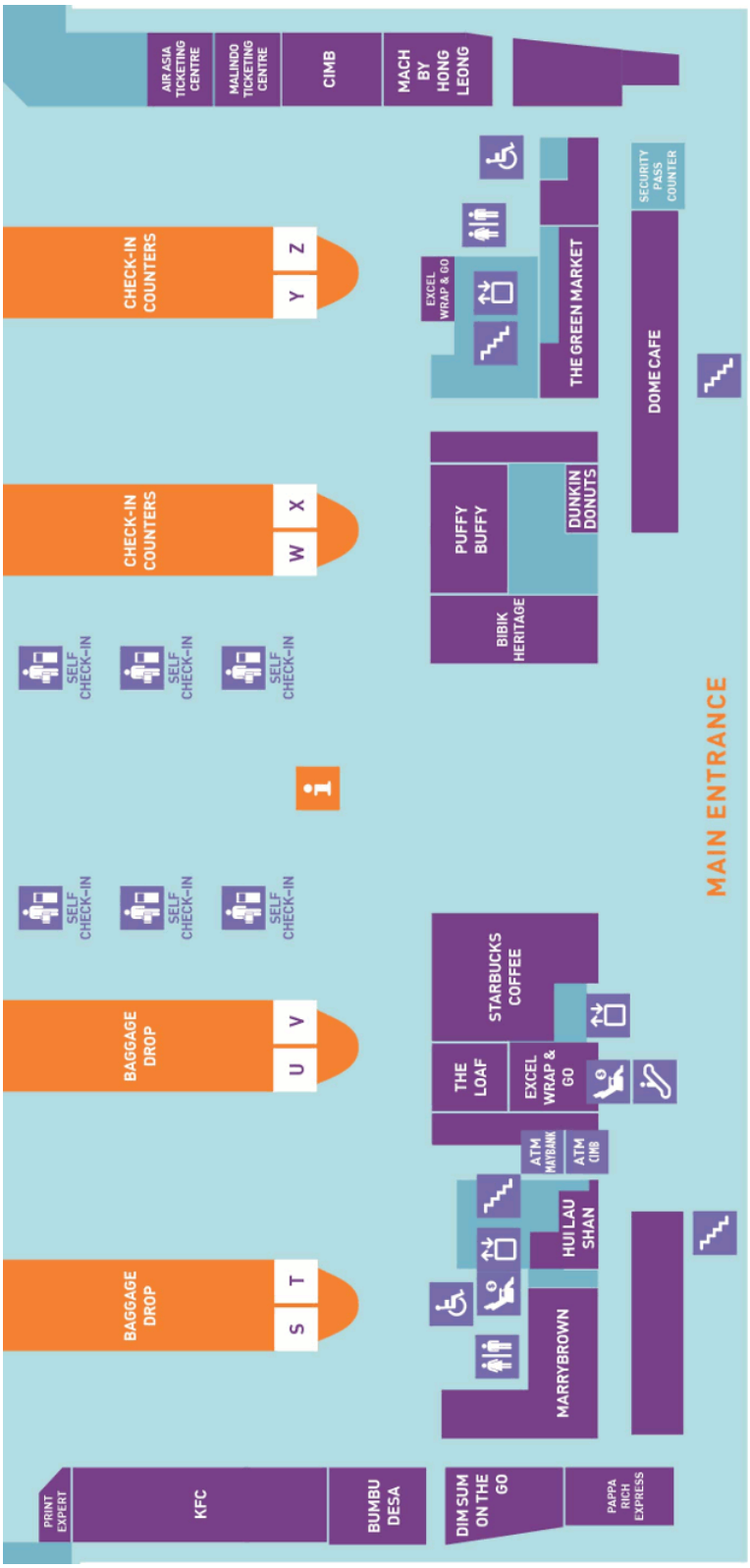

Figure 3.8: An excerpt of a map of an airport

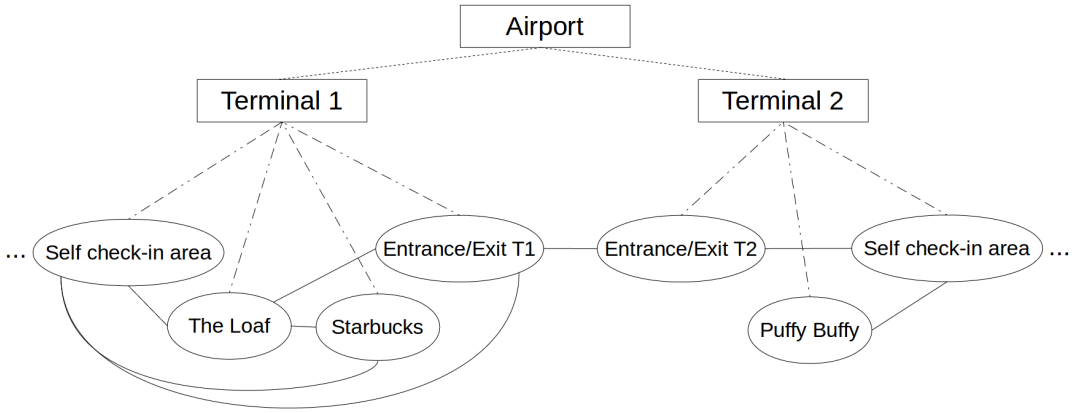

Figure 3.9: Example of an airport modelled using our knowledge model

First, we abstract the environment using our knowledge model integrating various aspects of the environment as illustrated in Figure 3.9. Then, we use the proposed ontologies to provide a description of the abstracted environment. In this example, we use the following prefixes:

- Base URI:
  `@base <http://www.example.org/environment/description/>`

- URI of the `ueao` ontology:
  `@prefix ueao: <http://www.semanticweb.org/ontologies/2018/1/ueao#>`

- URI of the `saa` ontology:
  `@prefix saa: <http://www.semanticweb.org/ontologies/2018/1/saa#>`

- URI of DBpedia resource:
  `@prefix dbr: <http://dbpedia.org/resource/>`

Starbucks coffee

In this example, we describe the location that contains Starbucks coffee and its connection to the self check-in area.

```
<Starbucks-location>
   a ueao:Location ;
   ueao:containsEntity <Starbucks-coffee> ;
   ueao:hasConnection <Connection-Starbucks-SelfCheckinArea> .

<Starbucks-coffee>
      a saa:CoffeeShop ;
      a dbr:CoffeeHouse ;
      ueao:hasEntityResource <https://www.starbucks.com/> .

<Connection-Starbucks-SelfCheckinArea>
      a ueao:Connection ;
      ueao:hasOrigin <Starbucks-location> ;
      ueao:hasDestination <SelfCheckinArea> ;
      ueao:hasConnectingResource <PathWay-5-Sensor-Resource> .
```

Self check-in area

In this example, we describe the location that is the area containing self-checkin kiosks.

```
<SelfCheckinArea>
      a ueao:Location ;
      ueao:containsEntity <Check-inKiosk1>, <Check-inKiosk2>;
      ueao:containsEntity <Check-inKiosk3>, <Check-inKiosk4>;
      ueao:containsEntity <Check-inKiosk5>, <Check-inKiosk6>.

<Check-inKiosk1>
      a saa:Check-inKiosk ;
               ueao:hasEntityResource <Checkin-Kiosk-Resource> .
<Check-inKiosk2>
      a saa:Check-inKiosk ;
               ueao:hasEntityResource <Checkin-Kiosk-Resource> .
<Check-inKiosk3>
      a saa:Check-inKiosk ;
               ueao:hasEntityResource <Checkin-Kiosk-Resource> .
<Check-inKiosk4>
      a saa:Check-inKiosk ;
               ueao:hasEntityResource <Checkin-Kiosk-Resource> .
<Check-inKiosk5>
      a saa:Check-inKiosk ;
               ueao:hasEntityResource <Checkin-Kiosk-Resource> .
<Check-inKiosk6>
      a saa:Check-inKiosk ;
      ueao:hasEntityResource <Checkin-Kiosk-Resource> .
```

3.4 Conclusion

In this chapter, we address the Research question 1 which is *how to acquire the information necessary for solving a multi-goal pathfinding problem?* To this end, we described our proposal for modelling and describing ubiquitous environments for multi-goal pathfinding. Our contributions presented in this chapter are the followings:

- a conceptual knowledge model for abstracting a ubiquitous environment integrating spatial, cybernetic, physical, and social dimensions with the notions of activities;
- an ontology for formally describing an environment based on the proposed knowledge model, which is extensible to support different environments;
- an ontology designed to capture the specifics of a smart airport based on the `ueao` ontology.

Solving multi-goal pathfinding in ubiquitous environments necessitates the accesses to relevant resources to retrieve useful up-to-date information about the environment. However, accessing various resources results in latency. In the next chapter, we present our collaborative search model for addressing the latency issue by collaboratively searching and accessing to resources in an asynchronous manner.

Chapter 4

Collaborative Multi-agent Search Model

Contents

Generally, during each iteration, a forward-search algorithm *selects* a node from the list of candidate nodes *Frontier*, *generates* its child nodes, *prunes* some unpromising nodes, and *updates Frontier* to include the remaining children [Ghallab 2016]. In classical search settings, the information required to compute the cost of a node is provided as a part of the problem. However, in ubiquitous environments where the state of the environments may evolve over time, it is impossible to provide complete and accurate knowledge of the environments prior to the search process. For example, suppose that a traveller in an airport wants to go from an entrance to a gate and that there are two possible paths to go to the gate: one using an elevator which requires less time to travel and one using the stairs which takes more time. At the beginning of the search, the elevator functions normally; however, during the search, the elevator becomes out of order. In such a situation, the path using the stairs should be the solution. Moreover, a solution to the same problem at time t may not be valid at time t' as the state of the environment may evolve and some of

the changes might have affected the parts of the solution. To be able to take into account such dynamic changes of the environments, up-to-date information about the environments is needed to find a solution that is valid with regard to the current state of the environments. Such information is retrieved from various sources such as the cybernetic, physical, and/or social entities located in the environments as well as the resources on the Web during path computation. In the previous chapter, we presented a knowledge model for abstracting a ubiquitous environment, providing the knowledge regarding the sources of information from which up-to-date information can be retrieved.

Accessing and retrieving data from various sources for up-to-date information create the overheads that result from the latency of processing and transferring data. Latency affects the performance of the search, specifically during the generation of child nodes where the cost of nodes is computed using the information retrieved from various resources. To address this latency issue, we propose a collaborative search model that mitigates latency by asynchronising the process of accessing resources and that is based on the distribution of processes to improve search efficiency. The aim of this chapter is to present the collaborative search model. The rest of the chapter is organised as follows. First, we provide an overview of the search model. Second, we present different types of agents in our search model in details. Finally, we summarise and conclude the chapter.

4.1 Collaborative Search Model

The collaborative search model is a multi-agent search model that is composed of multiple agents collaboratively searching on different parts of the search space. The aims of this search model are: mitigating latency and improving search efficiency. It addresses the latency issue by asynchronising the process of accessing resources such that agents are not required to wait for information and that the agents are able to perform other tasks while the information is being retrieved and transferred to the agents. Furthermore, it exploits the knowledge of the search space to distribute the work efficiently amongst the agents to improve search performance.

4.1.1 Overview of the model

This model can be applied to forward-search algorithms such as breadth-first search, depth-first search, uniform-cost search, and A*. More precisely, to distribute the workloads, agents explore different parts of the search space in parallel by executing a search algorithm, thus *selecting* nodes from their respective part of the search space. *Child generation* is modified into an asynchronous and non-blocking process where agents are able to execute other tasks while the necessary information is being retrieved from resources. When an agent discovers a path to a node that is not in its part of the search space, it communicates the information about the path to the agent that is in charge of the node for *pruning* and *updating* its local search process.

Definition 5 ***Collaborative multi-agent search model***
A collaborative search model is a tuple $CSM = (E_t, n_o, n_d, SAM, RAM, NAM, CR, f)$ *where*

- E_t *is the description of the environment at time* t *as previously defined in Definition 4;*
- $n_o \in L_t$ *is a node representing an origin location where* $L_t \in E_t$ *is a set of locations in the environment;*
- $n_d \in L_t$ *is a node representing a destination location;*
- SAM *is the model for search agents executing the search algorithm;*
- RAM *is the model for resource agents responsible for retrieving information from a set of resources;*
- NAM *is the model for network agents in charge of managing the search process and related communications;*
- CR *is a set of criteria for path evaluation;*
- f *is a cost function used to evaluate the path.*

A search algorithm adapted using the collaborative search model takes 2 elements as inputs: the description of the environment (defined in Definition 5 as E_t) and the request containing the origin node (n_o), the destination node (n_d), and a set of criteria (CR) for path evaluation. The representation of the environment provides the knowledge necessary for performing the search such as the spatial topology of the environment represented as a search graph, the organisational structure of the environment, the information regarding the cybernetic, physical, and/or social entities located in the environment, and the resources that can be accessed to retrieve the necessary information for computing path costs. The output of a search process is the optimal path from n_o to n_d, which is a minimum-cost path evaluated using the cost function f based on the given set of criteria CR.

The multi-agent model is composed of 3 types of agents: search agents, network agents, and resource agents. The search agents are the agents in charge of executing a given search algorithm to find the path. The network agents are responsible for distributing workloads amongst search agents, and they handle the communications amongst the search agents. The resource agents serve as the intermediary between the search agents and the resources. Their main role is to access resources to retrieve the information requested by the search agents, which abstracts away the complexity of accessing resources from the search agents.

To describe in a concise manner, the collaboration among the 3 types of agents to solve a search problem is as follows. At the beginning of a search process, there is a set of resource agents that are capable of accessing resources and one initial search agent Sa^0 that starts the search process. Sa^0 executes the search algorithm. After

expanding the first node, Sa^0 generates the child nodes. Sa^0 sends information about the child nodes that are not under its responsibility (i.e., not in the part of its search space) to its parent network agent Na^0. Na^0 communicates with other network agents to forward the information about the child nodes to the search agent(s) responsible. Based on various criteria for workload distribution (described in 4.4.3), Na^0 decides which existing search agent should be in charge of the child nodes, or creates a new search agent to handle the child nodes when necessary. When assigned to a node, a search agent sends a request to a resource agent to retrieve the necessary information for computing the cost of the node.

Furthermore, network agents use the organisational structure of the environment to progressively create a set of network agents to handle the communications amongst the search agents. In this way, the search agents are not required to have knowledge of the other search agents. The search process continues until the destination node n_d is found by a search agent, in which case the goal verification procedure (described in 4.3.3) is triggered to validate the optimality of the path. The search process terminates when the optimal path is found or when the entire search space has been explored.

4.1.2 Agent models

To enable a collaborative search, each type of agent is assigned with a set of roles to perform. In addition, they possess the knowledge necessary to perform their tasks. The knowledge related to the ubiquitous environment in which the problem is located is available in the abstraction of the environment described using the knowledge model presented in Chapter 3. In this section, we present the model for each type of agents involved in the search model.

Resource agent model

Resource agents serve as an intermediary between search agents and resources. They handle requests for information from search agents. The role of a resource agent is to access resources to retrieve the requested information. The model of a resource agent is as follows:

$$Ra = (I_R, Req)$$

where:

- I_R is knowledge on how to access a set of resources;
- Req is a list of requests that the resource agent has received and that need to be processed.

The knowledge of each resource agent is provided to the agent when it is created. The motivation behind using resource agents is to separate the search process executed by search agents from the process of accessing resources. In this way, search

agents are dedicated only to finding the solution, and delegate the tasks that have latency to resource agents.

Search agent model
The role of a search agent is to execute a search algorithm in a part of the search space for which it is responsible. The collaborative search model is based on the distribution of search processes. Therefore, search agents explore different parts of the search space in parallel to find the solution, and at the same time, collaborate with each other by communicating information deemed pertinent to one another. To perform its role, a search agent is modelled as follows:

$$Sa = (S, L_{Sa}, I_{RA}, Na)$$

where:

- S represents a search algorithm to execute;
- $L_{Sa} \subseteq L_t$ is a set of nodes in the search space of which it is in charge;
- $I_{RA} = ((res, RA))$ is the information about resource agents which contains pairs of resource type res and a set of resource agents RA that are capable of accessing resources of type res;
- Na is the network agent which is the parent of the search agent.

Network agent model
The roles of a network agent are the followings:

- Coordinate the communications amongst search agents;
- Separate the search space into different parts for search agents and distribute workloads amongst search agents.

Each network agent is in charge of a part of the search space. It handles the communications and workload distribution relevant to its part of the search space. To perform its role, each network agent is equipped with the knowledge as follows:

$$Na = (OS_{Na}, Na_p, SA_{Na}, NA_{Na})$$

where:

- $OS_{Na} \subseteq OS_t$ is a set of organisational entities which represent the part of the search space of which the network agent is in charge;
- Na_p is the its parent network agent;
- SA_{Na} is a set of search agents that are under its management;
- NA_{Na} is a set of network agents that are under its management.

Using network agents enables us to separate the search process from the communications and workload distribution.

4.1.3 Agent organisation and interactions

The foundation of our search model is the collaboration amongst search agents, network agents, and resource agents. In this section, we present the organisation and interactions that enable them to cooperate to realise the objectives of the model, namely, addressing latency and improving search efficiency.

Addressing latency
During the search process, latency results from the access to resources to retrieve information for computing the path cost. To handle this issue, in our search model, we separate the process of accessing resources from the search process by employing search agents to perform the search and resource agents to retrieve information from resources.

Figure 4.1 illustrates an example of an organisation of search agents (abbreviated as SA in the figure) and resource agents (abbreviated as RA in the figure) to address the latency issue. Each search agent is in charge of a part of the search space. How a search space is divided amongst search agents is presented in Section 4.4.3. For simplicity, in this example, we suppose that there is one search agent for each Zone. The search agent Sa_1 is in charge of the part of the search space that contains node L1 and node L3. Each resource agent is capable of accessing a set of resources. For instance, the resource agent Ra_1 is able to access resource ER1, ER2, CR1, and CR2. To compute the cost of L1 to L4, Sa_1 needs the information from resource CR1. Therefore, Sa_1 sends a request to Ra_1 to retrieve the necessary information.

It is essential to note that the interactions between a search agent and a resource agent to handle a request for information from a resource is asynchronous and non-blocking. After sending a request to a resource agent, the search agent proceeds with other tasks, and is not blocked while the resource agent is accessing the resource. Once the information has been retrieved, the resource agent sends the information to the search agent.

Improving search efficiency
In this collaborative search model, we combine *distribution* with *parallelism* to improve search efficiency. Different parts of the search space are *dynamically distributed*

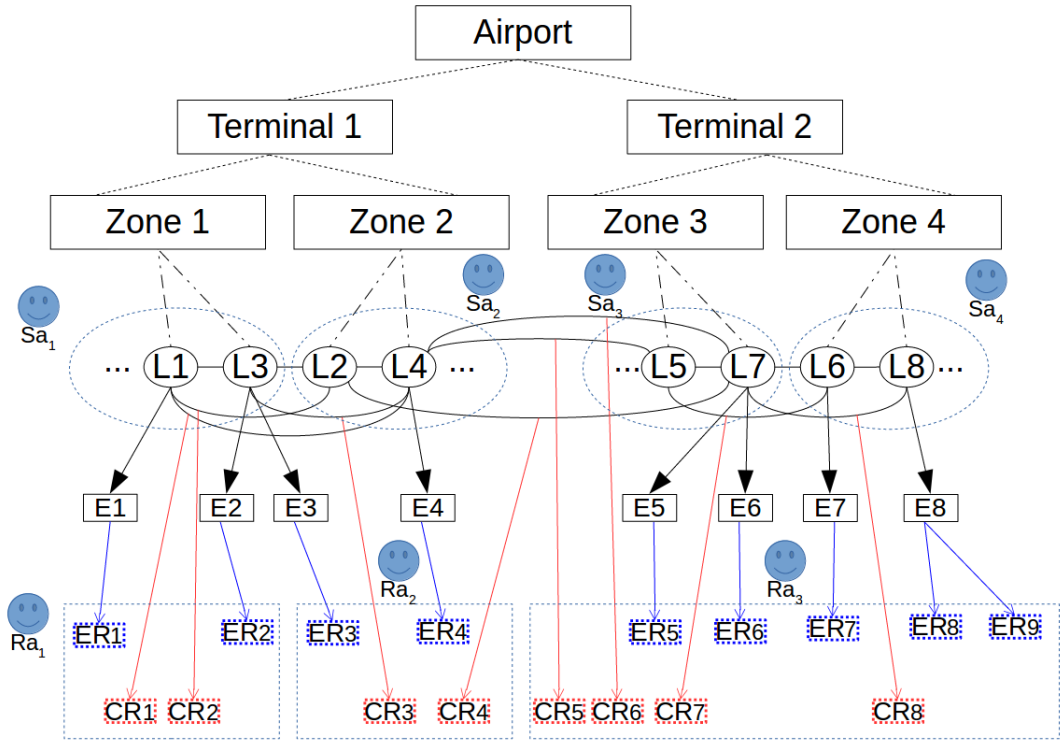

Figure 4.1: Agent organisation - addressing latency

to different search agents that explore their respective part *in parallel*. Distribution of the search space allows more parts of the search space to be explored at the same time. Parallelism of the search process enables faster exploration of the search space. Such distribution and parallelism are achieved through the organisation of and interactions between search agents and network agents. Network agents are responsible for dividing the search space into smaller parts and assigning them to search agents. The separation of the search space and the assignment are carried out dynamically during the search process. In this way, only the parts that are relevant to the current search problem are explored.

A search space is divided based on the organisational structure of its environment, as described in the previous chapter (Section 3.1). We exploit such a structure to organise network agents. Each network agent is in charge of a set of organisational entities, and thus the part of the search space under those entities. Let us consider an example of the organisation of network agents depicted in Figure 4.2. The organisational structure of the environment is a hierarchy (Airport - Terminal - Zone), which contains organisational entities such as Terminal 1, Zone 1, and Zone 2. In this example, we suppose that the origin node and the destination node are under Terminal 1, so we only need to create the network agents to manage Terminal 1. Network agent Na_{T1} is in charge of Terminal 1, Na_{T1Z1} of Zone 1, and Na_{T1Z2} of Zone 2. Na_{T1} is the parent agent of Na_{T1Z1} and Na_{T1Z2}. When Na_{T1Z1} or Na_{T1Z2} discovers a path to a node that is out of its responsibility scope (Zone 1 or

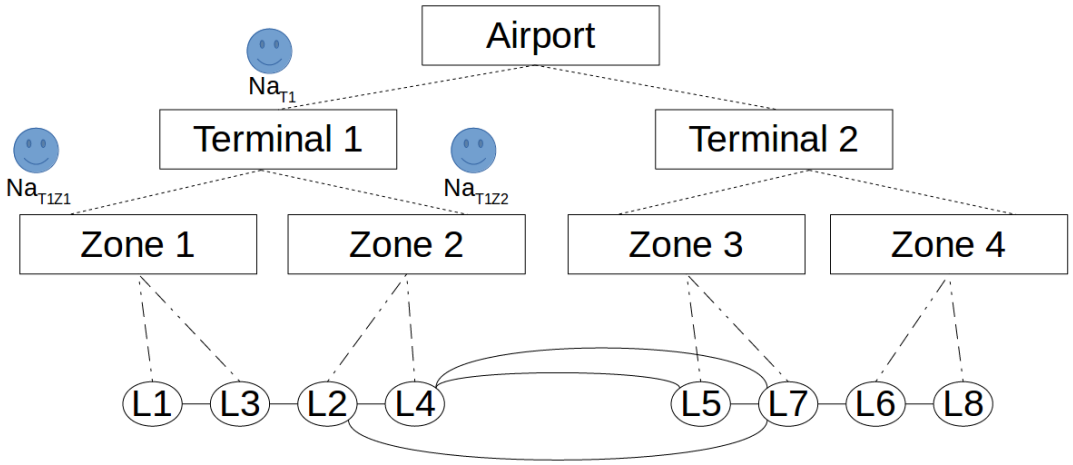

Figure 4.2: Organisation of network agents

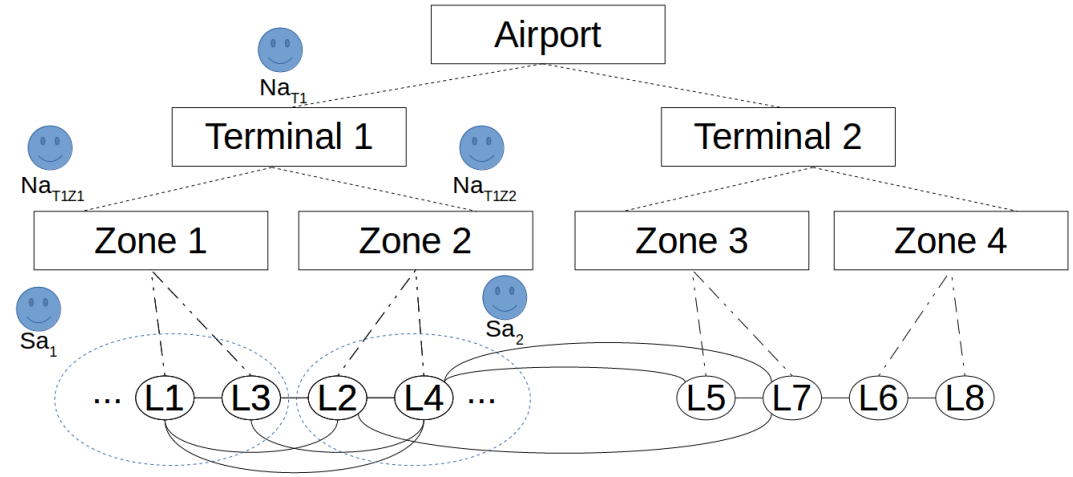

Figure 4.3: Organisation of network agents and search agents

Zone 2), it forwards the information to its parent agent to handle. The parent agent decides which existing agent to whom it should assign the newly discovered node or creates a new network agent to handle the organisational entity under which the new node is located.

Distributing workloads to search agents is handled by network agents. The network agents that are in charge of the organisational entities directly containing the nodes decide which nodes are to be assigned to which search agents and whether a new search agent should be created to handle the nodes. For instance, in the example illustrated in Figure 4.3, network agent Na_{T1Z1} and Na_{T1Z2} individually have a search agent to explore the parts of the search space under the organisational entities, Zone 1 and Zone 2 respectively, of which they are in charge.

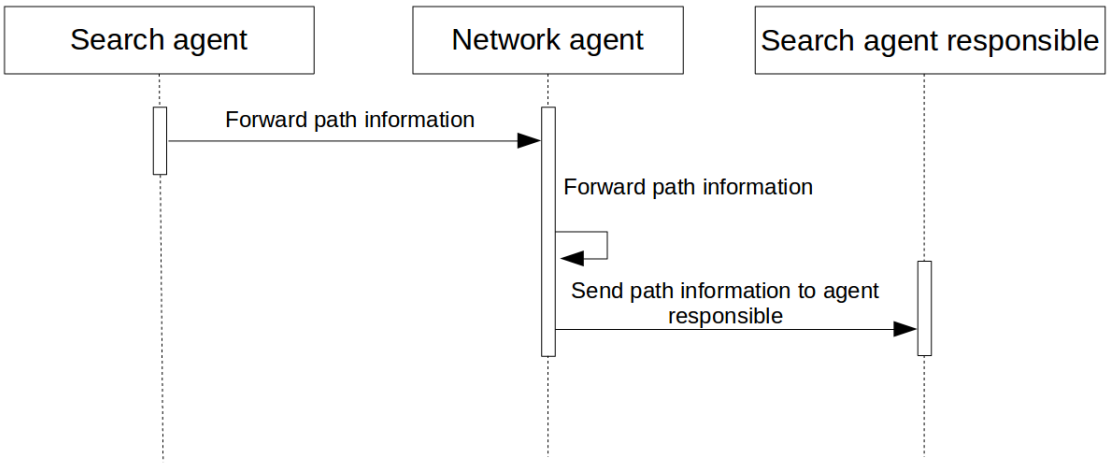

Figure 4.4: Interaction between search agents and network agents for routing path information

Addressing consequential issues

One of the most important issues of distributed and parallel search is not knowing the global state of the search process as each search agent executes the search locally. This issue commonly results in *redundant expansion of nodes* and *compromising optimality* of the solution. The proposed organisation of the network agents and search agents allows us to address this issue, in addition to search space separation and workload distribution.

First, to avoid the same node from being expanded by multiple search agents, a node should only be expanded by the search agent in charge of the node. Therefore, if another agent discovers a path to the node, it communicates that information to the search agent in charge without expanding the node. Figure 4.4 illustrates the interaction between search agents and network agents for routing the information about the path to a node to a search agent in charge. When a search agent discovers a path to a node that is not under its responsibility, it forwards the node to its parent agent, which is a network agent. The parent agent routes the node via other network agents (routing protocol described in Section 4.4.1) until the search agent in charge is found, in which case the information about the node is sent to that search agent.

Second, to determine the optimality of a solution, the global state of the search is required. In a parallel and distributed search, such information is unavailable as each search agent performs the search locally. In our model, to guarantee the optimality of the solution, we employ a goal verification procedure (described in Section 4.3.3). This organisation of search agents and network agents enables the communications amongst the search agents that are needed to verify the optimality of the solution.

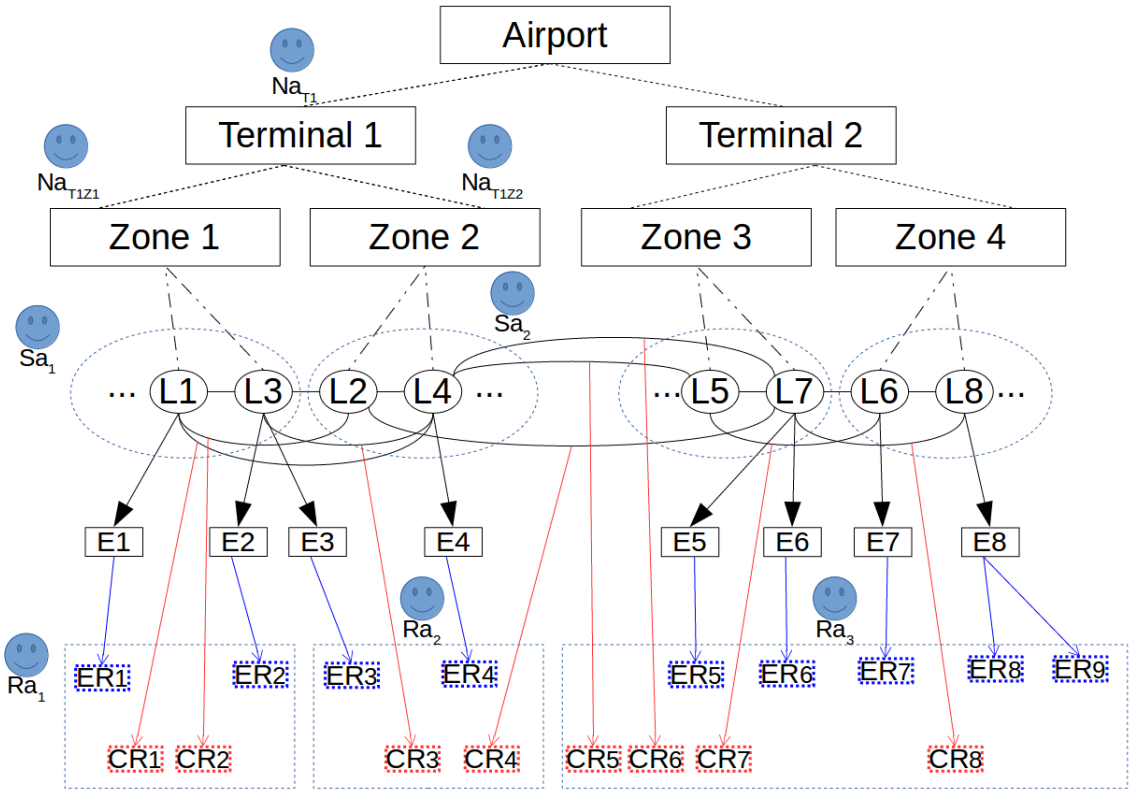

Figure 4.5: An example of the overall organisation of agents

Global view of the organisation

Figure 4.5 demonstrates an example of an organisation of the 3 types of agents. Each type of agents handles a separate concern, and agents are organised in a way that their collaboration is independent of the mechanisms they use to handle their respective concern. For instance, the methods resource agents employ to access resources have no influence on their collaboration with other types of agents. The same goes for the mechanisms used by network agents to manage communications and workload distribution, and for the search algorithm executed by search agents. In the following sections, we present how each type of agents is modelled.

4.2 Resource agents

The role of resource agents in our collaborative search model is to facilitate the access to resources to retrieve information. Resource agents can be considered as an interface to resources as they abstract away the complexity of accessing various types of resources. Search agents interact with resource agents to request for information from resources to compute the cost of nodes while executing a search algorithm. At the beginning of a search process, a set of resource agents that can access the resources relevant to the search space are created. The information about resource

agents is included in the knowledge base of search agents so that they are able to interact with resource agents.

Algorithm 3 Resource agent life cycle

```
1: Messages ← ∅
2: while true do
3:   for each message in Messages do
4:     if (message is an information request message) then
5:       create a process pr
6:       run process-request-information-message(message) on pr
7:     else if (message is a termination message) then
8:       terminate process
9:     end if
10:  end for
11: end while
```

Algorithm 4 process-request-information-message(*message*)

```
1: searchAgent ← get the search agent who sent the request from message
2: resource ← get the resource from message
3: response ← access-resource(resource)
4: send response to searchAgent
```

The execution cycle of a resource agent is illustrated in Algorithm 3. Each resource agent maintains a list of messages that it receives. At each cycle of execution, a resource agent processes all the received messages. When receiving an information request message, the resource agent creates an independent process to handle the request. Handling an information request is a time-consuming task as it involves accessing a resource to retrieve the requested information. Therefore, handling such a request on a separate process enables the resource agent to proceed with other tasks. Algorithm 4 shows how a resource agent handles an information request message. This algorithm requires the agent to access a resource (Algorithm 4, Line 3). The mechanisms required to access the resource is incorporated in the knowledge of the resource agent when it is created.

4.3 Search agents

In our collaborative search model, the main role of search agents is to execute a search algorithm to find the solution. The execution cycle of a search agent consists of *expanding a node*, *computing the cost of a child node*, and *processing received messages* as demonstrated in Algorithm 5. To perform its role, each search agent maintains the following data:

- a set of nodes *ResponsibledNodes* (L_{Sa}) representing the part of the search space of which the search agent is in charge,

- a set of candidate nodes ordered by node cost($Frontier \subset ResponsibledNodes$) for expansion,
- a set of expanded nodes ($Expanded \subset ResponsibledNodes$),
- a list of arcs ($ArcList$) where an arc is a path between two nodes and whose cost is to be computed,
- a list of arcs whose cost is being computed ($PendingArcList$),
- a list of received messages to be processed.

Each search agent is responsible for exploring a part of the search space. The nodes constituting the part of the search space are dynamically and incrementally assigned by the network agent parent of the search agent, and are stored in $ResponsibledNodes$. The dynamic assignment and distribution of workloads are presented in depth in Section 4.4.3.

Algorithm 5 Search agent life cycle (n)

```
1: ResponsibledNodes ← {n}
2: Frontier ← {n}
3: Expanded ← ∅
4: ArcList ← ∅
5: PendingArcList ← ∅
6: Messages ← ∅
7: while no termination message received do
8:    expand()
9:    compute-arc-cost()
10:   process-messages()
11: end while
```

4.3.1 Node expansion

As in a forward-search algorithm, each search agent Sa maintains a set of candidate nodes to expand $Frontier$ and a set of expanded nodes $Expanded$. In each iteration, Sa executes a search algorithm which starts by selecting a node n from $Frontier$ to expand, as illustrated in Algorithm 6. If n is the destination node, Goal Verification Procedure is initiated (described in Section 4.3.3) to verify the optimality of the path to the destination node, and Sa continues its execution until the path is verified or a better path to the destination node is found. Otherwise, it generates the child nodes of n. To generate each child node n', we need to compute its cost, which is the sum of the cost of n and the cost of the arc from n to n'. The cost of n is known because before adding a node to $Frontier$, its cost is computed. However, the cost of the arc from n to n', denoted as $a(n, n')$, is to be computed by sending a request to a resource agent to acquire the necessary information for the computation.

Algorithm 6 expand()

```
1: n ← POP(Frontier)
2: if n is a goal node then
3:    initiate Goal Verification Procedure
4:    return
5: end if
6: for each child of n do
7:    create a(n, child)
8:    if child ∈ ResponsibledNodes then
9:       add a(n, child) to ArcList
10:   else
11:      send a(n, child) to parent agent
12:   end if
13: end for
```

If n' belongs to the part of the search space of which the Sa is in charge (i.e., in its $ResponsibledNodes$), Sa stores $a(n, n')$ in its $ArcList$ to compute the arc cost later. Otherwise, Sa sends $a(n, n')$ to the search agent Sa' that is responsible for n' through the parent agent of Sa which is a network agent (refer to Section 4.4.1 for the description of routing path information). Upon receiving $a(n, n')$, Sa' discards $a(n, n')$ if it already knows the path to n' with a better cost than the path through n; this prevents Sa' from requesting for information to compute the cost of the path from n to n', which is clearly in a non-optimal path. Otherwise, Sa' adds $a(n, n')$ to its $ArcList$ for cost computation later.

For the purpose of demonstration, refer to Figure 4.5. Take the search agent Sa_1 as an example. Sa_1 is in charge of L1 and L3, so its $ResponsibledNodes$ contains L1 and L3. When Sa_1 expands L1, it discovers a path to L2, L3, and L4. Since L3 is under its responsibility, Sa_1 stores the $a(L1, L3)$ in its $ArcList$. For L2 and L4, Sa_1 sends them to its parent agent to forward them to the search agent(s) responsible.

4.3.2 Node cost computation

After node expansion, the search agent Sa pops one of the arcs in $ArcList$ to compute its cost as shown in Algorithm 7. The cost of an arc is determined using the information retrieved from resources associated with the arc. To obtain such information, Sa sends a request to one or multiple resource agents associated with the resources using the information about resource agents (I_{RA}). Depending on the type of resource (e.g., an API, a database), Sa chooses a resource agent to inquire.

Algorithm 7 compute-arc-cost()

```
1: arc ← POP(ArcList)
2: send a request for information about arc to a resource agent
3: add arc to PendingArcList
```

While the information is being retrieved, the arc is moved from *ArcList* to *PendingArcList* where all the arcs pending for requested information are stored. Computing an arc cost is a non-blocking process. After sending the request to a resource agent, *Sa* continues its execution. Once the necessary data is acquired, the resource agent sends it to *Sa*. *Sa* uses the information to compute the cost of the arc and the cost of the child node to which the arc points. This asynchronous mechanism for retrieving information enables search agents to perform other tasks while resources are being accessed, thus mitigating the latency.

4.3.3 Goal verification procedure

In this collaborative search model, each search agent does not possess global knowledge of the search state. To guarantee the optimality of the solution, we propose a goal verification procedure that is used to verify the optimality of the solution. When a destination node n_d is expanded, the expanding search agent *Sa* initiates the goal verification procedure. The objective is to determine whether the found path leading to n_d is the minimum-cost path. The verification is conducted in a distributed manner by each search agent. A path is verified as a minimum-cost path only if all the search agents involved in the search process reach a consensus about the validity of the path.

Local verification

A search agent performs location verification of a solution when it receives a goal verification request message. For each search agent, a path to n_d is optimal if there exists no node n where $f(n) < f(n_d)$. To verify this property, each search agent performs the following verification:

- If there is any node n in *Frontier* where $f(n) < f(n_d)$, the path is not verified;
- If there are arcs in *ArcList* or *PendingArcList*, the path is not verified. The cost of those arcs are still unknown, so it is impossible to determine the cost of the nodes to which those arcs point.

The procedure

To start this procedure, the initiator search agent *Sa* sends a goal verification request message to its parent agent *Na*. Upon receiving the message, *Na* executes the propagation protocol, illustrated in Algorithm 8, to propagate the message to other agents. *Na* sends the message to its child agents (both network agents and search agents) and its parent agent (network agent) that are not the source agent of the message it has received. Each network agent that receives the message repeats the propagation process by following the propagation protocol.

Figure 4.6 illustrates an example of the propagation of goal verification messages. In this example, the search agent Sa_1 initiates the Goal Verification Procedure, so it sends a goal verification request to its parent, the network agent Na_1. Na_1 follows the propagation protocol and forwards the message to its child agent Sa_2 and its

Algorithm 8 propagate(requestMessage)

1: *sourceAgent* ← the agent who sent the request message
2: **if** (*parentAgent* != NULL and *parentAgent* != *sourceAgent*) **then**
3: send *requestMessage* to *parentAgent* //sourceAgent of this message is the current agent that sends the message
4: **end if**
5: **for** (each *childAgent*) **do**
6: **if** (*childAgent* != *sourceAgent*) **then**
7: send *requestMessage* to *childAgent*
8: **end if**
9: **end for**

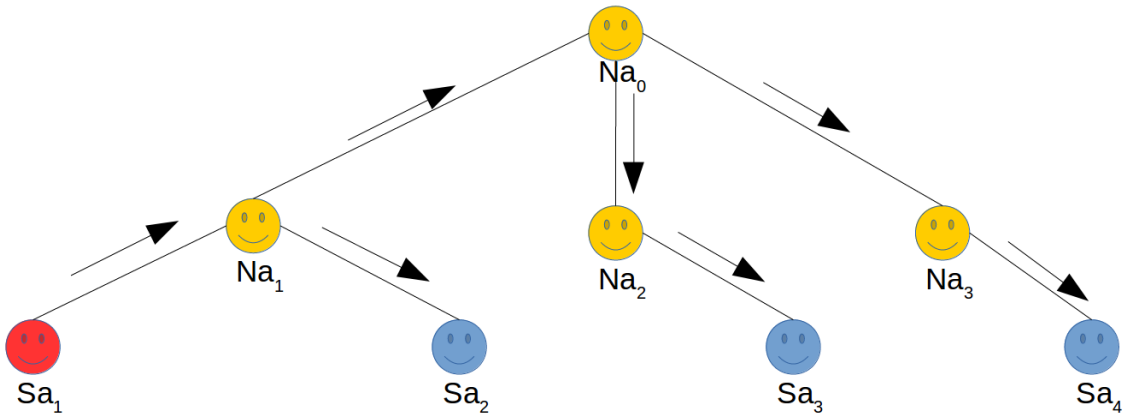

Figure 4.6: Goal verification message propagation example

parent agent Na_0. The process continues until all the search agents have received the request.

Search agents that receive the message perform local verification at every execution cycle until the path is verified or a better path is found. If the found path is verified by a search agent, the search agent sends a response message to its parent agent (network agent). When receiving a response message, network agents execute the back-propagation protocol, demonstrated in Algorithm 9 to forward the response message to the source of the request message. Therefore, eventually, the initiator search agent receives a single response message signifying the validity of the path directly from its parent agent. If a search agent founds a better path to the goal during node expansion, it initiates another goal verification procedure to replace the previous one.

In addition to validating the optimality of a path, the goal verification procedure enables search agents to filter unpromising nodes and arcs. When the found path is under local verification by a search agent, the knowledge about the found destination such as the path cost is used to discard unpromising nodes and arcs from *Frontier* and *ArcList*, respectively. For instance, suppose the path under local verification

Algorithm 9 back-propagate(responseMessage)

1: *sourceAgent* ← the agent who sent the request message
2: **if** (for each request message forwarded, a response message has been received) **then**
3: send *responseMessage* to *sourceAgent*
4: **end if**

has a cost x. Then, any node in $Frontier$ having a higher cost and any arc in *ArcList* whose origin node has a higher cost are not optimal and should be discarded.

4.3.4 Optimality and Termination

Goal verification procedure enables us to determine whether a path is a minimum-cost path. However, whether a path is optimal depends on the actual algorithm and the cost function that the algorithm uses. For instance, in uniform-cost search, the cost of a node is the cost from the start node to the node, which guarantees optimality. In such a case, a path verified by the goal verification procedure is an optimal path. However, for a greedy algorithm, a path verified by the goal verification procedure is a minimum-cost path based on the algorithm's cost function, but not necessarily an optimal path based on the set of given criteria. Such flexibility in terms of search algorithms used is possible thanks to the modularity of the proposed search model where different search algorithms can be used depending on the problem to solve. In this way, we are able to benefit the properties belonging to each specific algorithm.

The model terminates by finding a minimum-cost path if one exists, assuming the following properties:

- The search space is finite;
- The information about the search space is accurate;
- All messages arrive at their destinations;
- For every request to resource agents for information, we get a response.

A search process is terminated when a path verified by the goal verification procedure is found or when the entire search space has been explored. Naturally, the end of a search space is reached when all the search agents involved in the search process have no nodes in $Frontiers$, no arcs in *ArcList* and no pending arcs in *PendingArcList*.

4.4 Network agents

The role of a network agent is to manage and facilitate the search process of a part of the search space of which it is in charge. This management entails *routing*

path information to search agents, *separating a search space into different parts and assign them to different network agents*, *distribute workloads amongst search agents*, and *propagate goal verification procedure messages*. The execution cycle of a network agent is shown in Algorithm 10. To perform its roles, each network agent maintains the following data:

- a set of organisational entities *ResponsibledEntities* ($OS_{Na} \subset OS_t$) of which it is in charge,
- its parent agent *ParentAgent* (Na_p) which is a network agent that is in charge of the organisational entity that covers the organisational entity of the agent. This is not the case when the network agent is at the top of the organisation (i.e., in charge of the root of the organisational structure);
- a set of network agents *ChildNetworkAgents* (NA_{Na}) that are in charge of an organisational entity under one of its organisational entities,
- a set of search agents *ChildSearchAgents* (SA_{Na}) that it has created to explore the space under one of its organisational entities,
- a list of received messages to be processed.

For the purpose of demonstration, refer to Figure 4.5. Take the network agent Na_{T1Z1} as an example. Na_{T1Z1} maintains the following data:

- Zone 1 which is an organisational entity of which it is in charge, stored in *ResponsibledEntities*,
- Na_{T1} as its parent agent because Na_{T1} is in charge of Terminal 1, which covers Zone 1,
- Sa_1 as its child search agent, stored in *ChildSearchAgents*.

Initially, there is no network agent, and there is only one search agent, which is the initial search agent. Other network agents and search agents required to solve a search problem are not predefined, but are created and configured on-the-fly during the search process. During the routing, network agents and search agents are created dynamically to handle various tasks. The protocol for routing path information, which is the trigger to all the agent creation and task assignment, is presented in the following section.

4.4.1 Routing path information

Path information routing is a mechanism to allow search agents to collaborate amongst themselves by sharing path information. This mechanism also contributes to avoiding inconsistency such as redundant node expansion.

Bootstrapping

At the beginning of the search, there is an initial search agent Sa_0 that is responsible

Algorithm 10 Network agent life cycle ()

```
1: ParentAgent
2: ResponsibledEntities ← ∅
3: ChildNetworkAgents ← ∅
4: ChildSearchAgents ← ∅
5: Messages ← ∅
6: while true do
7:   for each message in Messages do
8:     if (message is a routing path information message) then
9:       path-information-routing-protocol(path information)
10:    else if (message is a goal verification request message) then
11:      propagate(message)
12:    else if (message is a goal verification response message) then
13:      back-propagate(message)
14:    else if (message is a termination message) then
15:      terminate process
16:    end if
17:  end for
18: end while
```

for expanding the origin node n_o. By expanding n_o, Sa_0 discovers a set of paths that lead to other nodes accessible from n_o. At this point, Sa_0 is only in charge of n_o, so each of the discovered paths is to be routed to a search agent in charge. A path or an arc $a(n, n')$ where n is the origin node of the arc and n' is the destination of the arc is to be sent to the search agent in charge of n'.

Sa_0 sends path information to its parent agent to route the information the search agents in charge. However, at this stage, network agents and other search agents have not been created yet. Therefore, Sa_0 creates the first network agent Na to route the information about the paths. Na becomes the parent agent of Sa_0, and it is in charge of the part of the search space under the organisational entity in which n_o is directly located. Upon receiving the path information, Na executes the path information routing protocol, shown in Algorithm 11, to forward the information to the search agent(s) responsible.

For the purpose of demonstration, refer to Figure 4.7 for an example of the bootstrapping phase of routing path information. In this example, the origin node L1 is expanded by the initial search agent Sa_0. Sa_0 creates a network agent Na_{T1Z1}, which is its parent agent. Na_{T1Z1} is in charge of all the nodes under Zone 1 as L1 is located under Zone 1. The information about the organisational structure of the environment such as Zones and Terminals is incorporated in the description of the environment following our knowledge model presented in Chapter 3. To route path information, Sa_0 sends the path information to Na_{T1Z1}. Na_{T1Z1} follows the path information routing protocol.

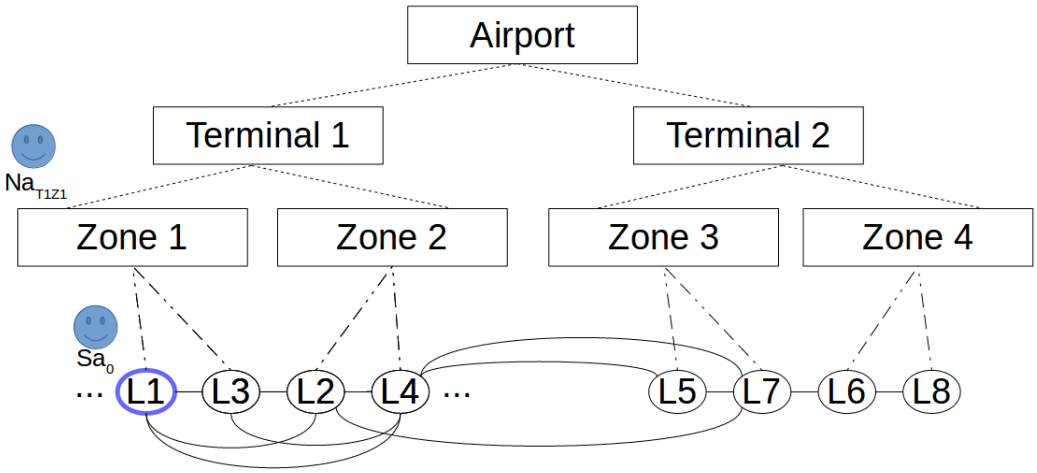

Figure 4.7: Routing path information – Bootstrapping example

Routing protocol

When a search agent Sa discovers a path from node n to node n', denoted as $a(n, n')$, where n' is not under its responsibility, Sa sends $a(n, n')$ to its parent agent Na to route the path information to the search agent Sa' in charge of n'. As mentioned in Section 4.1.3, network agents are organised using the organisational structure of the search space. Therefore, we can exploit the organisational structure to guide the search for Sa'. Sa', if it already exists, is under the management of a network agent that is in charge of an organisational entity in which n' is *directly* located. The routing protocol used by network agents to route path information is described in Algorithm 11.

When a network agent receives path information $a(n, n')$ to route, it retrieves the organisational information of node n' (Algorithm 11, Line 1). For example, suppose that n' is node L2 in Figure 4.7. Then, the organisational information of L2 is Zone 2–Terimal 1–Airport, which means that L2 is under the organisational entity Zone 2, Zone 2 under Terminal 1, and Terminal 1 under Airport. In our routing protocol, depending the organisational entity in which n' is located, the routing process functions differently. There are 3 different cases as follows:

Case 1: Not under any organisational entity of the network agent

In this case, n' is not a part of Na's search space (i.e., neither directly nor indirectly located in any of Na's organisational entities). If the parent agent of Na already exists, Na passes the control to its parent agent to execute the routing protocol by forwarding the path information to the parent agent. Otherwise, Na creates its parent agent to take charge of an organisational entity in which all the organisational entities of Na are located. Then, it forwards the path information to the parent agent. The routing protocol for this case is shown in Algorithm 11 (Line 24 – Line 31).

Figure 4.8 demonstrates an example of this case. After expanding node L1, Sa_0

Algorithm 11 path-information-routing-protocol($a(n, n')$)

1: $organisation \leftarrow$ get the organisational information of n'
2: **if** the executing agent Na is in charge of an organisational entity oe in $organisation$ **then**
3: **if** n' is directly located under oe **then**
4: **if** Na has no search agent that is the agent responsible of n' **then**
5: **if** Na has no search agents OR all search agents cannot take more responsibility **then**
6: Create a new search agent Sa and send $a(n, n')$ to Sa
7: Set Na as the parent agent of Sa and Sa as a search agent of Na
8: **else**
9: Select the search agent with the least responsibility and send it $a(n, n')$
10: **end if**
11: **else**
12: Send $a(n, n')$ to the search agent responsible
13: **end if**
14: **else**
15: Get oe' from $organisation$ where oe' is a direct child organisational entity of he and n' is directly or indirectly located under oe'
16: **if** Na has no child agents OR all child agents cannot take more responsibility **then**
17: Create a network agent Na' and make Na' responsible for oe'
18: Set Na as the parent agent of Na' and Na' as a child agent of Na
19: **else**
20: Assign he' to Na' where Na' is a child agent of Na with the least responsibility
21: **end if**
22: Forward $a(n, n')$ to Na'
23: **end if**
24: **else**
25: **if** Na's parent agent does not exist yet **then**
26: Create a network agent Na^p
27: Make Na^p responsible for the organisational entity that is the direct parent of all of Na's organisational entity(ies)
28: Set Na^p as the parent of Na and Na as the child of Na^p
29: **end if**
30: Forward the request $a(n, n')$ to the parent agent Na^p
31: **end if**

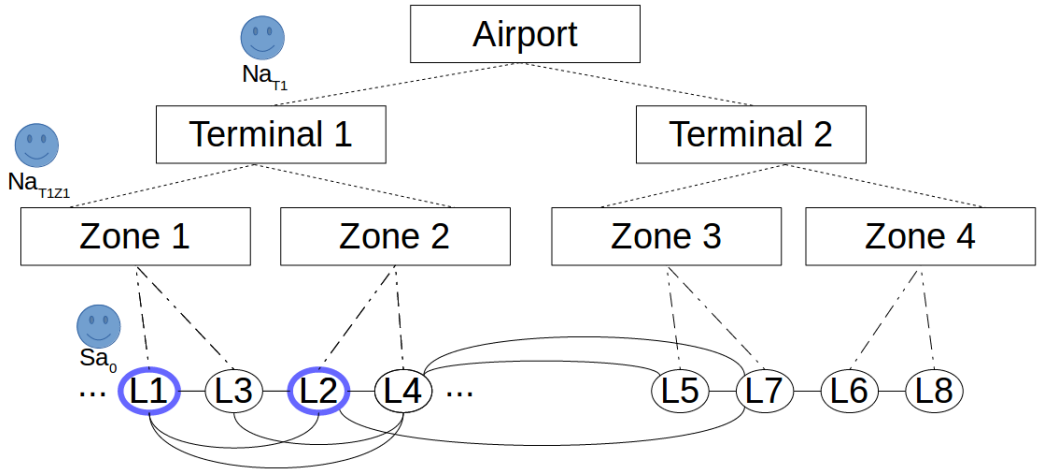

Figure 4.8: Routing path information – Case 1 example

discovers a path to L2, $a(L1, L2)$. Suppose that L2 is not under the responsibility of Sa_0, so Sa_0 sends the path information $a(L1, L2)$ to its parent agent Na_{T1Z1}. Upon receiving the routing request, Na_{T1Z1} executes the routing protocol. However, L2 is not located under any of the organisational entities of which Na_{T1Z1} is in charge. Therefore, it forwards the routing request to its parent agent Na_{T1}, which will execute the routing process.

Case 2: Indirectly under an organisational entity of the network agent In this case, node n' is not directly under any organisational entity of the network agent Na. It is under an organisational entity that is in turn under one of the organisational entities of Na. This means that the search agent Sa' in charge of n' is under the management of a direct or indirect child agent of Na. To route the path information, Na forwards the routing request to the child agent Na'. Na' is a direct child agent of Na. It is responsible for an organisational entity oe that is a direct sub-organisational entity of one of Na's organisational entities and under which n' is directly or indirectly located. If Na does not exist, oe is assigned to a child agent with the least responsibility (see 4.4.2 for further details on separating a search space). If all child agents of Na have reached the responsibility limit, oe is assigned to a new network agent, and the routing request is forwarded to the new network agent, which will continue the routing. The routing protocol for this case is shown in Algorithm 11 (Line 14 – Line 23).

An example of this case is illustrated in Figure 4.9. After expanding L1, Sa_0 discovers a path to L4, $a(L1, L4)$. To route that path information to Sa_1, it follows the process indicated earlier in Case 1. When the routing request arrives at Na_{T1}, the state becomes that of Case 2. L4 is not directly under Terminal 1, but is directly under Zone 2, which is in turn under Terminal 1. Therefore, L4 is indirectly under Terminal 1 which is under the responsibility of Na_{T1}. To route this, Na_{T1} forwards the request to its child agent Na_{T1Z2}, which then forwards the request to its child

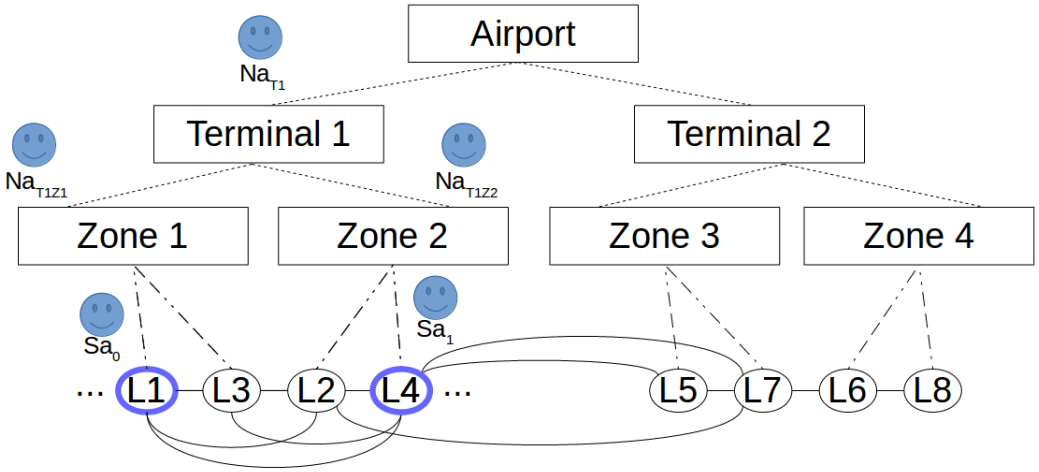

Figure 4.9: Routing path information – Case 2 example

search agent Sa_1.

Case 3: Directly under an organisational entity of the network agent In this case, node n' is under an organisational entity that the network agent Na manages. Amongst all the child search agents of Na, if there is a search agent Sa' responsible for n', then Na sends the path information $a(n, n')$ to Sa', and the routing process terminates. If n' has not been assigned to any search agent (i.e., Sa' does not exist), Na chooses a child search agent that has the *least responsibility* to take charge of n'. However, when all the child search agents of Na have reached the *responsibility limit*, a new child search agent is created to take charge of n' (see 4.4.3 for details on responsibility limit and search agent creation). The routing protocol for this case is shown in Algorithm 11 (Line 3 – Line 13).

Consider the example illustrated in Figure 4.10. Suppose that by expanding node L1, search agent Sa_0 discovers a path to L3, $a(L1, L3)$ and that Sa_0 is not in charge of L3. Therefore, Sa_0 sends $a(L1, L3)$ to its parent agent Na_{T1Z1}, which is in charge of Zone 1. Since L3 is directly under Zone 1, Na_{T1Z1} is able to assign it to one of its child search agents that has the least responsibility. Suppose that Sa_0 has not reached its responsibility limit, so $a(L1, L3)$ is assigned to Sa_0. Upon receiving the assignment, Sa_0 adds L3 to its $ResponsibledNodes$ and $a(L1, L3)$ to its $ArcList$ to compute its cost later on.

4.4.2 Separating a search space

One of the role of network agents is to divide the search space into different relevant parts such that search agents can explore them in parallel. The separation of a search space is done based on the organisational structure of the search space. Each network agent is responsible for a part of the search space which is composed of a set of organisational entities. This means that they manage the search process

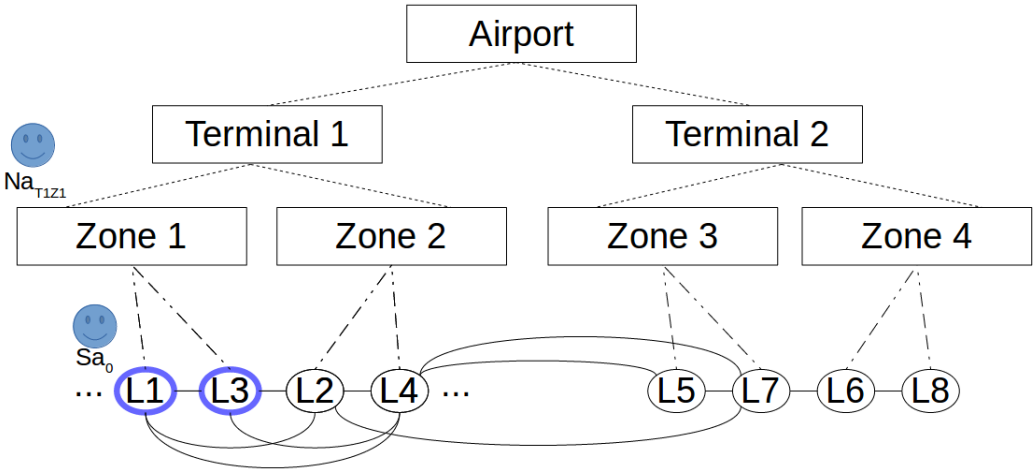

Figure 4.10: Routing path information – Case 3 example

that involves the nodes under the organisational entities of which they are in charge. There are 2 ways in which an organisational entity is assigned to a network agent:

- The network agent that creates a new network agent can assign an organisational entity to the newly created agent;
- The parent network agent can assign an organisational entity to its child network agent.

A parent network agent assigns an organisational entity to another network agent under its management that has the least responsibility. The responsibility of a network agent is measured by the sum of the number of nodes under the organisational entities for which it is responsible. We employ such an indicator because the number of nodes determine the number of potential tasks such as routing and other communications a network agent has to handle. Each network agent has a responsibility limit that is the maximum number of nodes it should handle. This limit is determined by the computational resources available. Setting the limit low results in having more network agents, but this would avoid problems such as agents overloaded with tasks to perform. Different limits are experimented, and the results are discussed in the Chapter 6.

4.4.3 Distributing workloads

A set of search agents are created to explore each part of the search space. Distributing workloads is a mechanism that balances the search work amongst search agents. It is one of the roles of network agents to distribute the workloads amongst their child search agents to explore the part of the search space of which they are in charge.

When a new node is discovered, a network agent assigns the node to one of its child search agents that has the least responsibility. To compute the responsibility

of a search agent, we take into account its current workload, which is the number of nodes in its *Frontier* and the number of nodes for which it is responsible *ResponsibledNodes*. The process of assigning a new node to a search agent is as follows:

1. Network agent sends a request for responsibility information to each of its child search agent;
2. Each child agent replies with the requested information;
3. Based on the received information, the network agent decides to which child search agent the node should be assigned.

The workload indicates the current tasks that a search agent has to execute, and the number of nodes in *ResponsibledNodes* indicates the amount of potential tasks that it may have to do. The potential tasks include requesting the cost of arcs or paths, processing update messages, and pruning. Using both criteria to measure responsibility enables us to assign more work to a search agent that is more likely to become idle (i.e., having few current tasks), and also preventing workload assignments to the search agents that might potentially be occupied (i.e., responsible for many nodes).

However, if all the child search agents have reached the responsibility limit, a new search agent is created to take on the new task. The reason for introducing responsibility limit is to distribute workloads among the search agents exploring the same part of the search space. This is essential when the part of the search space of a network agent is large. The responsibility limit is determined according to two factors: the computational resources available and the search space. If the computational resources are limited, the responsibility limit should be high to reduce the number of search agents. This configuration, however, may affect the efficiency when working with a large graph. Otherwise, the limit should be low, resulting in more search agents exploring in parallel.

4.5 Conclusion

In this chapter, we address the Research question 2 which is *how to address the latency of accessing resources and transferring data from the resources?* To this end, we presented our collaborative multi-agent search model that aims at mitigating latency resulting from resource accesses during a search process and at improving search efficiency. This model is based on multiple agents working collaboratively towards a shared goal. It can be applied to forward-search algorithms such as uniform-cost search and A* search.

In the proposed model, the latency issue is handled by asynchronising the process of accessing resources such that search agents are able to perform other tasks while the information is being retrieved. Furthermore, the organisation of and interaction amongst agents employed in the model enable search agents to explore different

parts of the search space in a parallel and distributed manner to reduce the search time, while still guaranteeing the optimality of the solution.

The collaborative search model described in this chapter can be used to solve pathfinding problems. However, it does not take into account the notion of goals. In the following chapter, we present our approach to address multi-goal pathfinding by formulating it as a multi-layered search problem, and we present how we can apply the collaborative multi-agent search model on each layer to solving multi-goal pathfinding as well as to address the latency issue.

Chapter 5

An Approach to Multi-goal Pathfinding in Ubiquitous Environments

Contents

Multi-goal pathfinding goes beyond the classical pathfinding problems to incorporate the constraints of goals to the problems. It consists of two interdependent problems, namely the pathfinding problem and the goal satisfaction problem. Furthermore, in ubiquitous environments, multi-goal pathfinding is attributed with the dynamics of the environments, which renders the problem more complex. In the previous chapters, we presented a semantic model for abstracting a ubiquitous environment integrating the aspects needed for solving multi-goal pathfinding. We also described a collaborative multi-agent search model that is capable of handling the latency resulting from accessing to resources during a search process, which is crucial when solving multi-goal pathfinding in ubiquitous environments. We can solve pathfinding problems in a ubiquitous environment by using the semantic model to abstract the environment as a search graph and employing the collaborative search

model to perform the search over the search graph. However, to solve multi-goal pathfinding, we need to address both the pathfinding problem and the goal satisfaction problem. In other words, it is necessary to determine not only the path from an origin to a destination, but also the locations along the path in which a set of given goals can be satisfied.

In this chapter, we present our approach to solving multi-goal pathfinding in ubiquitous environments. In this approach, we formulate a multi-goal pathfinding problem as a multi-layered search problem, the first layer representing the goal satisfaction problem and the second layer the pathfinding problem. In this way, search algorithms can be used to search on each layer to find the locations for satisfying the goals and the optimal path connecting the locations. To address the dynamics of the environments, we propose a mechanism that takes into account the changes in the environment and updates the solution accordingly.

The rest of the chapter is organised as follows. First, we provide an overview of the approach. Second, we present how the semantic model is used to abstract a ubiquitous environment in the approach. Third, we describe a method for extracting a pertinent subgraph from the search graph for a given multi-goal pathfinding problem to reduce the search efforts. Fourth, we present how we generate a search graph integrating the given goals with the locations in the environment. Fifth, we demonstrate how search algorithms are employed in our approach to perform the search over a multi-layered graph to find the solution to a multi-goal pathfinding problem. Sixth, we present a mechanism that can be used to handle the effects of the dynamics of the environment on the solution. Finally, we conclude the chapter.

5.1 Overview of the approach

This approach takes a multi-goal pathfinding problem as an input, and generates, as an output, the solution to the problem. The followings are the definitions that are used in the rest of the chapter:

Definition 6 ***Multi-goal pathfinding problem***
A multi-goal pathfinding problem is a tuple MGPF = (E_t, n_o, n_d, G, CR, f) where:

- *E_t is the description of the environment at time t as previously defined in Definition 4;*
- *$n_o \in L_t$ is a node representing the origin location where L_t is a set of nodes representing the location in E_t as defined in Definition 4;*
- *$n_d \in L_t$ is a node representing the destination location;*
- *G is an ordered list of goals to satisfy;*
- *CR is a set of criteria for path evaluation;*
- *f is a cost function used to evaluate the path.*

The input of this approach is a multi-goal pathfinding problem, defined in Definition 6, which is composed of 5 elements. The first component is the description of the environment, E_t, in which the multi-goal pathfinding problem to be solved is situated. This description provides the information regarding different aspects of the environment needed in the approach to solve the problem such as the spatial topology and the cyber-physical-social aspect of the environment. The second component is the path request which consists of an origin location n_o and a destination location n_d. The third element is an ordered list of goals G that need to be satisfied along the path from the origin location to the destination location. The criteria for path evaluation CR, which is the fourth component, are problem-specific. For instance, a criterion can be distance, monetary price, duration, or all of them combined. The last component, the cost function f determines how the criteria are considered when computing path cost. For example, given 3 different criteria – distance, price, and time, a given cost function may prioritise a subset of the criteria or compromise all of them equally.

Definition 7 ***Multi-goal pathfinding solution***
A multi-goal pathfinding problem is solved when an optimal solution is found. A solution is a path that connects the origin node to a list of nodes, through which all the goals can be satisfied in the given order, and to the destination node. A solution is optimal if it has a minimum cost evaluated using the cost function f.

The output of this approach is a solution to the given multi-goal pathfinding problem. A solution to a multi-goal pathfinding is composed of a path, the locations in which each goal can be satisfied, and the path cost. A path is an ordered list of locations that connects the origin location to the destination location. In the path, the locations to satisfy each goal are included and connected to other locations such that by following the path, we can reach the destination by also passing by the locations where the goals can be achieved. The cost of a path is the sum of the cost of moving between two locations in the path.

Workflow of the approach
This approach consists of 4 steps: (1) generation of the description of the environment, (2) subgraph extraction, (3) goal-space graph generation, and (4) multi-layered search. The first step is problem-independent in the sense that an environment is described independently of any specific multi-goal pathfinding problem. The description of an environment can be used to solve different multi-goal pathfinding problems as long as the problems are located in the environment. The other steps of the approach are problem-specific. The workflow of the approach is illustrated in Figure 5.1.

As in the classical approaches to pathfinding, the first step in our approach is to abstract the environment as a search graph. To this end, we employ our knowledge model (described in Chapter 3) to describe the environment integrating the aspects necessary for solving multi-goal pathfinding. Generating the description

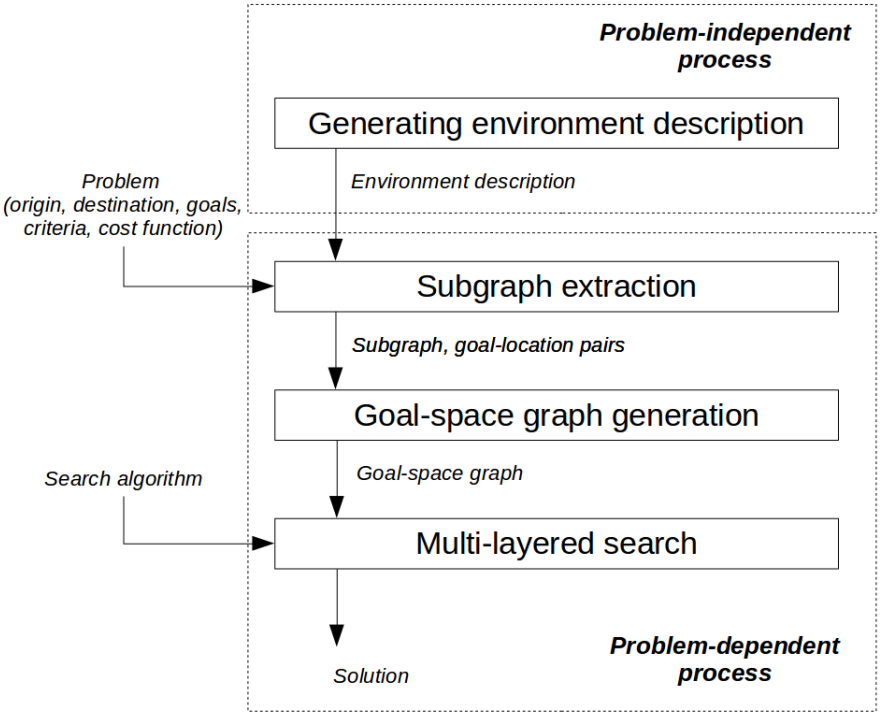

Figure 5.1: Workflow of the approach

of the environment is conducted prior to the problem-solving process. The output of this step is a description the environment in the form of a search graph.

The second step of the approach is to extract from the search space the information that is specific to a given multi-goal pathfinding problem. In this step, we identify the locations in which the given goals can be satisfied. In addition, we extract from the search graph the subgraph that contains the locations allowing all the goals to be satisfied.

The third step takes the information extracted during the second step as an input, and generates a graph that connects pairs of goal and locations following the order given in the problem. The generated graph is entitled a goal-space graph, which is the output of this step.

In the last step, we combine the goal-space graph and the search graph of the environment to construct a multi-layered graph. The first layer of the graph represents the locations in which the goals can be satisfied in the given order. The second layer is the complete search graph representing the environment. Then, we apply a search algorithm on the first layer to determine which location to satisfy each goal. On the second layer, we use a search algorithm to find the optimal path to connect the chosen locations from the first layer.

5.2 Generating the description of the environment

The objective of this step is to generate a description of a given environment. Such a description is an abstraction of the real environment that includes the aspects necessary for solving multi-goal pathfinding such as the spatial topology, the organisational structure, and the cybernetic, physical, and/or social entities of the environment. To provide such a description, we model the environment using our knowledge model, which was presented in Chapter 3. Then, we employ the proposed ontology, entitled Ubiquitous Environment Abstraction Ontology (described in Section 3.2), with extensions if necessary, to generate a description of the environment in RDF. An example of an environment description is provided in Section 3.3.2.

The knowledge model abstracts an environment as a graph where locations are nodes and the paths between connected locations are arcs. The knowledge required to solve multi-goal pathfinding is embedded in the description of the graph. Both the graph and the embedded knowledge are used in the other steps of the approach.

It is important to recall that ubiquitous environments are inherently dynamic. Some of the elements included in the environment description such as cybernetic, physical, and/or social entities and connections between locations are mobile or may change their state over time. In consequence, the environment description needs to be updated to take into account the changes in order to provide an accurate representation of the environment. To this end, we can use a mechanism that continuously detects the changes in the real environment. When a change is detected, the environment description is updated to reflect that change accordingly. The dynamics of the environment is addressed in Section 5.6.

5.3 Subgraph extraction

In this step, we take as an input a multi-goal pathfinding problem or request which is composed of an origin location, a destination, an ordered list of goals to satisfy, a set of criteria for path evaluation, and a cost function. The component of the problem that is used in this step is the ordered list of goals to satisfy. The objectives of this step are as follows:

- Identify the locations in the environment where each given goal can be satisfied,
- Extract from the graph of the environment a subgraph in which all the goals in the given list can be satisfied.

In order to identify the locations in which a goal can be satisfied, first, we need to be able to determine through which entity a goal can be satisfied.

5.3.1 Identifying supported activities

As defined in Definition 4, a goal g is satisfied by carrying out an activity $act \in Act_t$. In our approach, we assume that there exists a function that allows us to

determine the activity that satisfies a given goal. Such a fucntion can be based on an ontology describing the relations between goals and activities supported in a given environment. One of the challenges that we address in this step is how to identify the cybernetic, physical, and/or social entity through which an activity can be carried out to satisfy a goal. To this end, we propose an algorithm that is able to identify the activities that can be carried out through an entity.

To determine the activities an entity can support, we use the knowledge in the description of entity (part of the environment description), as demonstrated in Algorithm 12. This algorithm takes the URI of the entity as an input. In the case where the potential activities supported by the entity are explicitly provided in its description, we can use that knowledge directly to determine the types of activities (Algorithm 12: line 2-5).

In addition, we also make use of the ontology to identify the types of activities that the entity supports (Algorithm 12: line 6-12). The associations between subclasses of `ueao:CPSEntity` and subclasses of `ueao:PotentialActivity` are defined in the ontology through existential restrictions on the property `ueao:supportsActivity`. We exploit such connections to identify the classes of activities *cpse* supports.

Algorithm 12 getClassesOfSupportedActivities(*cpse*)

1: retrieve the description of *cpse*
2: *supportedActivities* ← get the objects of property *ueao*:*supportsActivity* where the subject is *cpse*
3: **for** each *activity* in *supportedActivities* **do**
4: add the class of *activity* to *classesOfSupportedActivities*
5: **end for**
6: *cpseClass* ← get the class of *cpse*
7: retrieve the description of *cpseClass*
8: **for** each *superClass* of *cpseClass* **do**
9: **if** *superClass* is an anonymous class that supports instances of a subclass of *ueao*:*PotentialActivity* **then**
10: add subclass of *ueao*:*PotentialActivity* to *classesOfSupportedActivities*
11: **end if**
12: **end for**
13: return *classesOfSupportedActivities*

For the purpose of demonstration, suppose that the following is the description of a physical entity KFC that is located in an airport.

```
<KFC>
      ueao:supportsActivity saa:EatFriedChicken ;
      ueao:supportsActivity saa:EatBurger ;
      ueao:supportsActivity saa:DrinkSoda ;
      ueao:supportsActivity saa:AccessInternet ;
      a saa:FastFoodRestaurant ;
      ueao:hasEntityResource <https://www.kfc.fr/> .
```

In the description, it is explicitly stated via the property `ueao:supportsActivity` that KFC supports 4 types of activities, namely eat fried chicken, eat a burger, drink soda, and access to the Internet. In addition, it is also stated that KFC is an instance of the class `saa:FastFoodRestaurant`. KFC supports any activity that is supported by `a saa:FastFoodRestaurant` because it is an instance of that class. Therefore, we can look up the description of the class `a saa:FastFoodRestaurant` to identify the activities supported by that class.

5.3.2 Extracting locations

The first objective of this step of the approach is to identify the locations in which each goal can be satisfied. To this end, we propose an algorithm for extracting the locations using the environment description as shown in Algorithm 13. The algorithm consists of 2 main steps. First, given a goal g, we identify the activity *act* required to satisfy the goal via a function we assume exist (Algorithm 13 Line 1). Second, we explore all the locations to determine if it can satisfy the goal (Algorithm 13 Line 4–16). For each location, we examine the cybernetic, physical, and/or social entities it contains. If one of these entity supports *act*, then location is added to the list of locations in which the given goal can be satisfied. To determine if an entity supports *act*, we make use of the algorithm entitled *getClassesOfSupportedActivities()* as previously defined in Algorithm 12. At the end of the algorithm, a set of locations in which the given goal can be satisfied is returned.

Algorithm 13 identify-location(goal)

```
1: activity ← get the activity required to satisfy goal
2: organisationEntities ← all the organisational entities of the environment
3: locations ← ∅
4: for each organisational entity oe in organisationEntities do
5:    access the description of oe
6:    for each location l under oe do
7:       access the description of l
8:       for each entity cpse located in l do
9:          activityClasses ← getClassesOfSupportedActivities(cpse)
10:         if activity is an instance of one of the classes in activityClasses then
11:            add l to locations
12:            break
13:         end if
14:      end for
15:   end for
16: end for
17: return locations
```

For example, suppose there is a traveller in an airport who has 4 goals to achieve in the airport, namely check-in, have lunch, use a lavatory, and withdraw some money. For each of these goals, we use Algorithm 13 to identify the locations where it can be satisfied. Using the algorithm, we are able to identify the locations for each goal as below:

Goal 1: Check-in

- The activity associated to goal: check-in activity
- The locations that can satisfy check-in activity: self check-in area in Terminal 1 and self check-in area in Terminal 2

Goal 2: Have lunch

- The activity associated to goal: eat activity
- The locations that can satisfy eat activity: KFC, The Loaf, Starbucks, and Dunkin Donuts

Goal 3: Use a lavatory

- The activity associated to goal: toilet activity
- The locations that can satisfy toilet activity: west toilet, east toilet, and KFC

Goal 4: Withdraw some money

- The activity associated to goal: withdraw money activity
- The locations that can satisfy withdraw money activity: ATM Maybank location and ATM CIMB location

Having identified the locations for each goal, we use that knowledge to make pairs of goals and their associated locations. In this example, the goal-location pairs are as follows:

- Check-in: (Check-in, self check-in area in Terminal 1) and (Check-in, self check-in area in Terminal 2)
- Have lunch: (Have lunch, KFC), (Have lunch, The Loaf), (Have lunch, Starbucks), and (Have lunch, Dunkin Donuts)
- Use a lavatory: (Use a lavatory, west toilet), (Use a lavatory, east toilet), (Use a lavatory, KFC)
- Withdraw some money: (Withdraw some money, ATM Maybank location) and (Withdraw some money, ATM CIMB location)

These goal-location pairs will be used in the next step of the approach to generate a goal-space graph.

5.3.3 Extracting a subgraph

The second objective of this step is to extract a subgraph that contains all the locations in which all the given goals can be satisfied. The subgraph is a set of organisational entities that cover all those locations. The algorithm to extract a subgraph is shown in Algorithm 14. This algorithm takes the locations in which all the given goals can be satisfied as an input. Then, it returns a set of organisational entities under which the input locations are situated.

Algorithm 14 extract-relevant-organisational-entities(locations)

```
1: organisationalEntities ← ∅
2: for each location l in locations do
3:     access the description of l
4:     oe ← get the organisational entity under which l is located
5:     add oe to organisationalEntities
6: end for
7: return organisationalEntities
```

In the context of the previous example of the traveller with 4 goals to satisfy, the subgraph extraction algorithm returns 2 organisational entities, namely Terminal 1 and Terminal 2 as the locations where the goals can be satisfied are situated in both Terminal 1 and Terminal 2. The subgraph will be used as knowledge for the heuristics of the search algorithm in the last step of the approach.

5.4 Goal-space graph generation

In this step, we know the locations where each goal can be satisfied as this knowledge is discovered in the previous step. However, it is possible that a goal can be satisfied in more than one locations. Therefore, the problem is to determine the optimal location to satisfy each goal. The objective of this step is to construct a graph, entitled goal-space graph, where nodes are goal-location pairs and arcs represent the connection between goals in an order given as a part of the multi-goal pathfinding problem. The goal-space graph represents the search space of the given multi-goal pathfinding problem as it incorporates both the pathfinding problem and the problem of selecting nodes amongst the goal-location pairs.

A goal-space graph, denoted by π, is an acyclic graph that can be viewed as a partial plan in which there may be multiple goal-location pairs for each goal. We adapt the definition of a partial plan in [Ghallab 2016], and define $\pi_t = (L_t, GV_t, GE_t)$ where:

- L_t, as previously defined in Definition 4, is a set of locations in a given environment at time t;
- GV_t is a set of nodes of an acyclic graph. Each node $v \in GV_t$ contains a goal $g \in G$ and a location $l \in L_t$ where g can be satisfied at l;

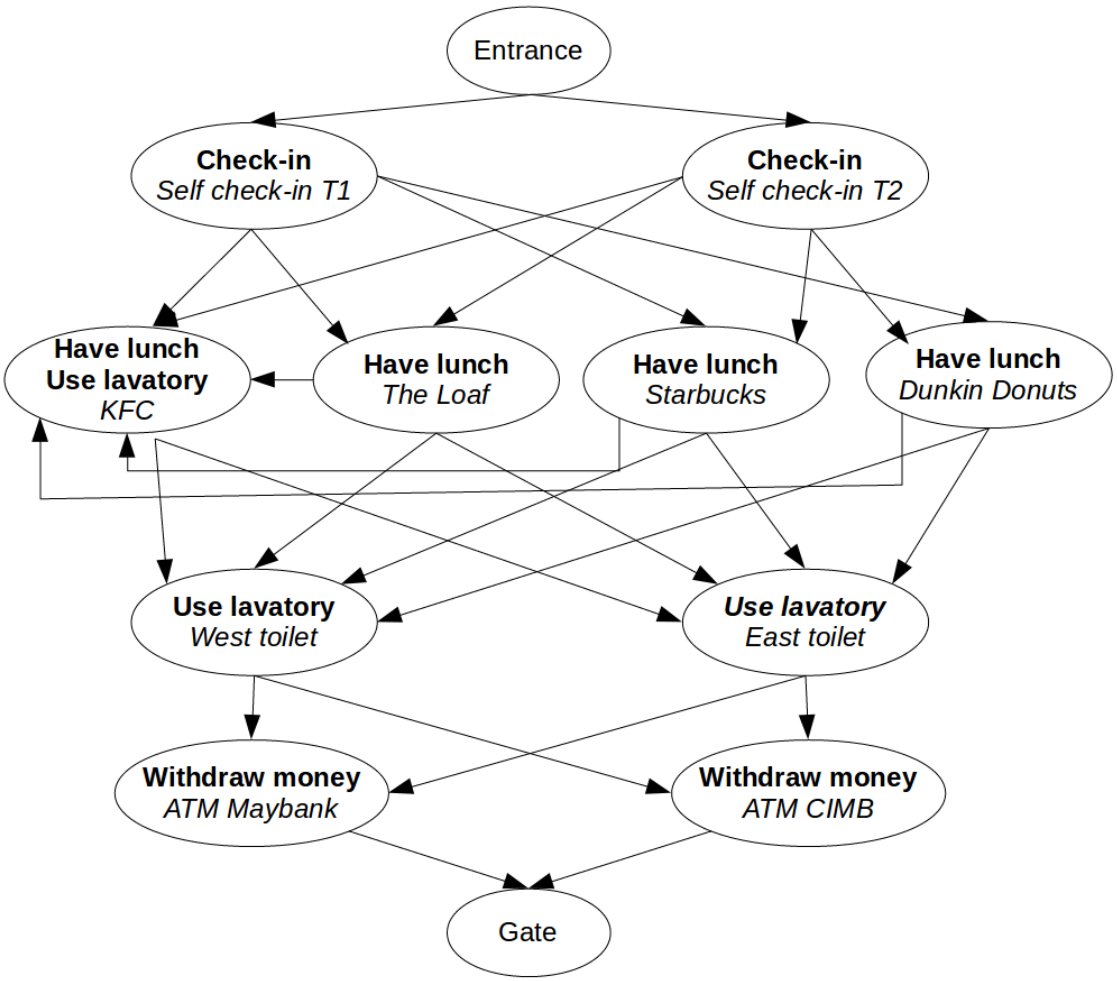

Figure 5.2: An example of a goal-space graph

- GE_t is the edges of an acyclic graph. They represent ordering constraints on the goals in GV_t. We define $v \prec v'$ if $v \neq v'$ and (GV_t, GE_t) contains a path from v to v'.

For an illustration purpose, refer back to the example regarding the goal-location pairs in Section 5.3.2. Suppose that the traveller wants to go from the main entrance of Terminal 1 to a gate in Terminal 1. On his way, he wants to achieve the 4 goals as previously mentioned. With those goal-location pairs, we can generate a goal-space graph as shown in Figure 5.2.

The output of this step is the goal-space graph. Having generated the graph, the next step is to choose the optimal location to satisfy each goal and to find the path that connects between the selected locations, which is presented in the following section.

5.5 Multi-layered search

Having generated a goal-space graph that represents both the *environment* and the *goals* specific to the given multi-goal pathfinding problem, in this step, the objective is the *search within the goal-space graph for the optimal path from the start node to the destination node.*

5.5.1 Searching on a multi-layered graph

In a goal-space graph, nodes integrate 2 aspects of the problem: the goals and the spatial dimension (locations). Unlike in the search graph of the environment, the locations of 2 connected nodes in a goal-space graph are not necessarily directly connected at the spatial level. Computing the cost of an edge between 2 nodes in a goal-space graph is equivalent to a problem of finding a path between 2 corresponding nodes on the search graph. Therefore, performing a search on a goal-space graph is actually searching on 2 different graphs, the goal-space graph itself and the search graph. We consider each graph as a layer, which makes it a multi-layered graph.

The generic steps required to perform a multi-layered search is shown in Algorithm 15. Various search algorithms can be used to perform the search. However, the modification required is the computation of the cost of nodes (Algorithm 15 Line 5). Depending on the algorithm, the cost function used to compute the cost of a node n is either $f(n) = g(n)$ for an uninformed algorithm or $f(n) = g(n) + h(n)$ for an informed algorithm where $g(n)$ is the real cost between the origin node n_{origin} and the current node n and $h(n)$ is the cost estimate between n and the destination node $n_{destination}$. Since connected nodes in a goal-space graph are not necessarily directly connected in the search graph, to compute $g(n)$, we need to perform a search in the search graph to find the path between n_{origin} and n and its cost.

Algorithm 15 Generic steps for a multi-layered search

1: $Frontier \leftarrow \{n_{origin}\}$
2: $Expanded \leftarrow \emptyset$
3: **while** goal not found or not the end of search space **do**
4: select a node $n \in Frontier$ to expand
5: *compute the cost of n*
6: remove n from $Frontier$ and add it to $Expanded$
7: goal test
8: pruning
9: **end while**

It is important to recall the issue of latency that results from resource accesses to the cost of nodes. The algorithm that is chosen to search on the search graph of the environment should be able to address this issue.

5.5.2 Heuristics

Heuristics play an important role in graph search. Good heuristics lead a search process to find a path quicker and to expand less nodes. In our context, node expansion can be costly. Expanding a node requires the algorithm to compute the cost function of every reachable neighbour. Considering it needing access to resources to retrieve information for the computation, the number of node expansions has a significant impact on the overall performance of the algorithm. For example, accessing resources might introduce latency or problems with threshold limits in accessing the API of resources. Therefore, we propose 3 heuristics that exploit the knowledge of the search space to improve the search process. A path cost is computed based on the given criteria such as distance, price, time, or all of them combined. Therefore, we need some sorts of scaling factors to express the real path cost and the estimated cost computed using the heuristics in the same unit. These heuristics can be used in search algorithms to search on a goal-space graph and a search graph. How these heuristics are used in search algorithms is described in the next section.

Heuristic on the goal-space graph
On the goal-space graph, denoted as π, we design a heuristic h_π for evaluating each node that is computed based on the number of goals that can be satisfied at the node's location and the quality of cybernetic, physical, and/or social entities (CPS entities) the location contains. To evaluate a CPS entity, we retrieve information from the resources associated with the entity, as provided in the description of the entity using the property `ueao:hasEntityResource`. The rationale behind using such information is that there may be multiple locations where each goal can be satisfied. Introducing qualitative information as a part of the cost function allows us to find a better location for each goal.

Heuristics on the search graph
On the search graph, denoted as CSG, we propose 2 heuristics, namely the goal-based heuristic and the organisation-based heuristic. Goal-based heuristic h_{goal} uses the knowledge extracted during the subgraph extraction step, described in Section 5.3. This heuristic prioritises nodes that are in the subgraph. Let CSG_{sub} be a subgraph, L_{sub} a finite set of locations in CSG_{sub} and OE_{sub} a finite set of organisational entities in CSG_{sub}. If a node n belongs to L_{sub} or a direct or indirect organisational entity of n belongs to OE_{sub}, we prioritise n by assigning its heuristic cost $h_{goal}(n) = 0$. Otherwise, $h_{goal}(n) = 1$. The logic behind this heuristic is that the subgraph contains the locations and organisational entities in which the given goals can be satisfied. Expanding those nodes may lead to more promising paths as they are the locations or under the organisational entities where at least one goal can be satisfied.

Organisation-based heuristic $h_{organisation}$ prioritises the expansion of nodes that are located under an organisational entity close to that of the destination. Let $n_{destination}$ be the destination node and n be the node being evaluated. If the organisational entity of n is different from that of $n_{destination}$, we add a value of 1 to the

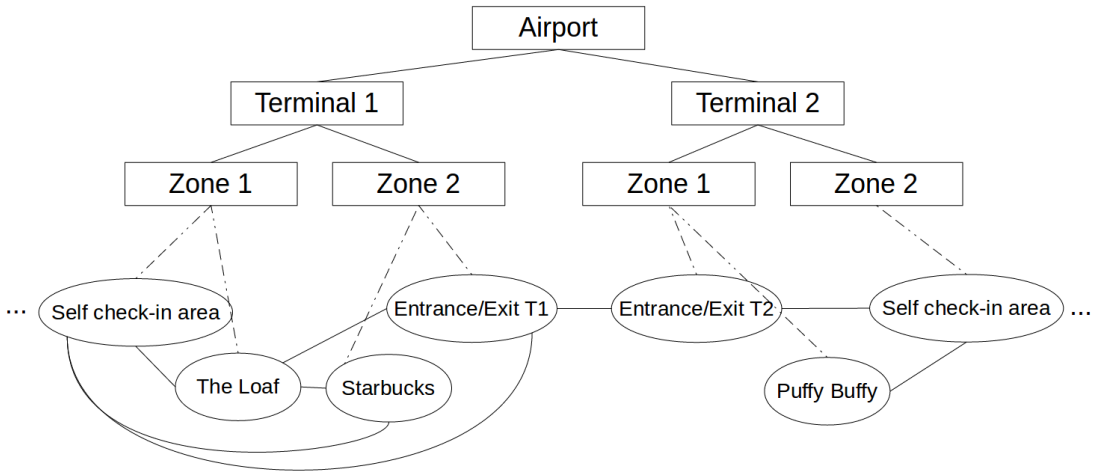

Figure 5.3: Hierarchy-based organisation example

heuristic cost of n. If the organisational entity has multiple level (e.g., a hierarchy), for each level of a hierarchy, if the organisational entity of n is different from that of $n_{destination}$, we increase the the heuristic cost by 1. For example, refer to Figure 5.3 and assume that Starbucks is the destination node. The $h_{organisation}$(Entrance/Exit T1) = 0 because Entrance/Exit T1 and Starbucks share the same organisational entity at every level. The $h_{organisation}$(The Loaf) = 1, The Loaf is under an organisational entity *Zone 1*, while Starbucks is under *Zone 2*. The $h_{organisation}$(Puffy Buffy) = 2 because there are 2 levels of organisational entities that Puffy Buffy and Starbucks do not share. The underlying motivation of this heuristic is that moving between two nodes in the same organisational entity is less costly than between nodes in different organisational entities.

5.5.3 Multi-layered A* search

To demonstrate how search algorithms can be employed in our approach to perform the multi-layered search, we present how we adapt the A* algorithm into a multi-layered A* (MLA*) to handle resource accesses and allow communication between a goal-space graph π layer and its underlying search graph CSG. We chose A* for this demonstration because it allows us to use the knowledge-based heuristics we propose for each layer of a multi-layered graph. The aim of A* is to find a minimum-cost path. It evaluates nodes by their cost using a cost function $f(n) = g(n) + h(n)$.

Searching in the goal-space graph

In MLA* (Algorithm 16 and Algorithm 17), we combine 2 different versions of A*, an A^*_π for searching on the goal-space graph layer and an A^*_{CSG} for the search graph layer. A^*_π (Algorithm 16) behaves similarly to the classic A* except during node expansions. When a node n is expanded, a set of nodes reachable from n is found. However, the location of n and the location of each of its neighbours $l_{n'}$ might not

be directly connected in the search graph. To compute the path between n and each of its neighbours, A^*_π instantiates an A^*_{CSG} to search in CSG for a path between l_n and $l_{n'}$. To evaluate each node, A^*_π uses a cost function $f_\pi = g_\pi + h_\pi$. The cost of movement g_π on π layer is obtained from executing an A^*_{CSG} search.

Searching in the search graph

In the search graph, A^*_{CSG} (Algorithm 17) is based on the classic A* with a modification to communicate cached results of pathfinding, namely the path and its costs, to A^*_π so that those results can be reused. It is necessary to note that on the CSG layer, the search space is composed of locations, while organisational entities are used for computing the heuristics. To evaluate a node, A^*_{CSG} uses a cost function $f_{CSG} = g_{CSG} + h_{CSG}$. To compute g_{CSG}, we use information from the resources provided in the description of the environment. As for the heuristic cost h_{CSG}, we combine two heuristics, namely h_{goal} and $h_{organisation}$.

Algorithm 16 $A^*_\pi(\pi, n_{start}, n_{dest}, CSG)$

```
1: openList ← [n_start]; closedList ← []; cachedPathCost ← []
2: while openList not empty do
3:    n ← pop-min(openList)
4:    if n is the destination n_dest then
5:       return n
6:    else
7:       generate n's successors and set their parents to n
8:       for all n' successor of n do
9:          g ← cachedPathCost(l_n → l_n')
10:         if g is null then
11:            path_n-n' ← A*_CSG(CSG, l_n, l_n', cachedPathCost)
12:            g ← path_n-n'.cost
13:         end if
14:         n'.g ← g + n.g
15:         n'.h ← h_π(n')
16:         n'.f = n'.g + n'.h
17:         if n' not in openList nor closedList then
18:            add n' to openList
19:         else if n' in openList and f(n') < f(n'_openlist) then
20:            replace n'_openlist by n'
21:         else if n' in closedList and f(n') < f(n'_closedlist) then
22:            remove n'_closedlist from closedList
23:            add n' to openList
24:         end if
25:      end for
26:   end if
27: end while
```

Algorithm 17 $A^*_{CSG}(CSG, l_{start}, l_{dest}, cachedPathCost)$

```
1: openList ← [l_start]; closedList ← []
2: while openList not empty do
3:    l ← pop-min(openList)
4:    if l is the destination l_dest then
5:       return l
6:    else
7:       generate l's successors and set their parents to l
8:       for all l' successor of l do
9:          g ← request-resource r^{l-l'}
10:         add [(l → l'), g] to cachedPathCost
11:         h ← l'.h_goal + l'.h_hierarchy
12:         l'.f = l.g + g + h
13:         if l' not in openList nor closedList then
14:            add l' to openList
15:         else if l' in openList and f(l') < f(l'_openlist) then
16:            replace l'_openlist by l'
17:         else if l' in closedList and f(l') < f(l'_closedlist) then
18:            remove l'_closedlist from closedList
19:            add l' to openList
20:         end if
21:      end for
22:   end if
23: end while
```

Discussion
The output of this step is a minimum-cost path from the origin location to the destination that passes through the locations allowing the given goals to be satisfied in the given order. The path is the solution to the given multi-goal pathfinding problem. However, in this approach, until now, we have not taken into account the dynamics of the environment yet. During the search process, it is likely that the state of the environment evolves, which may influence the solution for which we are searching. In addition, after a solution has been found and during the time in which the traveller is following the proposed path, changes that affect the solution may also occur. These changes may invalidate the current solution. Therefore, both during the search process and the execution of the path, relevant changes in the environment need to be considered to adapt the solution accordingly. To this end, we propose a mechanism for handling such dynamics that is integrated into our approach, which is presented in the following section.

5.6 Handling the dynamics of the environment

To guarantee the validity of the solution, dynamic changes in the environment need to be handled during the search process and path execution. Changes in the environment result in changes in the cost of the arcs in the search graph, and thus the goal-space graph. To address the arc cost changes in the search graph, we extend our collaborative search model, described in Chapter 4, to incorporate the necessary steps to take into account the changes. For the search in a goal-space graph, we propose an update mechanism to handle the changes.

5.6.1 Dynamic search in the search graph

To handle the dynamics in the search graph, we extend our collaborative search model to handle the arc cost changes. The principle steps are as follows:

- Perform the first search using the collaborative search algorithm to find an optimal path;
- Make use of the multi-agent infrastructure constructed during the first search to communicate the changes and update the path accordingly, during path execution.

The first search
In the collaborative search model, when a goal is found, a goal verification procedure is initiated. If the procedure succeeds, then the path to the found goal is verified as the optimal path. During the search process, a multi-agent infrastructure that is composed of search agents, network agents, and resource agents is constructed. An example of the multi-agent infrastructure is depicted in Figure 5.4.

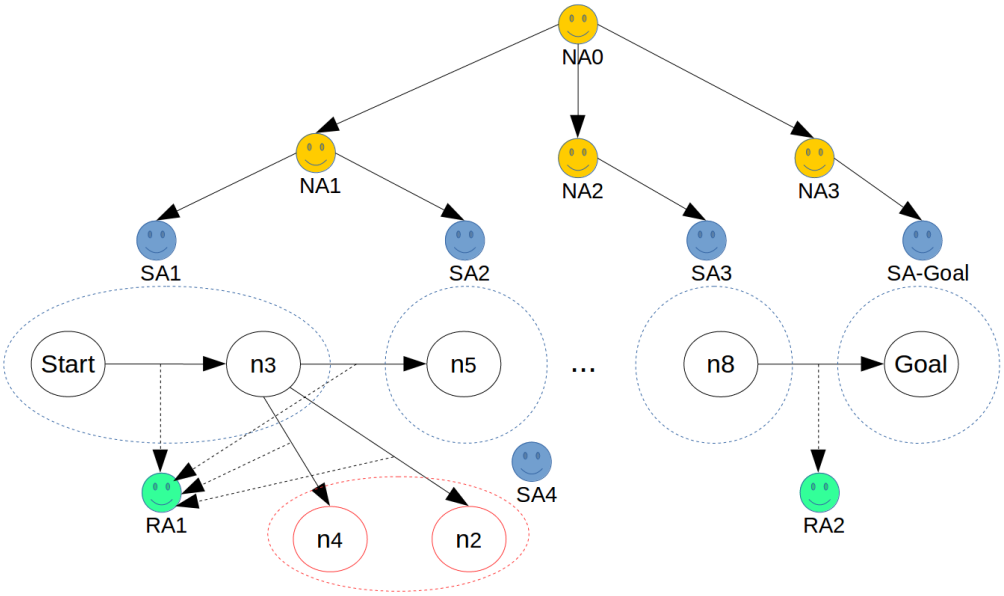

Figure 5.4: Example of a path and a multi-agent infrastructure constructed during the search process

Backtracking – setup for the incremental search

Having found the optimal goal during the first search, the next step is to backtrack from the goal node to build the path. We exploit the backtracking process to extend the multi-agent infrastructure to setup for the incremental searches that may follow when the environment changes. The search agent that found the goal node, named goal agent, is the agent that performs the backtracking. The steps required to setup for the incremental searches are as follows:

Step 1: Take charge of all nodes in the path

During the backtracking process, the goal agent takes charge of all the nodes in the optimal path as the agent will be monitoring and managing changes in the path. Therefore, for each node in the path, the goal agent takes the information about the node from the search agent responsible for the node. To complete this transition of responsibility, the goal agent informs the network agents associated with the nodes in the path about the changes so that the network agents can route the information about those nodes to the goal agent instead of the search agents formerly responsible for those nodes, during the search in the future.

For a demonstration, refer to Figure 5.4. The goal agent SA-Goal executes the backtracking process. It takes charge of all the nodes in the path such as goal node, n8, n5, n3, and start node. In addition, SA-Goal informs the network agents NA1 and NA2 to route any information regarding start node, n3, n5, and n8 to itself and not to SA1, SA2, and SA3.

Step 2: Subscribe for changes in the path

To be informed about the changes in the current path, the goal agent sends a request to the resource agents associated with the path to notify it when there are changes. Let us refer to Figure 5.4. The search agent SA-Goal sends a request to the resource agent RA2 to keep accessing the resources associated with the arc connecting n8 to Goal to detect changes and to notify it when changes occur. SA-Goal sends the same type of request to RA1 for the arc between Start and n3 and the arc between n3 and n5.

Step 3: Subscribe for changes in the successor nodes

In addition to the changes in the current path, changes in other parts of the search space may also affect the current path. Changes in the arcs leading to the successor nodes of the nodes in the current path also have to be taken into account as they may lead to a new optimal path. Let us refer again to Figure 5.4. The nodes n4 and n2 are not part of the current path, but they are the successors of the node n3 that is in the current path. During the first search, n5 was selected because the path to Goal through n5 is less costly than that via n4 and n2. However, the change of the arc cost between n3 and n4 or n3 and n2 may make the path through n4 or n2 less costly than via n5. Therefore, we need to monitor the changes to these successor nodes. To detect changes in the successor nodes, the goal agent requests the search agents responsible for the successor nodes to notify it when changes occur. The search agents responsible in turn requests the resource agents to notify them when there are changes in the arcs to successor nodes. In the infrastructure shown in Figure 5.4, SA-Goal requests for change notifications of n4 and n2 from SA4. SA4 subscribes to RA1 for change notification of the arc between n3 and n4 and n3 and n2.

Node selection is based on the cost evaluated by the cost function $f(n) = g(n) + h(n)$ where $g(n)$ is the path cost from the start node to the node being evaluated n and $h(n)$ is the estimated cost to go from n to the goal node. The heuristic function h evaluates a path cost based on some knowledge of the environment, and is not affected by the dynamic changes. Therefore, only the changes in the cost evaluated by g is affected by the changes. Provided that h is consistent, which means that it never overestimates the path cost, then real path cost is bigger or equal to the cost estimated by h. Therefore, changes in other parts of the search space, other than the current path and the successor nodes, are not taken into account. The reason is that when choosing between 2 successors n_1 and n_2 of a node n_0 where n_0 and n_1 are in the current optimal path, if $f(n_2) > f(n_1)$, then we can assert that the cost of any path through n_2 is higher than the cost of the current optimal path through n_1. Therefore, only the changes to $g(n2)$ can invalidate the current optimal path.

Handling changes
After the backtracking, the goal agent waits for notifications from either the resource agents about the changes in the path or from the search agents of the successor nodes.

Case 1: Handling changes within the current path
When the goal agent receives a notification from a resource agent about the changes in an arc cost within the current path, it recomputes to cost of the affected node and decides whether to keep the current path or explore other alternatives. Let n_1 be the affected node and a successor node of n_0 where both n_0 and n_1 are in the current path. Let n_2 be a successor node of n_0 where n_2 is not in the current path. The goal agent decides to keep the current path through n_1 if $f(n_1) <= f(n_2)$. Otherwise, the goal agent explores the path via n_2 by expanding n_2 and continues the search until the goal node is found. Then, it performs the backtracking process again with the new path. During the backtracking, nodes that are not in path anymore are assigned to their former search agents responsible. The goal agent also unsubscribes from the network agents and resources agents regarding those nodes.

Case 2: Handling changes outside the current path
When there are changes that affect a successor node, the search agent responsible recomputes the cost of the affected node and notifies the goal agent. Let n_0 be a node in the current path, n_1 a successor of n_0 and also in the current node, and n_2 a successor of n_0 that is affected by the change. If $f(n_2) >= f(n_1)$, the goal agent keeps the same path. If $f(n_2) < f(n_1)$, the goal agent explores the path via n_2 by expanding n_2 and continues the search until it finds the goal node.

Discussion
While the goal agent explores other alternative paths, more new updates may occur. Updates during the search process are not considered immediately. The goal agent waits until it finishes the current search process, and then it takes into account all the new updates at once. In addition, as the path changes, more and more arc costs are known. Therefore, the subsequent search processes may take lesser and lesser time.

5.6.2 Dynamic search in the goal-space graph

As mentioned in Section 5.5, nodes in a goal-space graph are not necessarily directly connected at the spatial level. To compute the cost of an arc between 2 nodes, we need to execute a search in the search graph to find the path between those 2 nodes. Therefore, changes that affect the search graph may also affect the path on the goal-space graph. Therefore, it is also necessary to handle the dynamic changes in the goal-space graph that result from the changes in the search graph. The main steps required to manage the changes are as follows:

- Perform the first search using a search algorithm to find an initial path;

- Extend the multi-agent infrastructures constructed to search for the paths between the goal-space graph nodes in the search graph.

The first search

After the goal-space graph is constructed, we can use a search algorithm to find the path between the start node and the goal node. During the search process, to compute the cost of each arc between 2 nodes, we perform a search using our collaborative search model adapted to the dynamics of the graph as presented earlier in Section 5.6.1. Therefore, after the solution is found we have a set of multi-agent infrastructures, each of which is responsible for finding the path between 2 nodes of the goal-space graph. These infrastructures will be used to communicate the changes from the search graph to the goal-space graph search process.

The multi-agent infrastructures were created to search for different paths. However, they search in the same search space. To avoid requesting resource agents for the same information, which creates unnecessary latency, resource agents store the information requested about the search space so that it can be reused when requested by agents from multiple infrastructures.

Backtracking – setup for the incremental search

To manage the changes in a goal-space graph, we extend the multi-agent infrastructures constructed during the first search. This is done during the backtracking process. When the search algorithm found the goal, it starts the backtracking process to build the path. During the backtracking, we create a search agent Sa_{gsg} to manage the changes in the goal-space graph by interacting with other agents in the infrastructures. Sa_{gsg} sends a request to each goal agent of the infrastructures for each arc to notify it when the path corresponding to each arc changes. Only the infrastructures for the arcs in the current path and the arcs connecting to the successors of the nodes in the current path are kept after the first search. Other irrelevant infrastructures are terminated.

Handling changes

When a path that corresponds to an arc in the goal-space graph changes, the goal agent of the corresponding infrastructures informs the search agent of the goal-space graph Sa_{gsg}. If the affected arc leads to a node in the path, Sa_{gsg} recomputes the cost of the affected node n. If the cost is smaller, then the current path remains. Otherwise, Sa_{gsg} compares the cost of n, denoted as $f(n)$ with the cost of its neighbours. If there is a node n_1 that is a neighbour node of n where $f(n_1) < f(n)$, Sa_{gsg} expands n_1 and explores the alternative path via n_1.

If the affected arc is an arc to a node that is a successor of a node in the current path, Sa_{gsg} recomputes the cost of the successor and decides whether to keep the current path or to expand the successor. The decision is made based on the cost of the successor. Let n_0 be a node in the goal-space graph and in the current path, n_1 be a successor of n_0 that is also in the current path, and n_2 be a successor of n_0 but not in the current path and whose cost, denoted as $f(n_2)$, has been

updated. If $f(n_2) >= f(n_1)$, Sa_{gsg} updates its cost and keeps the current path. If $f(n_2) < f(n_1)$, Sa_{gsg} expands n_2 and explores the path via n_2 by using the same search algorithm used in the first search, but starting from n_2, to check if there is a better path via n_2.

If a better path is found, Sa_{gsg} performs the backtracking process again. During the backtracking, Sa_{gsg} terminates the infrastructures corresponding to the arcs that are no longer in the path, nor leading to the successors of the nodes in path. It is important to recall that the node cost computation produces latency. Therefore, when the traveller is following the proposed path and has reached a location to satisfy a goal, we terminate, if any, the infrastructures corresponding to the arcs leading to the locations for achieving that goal.

5.7 Conclusion

In this chapter, we address the Research question 3 which is *how to solve multi-goal pathfinding?* and the Research question 4 which is *how to address the dynamic state of the environment during path computation and path execution?* We have presented our approach to multi-goal pathfinding. The way in which our knowledge model is used to describe a ubiquitous environment was provided. We describe the algorithms to extract the knowledge specific to a given multi-goal pathfinding problem from the description of the environment. We presented a method to generate a goal-space graph based on a given set of goals and a search graph. A goal-space graph is an abstract graph built on top of a search graph, making it a multi-layered graph. An adaptation of A* algorithm was made to demonstrate the search over a multi-layered graph. We proposed different heuristics exploiting the knowledge of the environment to be used in search algorithms. A multi-agent based mechanism was proposed to handle the dynamics of the environment to provide valid and update-to-date solutions.

Part III

EVALUATION

Chapter 6

Experimentation and Validation

Contents

To validate our proposal for solving multi-goal pathfinding problems in ubiquitous environments, we conducted various experiments to evaluate different components of our approach. More precisely, we validated our semantic model for describing ubiquitous environments as well as the proposed heuristics that are based on the knowledge described using the model. The collaborative multi-agent search model was evaluated based on different criteria.

The objective of this chapter is to present the experiments that were conducted for the evaluation of our approach. The rest of the chapter is organised as follows. First, we present the experiments for validating the knowledge model and the heuristics. Second, the evaluations of the collaborative multi-agent search model are provided. For each section, the configurations of the experiments, the empirical results, and the discussion based on the results are provided. Finally, we conclude the chapter with an overall discussion of the results.

6.1 Validation of the knowledge model and the heuristics

In this section, we present the experiments that we conducted to validate the semantic model and the heuristics. The objective of this evaluation is to demonstrate the validity of the knowledge model in describing ubiquitous environments, especially for solving multi-goal pathfinding, and of the heuristics that exploit the knowledge described using the model. This evaluation is based on the experiments to show the improvements that we can benefit from the heuristics.

6.1.1 Experiment configurations

The experiments were executed on a 2.4GHz Intel Core i7 machine with 16GB of RAM. For these experiments, we use the semantic model to describe 3 different environments, namely Environment 1, Environment 2, and Environment 3, whose properties are described in Table 6.1. The 3 environments are of the same size – containing 10000 locations. For Environment 1, the organisational structure is a 3-level depth hierarchy that is composed of 1 organisational entity at the first level, 10 at the second, and 100 at the third. For Environment 2 and Environment 3, the organisational structure is a 4-level depth hierarchy consisting of 1 organisational entity at the first level, 10 at the second, 100 at the third, and 1000 at the fourth. The differences between Environment 2 and Environment 3 is that in Environment 2, one location is connected to only one other location, and one leaf organisational entity (i.e., the organisational entity at the lowest level directly connected to the locations) has only one exit point to another leaf organisational entity via a location under its coverage.

For each experiment, we used 2 types of problems. The problems of type (1) consist in finding the path between a start and a destination locations that are under the same leaf organisational entity, while type (2) problems find the path between 2 locations of different leaf organisational entities. For each problem, we executed the search algorithm twice, and we took the average of the results. In our context, node expansion has a significant impact on the time performance of the algorithm as expanding each node requires an access to a resource for the information necessary for determining the cost of the path to the node. Therefore, we also introduced latency in accessing resources during node expansion. The way in which we simulated the latency is as follows. For the latency of n milliseconds, we generated randomly a value between 0 and n for each access to a resource.

6.1.2 Experiment 1: Comparing different heuristics

The objective of this experiment is to compare the efficiency that can be obtained from the proposed heuristics, namely h_{goal} and $h_{organisation}$, as described in Section 5.5.2. To this end, we used the classical A* algorithm to solve problems in Environment 3 using 3 different heuristics, h_{goal}, $h_{organisation}$, and h which is a combination of h_{goal} and $h_{organisation}$. Uniform-cost algorithm, which is independent of heuristics, was used as the baseline algorithm for the comparison.

Environment 1	
Environment properties	**Description**
Organisational structure	3-level depth hierarchy
Locations	10000
CPS entities	10000
Accessible locations from a location	1
Exit points from an organisational entity	1
Environment 2	
Environment properties	**Description**
Organisational structure	4-level depth hierarchy
Locations	10000
CPS entities	10000
Accessible locations from a location	1
Exit points from an organisational entity	1
Environment 3	
Environment properties	**Description**
Organisational structure	4-level depth hierarchy
Locations	10000
CPS entities	10000
Accessible locations from a location	3
Exit points from an organisational entity	2

Table 6.1: Environment 1–3

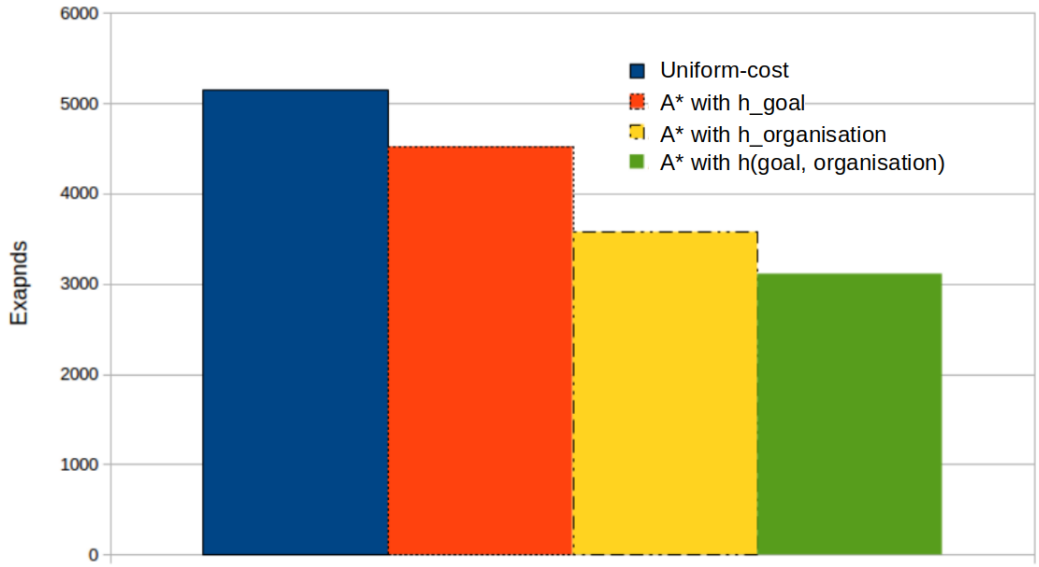

Figure 6.1: Comparison of expands by A* with heuristics and by uniform-cost

Figure 6.1 illustrates the number of nodes expanded by A* using different heuristics and by uniform-cost. A* with h expands the least reducing almost 40% of the expanded nodes compared to uniform-cost, followed by $h_{organisation}$ and then h_{goal}. The proposed heuristics direct the search process towards more promising nodes including the nodes that are in the extracted subgraph and the nodes that are close to the destination node. This enables the algorithm to explore less irrelevant parts of the search space.

To see how the heuristics influence the time performance of the algorithm, we introduced the latency of 1, 3, and 5 milliseconds (ms) during node expansion. The results are shown in Figure 6.2. With the heuristics, the algorithm expands less nodes, and thus finds the path faster. However, without latency, the time difference is minimal. This is due the fact that when using the heuristics, the algorithm is affected by the time required to compute the heuristics. The time efficiency gained by using the heuristics rises significantly as we increase the latency.

6.1.3 Experiment 2: Evaluating the heuristics based on problem types

The aim of the second experiment is to evaluate the heuristics in solving different types of problems. In this experiment, we used A* algorithm with a heuristic h, which is a combination of $h_{organisation}$ and h_{goal}, to solve the 2 types of problem, as mentioned earlier in the experiment configurations, in Environment 1. We compared the results of A* with the baseline algorithm uniform-cost.

Figure 6.3 depicts the time efficiency (in percentage) gained by using A* with h compared to uniform-cost. For problems of type (1), uniform-cost is more efficient than A*, with or without latency. As shown in Figure 6.3, the efficiency remains

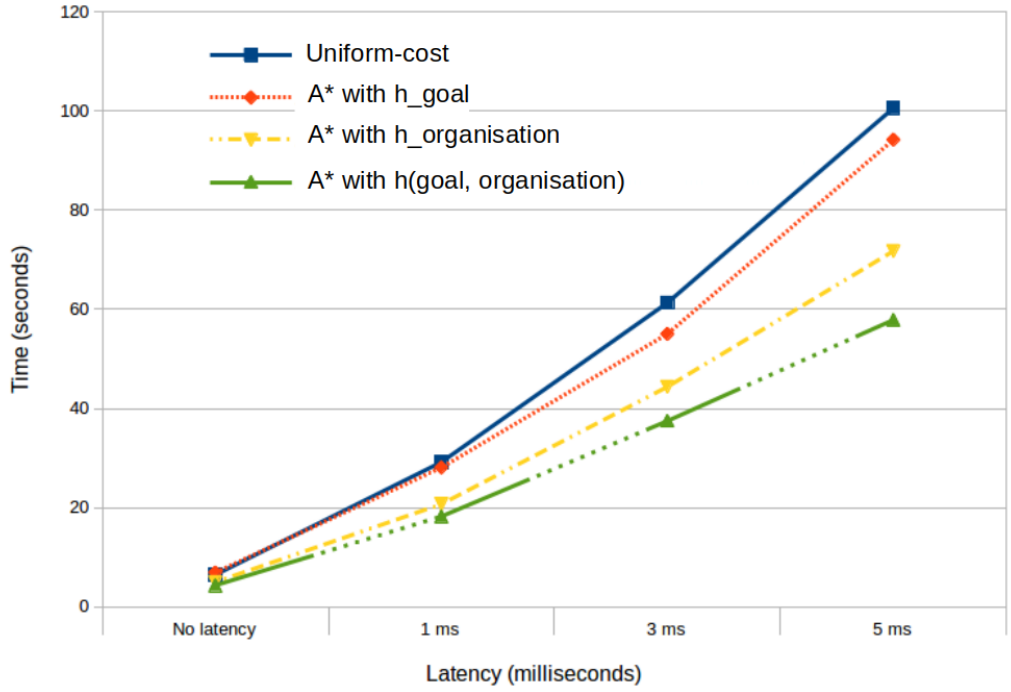

Figure 6.2: Runtime of A* using different heuristics and of uniform-cost

negative over various latency values. The main reason for such results is that since both the origin and the destination nodes are under the same organisational entity, the heuristic $h_{organisation}$ has no effect. In addition, A* needs more time to compute the heuristics.

For type (2) problems, the origin and the destination nodes are under different organisational entities, so the heuristics enable to algorithm to expand less nodes. However, the time efficiency gained from the reduced expanded nodes cannot compensate the time required to compute the heuristics. This results in uniform-cost performing better than A* when there is no latency. However, A* starts to outperform uniform-cost when approximately 2 ms of latency is introduced.

6.1.4 Experiment 3: Evaluating the heuristics over different environment structures

The objective of the third experiment is to determine how the heuristics contribute to the improvement of search efficiency in different types of environments. In this experiment, we compared A* algorithm with the heuristic h, a combination of $h_{organisation}$ and h_{goal}, to uniform-cost over 3 environments, namely Environment 1, Environment 2, and Environment 3. The time efficiency obtained by using the heuristics is illustrated in Figure 6.4.

The heuristics are most efficient in Environment 3 followed by Environment 2 and Environment 1. This is owing to the fact that, in Environment 3, there are

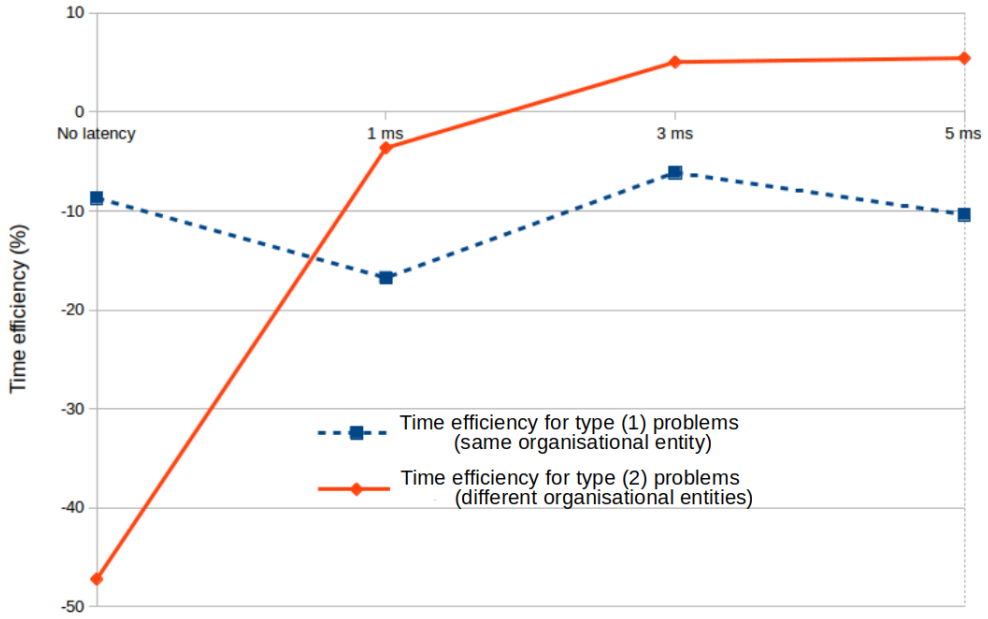

Figure 6.3: Runtime efficiency of A* with $h_(goal, organisation)$ compared to uniform-cost on different problem types

more connections between nodes and more exits between organisational entities. With more connections and exits, the heuristics guide the search process to choose better nodes among the available options. It is noteworthy that, using the heuristics in Environment 3 reduces a substantial number of expanded nodes, sufficient to compensate for the time required to compute the heuristics and to provide an efficiency roughly 35% over uniform-cost even without latency. In Environment 1 and 2, however, there are not many options. From each node, only one path is available. Therefore, both heuristics $h_{organisation}$ and h_{goal} are less effective in comparison.

Environment 1 and 2 are the same in terms of connectivity. The difference is that nodes are more distributed among the organisational entities in Environment 2. In Environment 1, each 1000 nodes are grouped under a single organisational entity, so the heuristic $h_{organisation}$ has a minimal effect. As a result, A* performs better in Environment 2 than in Environment 1. Uniform-cost outperforms A* when there is no latency for both Environment 1 and Environment 2 because the number of reduced expanded nodes cannot compensate for the time A* needs to compute the heuristics. However, when latency is introduced, the time needed to expand each node increases, making A* more efficient than uniform-cost.

6.1.5 Discussion

In most cases, the heuristics help the algorithm reduce irrelevant node expansion. They are more efficient when solving complex problems in well-connected and distributed environments. However, the time required to compute the heuristic func-

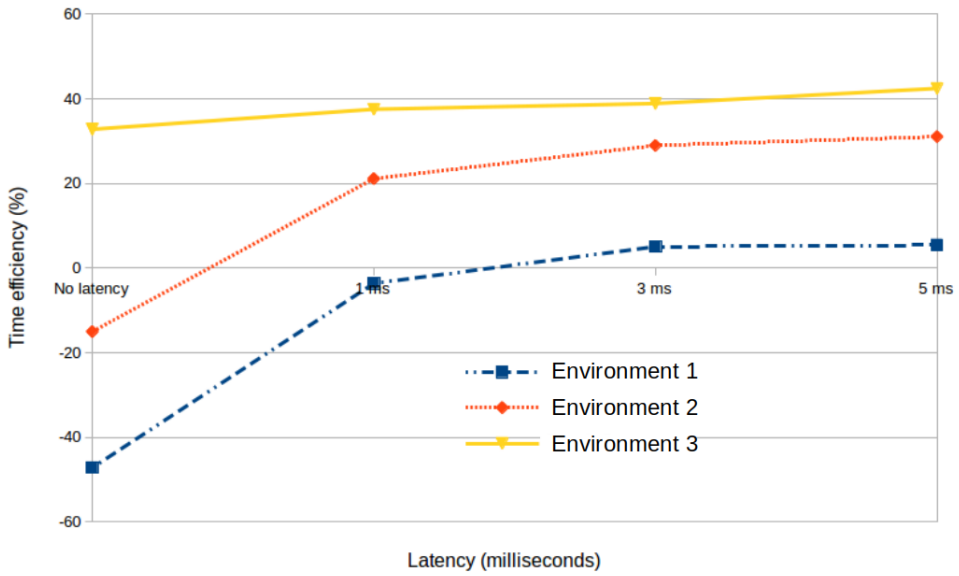

Figure 6.4: Run time efficiency of A* with $h_(goal, hierarchy)$ compared to uniform-cost over different environment structures

tion is not negligible. According to the experiments, uniform-cost performs better in most cases when there is no latency as the time obtained from reduced node expansion cannot compensate the heuristic computation time. A* starts to surpass uniform-cost when latency, around 1 ms, is introduced. The time efficiency grows in function of the latency, as shown in Figure 6.4.

6.2 Evaluation of the collaborative multi-agent search model

In this section, we provide an evaluation of the collaborative multi-agent search model. The objectives of the proposed search model are to handle latency and to improve search efficiency. To evaluate the efficiency of the search model in addressing these 2 aspects, we conducted the experiments to examine the performance of the search model in different problem types, environment structures, and problem scales.

6.2.1 Experiment configurations

The experiments were conducted on a 1.8GHz Intel Core i7 machine with 8GB of RAM. We applied our collaborative search model in uniform-cost algorithm, creating a collaborative uniform-cost algorithm. In our experiments, we compare the collaborative uniform-cost with the uniform-cost. The choice of uniform-cost for our experiments was motivated by the fact that uniform-cost is independent of any domain-specific or case-based heuristics. Consequently, the impacts of the collaborative search model on the performance of uniform-cost can be accurately observed.

Moreover, in each experiment, we used the search model to solve 2 types of problems as in the previous evaluation in Section 6.1. We also simulated the latency in accessing resources between 0 and 5 ms. For these experiments, we use 7 different environments as described in Table 6.2

6.2.2 Experiment 1: Evaluating the search model based on problem types

In the first experiment, we used the collaborative uniform-cost search and uniform-cost search to solve both types of problems in Environment 4. Time efficiency in (%) gained by using collaborative uniform-cost compared to uniform-cost for each type of problem is illustrated in Figure 6.5.

The results show that for type (1) problems whose start and goal nodes are under the same organisational entity, uniform-cost is more efficient when there is no latency. For this type of problems, both algorithms found the solution quickly. However, collaborative uniform-cost takes more time because of the overheads resulting from the following actions: access to the descriptions of organisational entities to retrieve the knowledge necessary in the algorithm, creation and management of agents, communications among agents, and goal verification procedure. However, these overheads become negligible when latency is present. As shown in Figure 6.5, collaborative uniform-cost outperforms uniform-cost starting from approximately 1 ms of latency, and the time efficiency continues to rise upto around 65% at 5 ms latency.

For the type (2) problems in which the start and the goal nodes are located under different organisational entities, collaborative uniform-cost is also less efficient without latency due to the overheads. However, collaborative uniform-cost is more efficient in solving type (2) problems than type (1). The reason is that the proposed search model is based on collaborative agents exploring different parts of the search space in parallel, which enables the algorithm to discover the goal node faster. With latency, collaborative uniform-cost is remarkably more efficient than uniform-cost, reaching over 90% when 5 ms latency is introduced.

Apart from the runtime, we also evaluated the search model using the number of expanded nodes and requests to resources for information. Table 6.3 shows a comparison of expands and requests between collaborative uniform-cost and uniform-cost for each of the problem types. For both types of problem, collaborative uniform-cost expands more nodes than uniform-cost. The reason behind this result is that, in the collaborative search model, each search agent has only partial knowledge of the search process, so it selects nodes to be expanded based on its limited knowledge. This leads to the expansion of unpromising nodes. However, collaborative uniform-cost sends requests to resources substantially less than uniform-cost. The difference is as much as over 50%. The gain in terms of requests is thanks to the agent collaboration and the goal verification procedure implemented in the collaborative model. Each request to determine the cost between 2 nodes is forwarded to the search agent responsible who has the knowledge to determine the relevancy of

Environment 4	
Environment properties	**Description**
Organisational structure	3-level depth hierarchy (1, 10, 100)
Locations	10000
Accessible locations from a location	5
Exit points from an organisational entity	5
Environment 5	
Environment properties	**Description**
Organisational structure	3-level depth hierarchy (1, 10, 100)
Locations	10000
Accessible locations from a location	3
Exit points from an organisational entity	3
Environment 6	
Environment properties	**Description**
Organisational structure	3-level depth hierarchy (1, 5, 50)
Locations	10000
Accessible locations from a location	3
Exit points from an organisational entity	3
Environment 7	
Environment properties	**Description**
Organisational structure	3-level depth hierarchy (1, 2, 10)
Locations	10000
Accessible locations from a location	3
Exit points from an organisational entity	3
Environment 8	
Environment properties	**Description**
Organisational structure	3-level depth hierarchy (1, 10, 100)
Locations	10000
Accessible locations from a location	1
Exit points from an organisational entity	1
Environment 9	
Environment properties	**Description**
Organisational structure	3-level depth hierarchy (1, 10, 100)
Locations	5000
Accessible locations from a location	3
Exit points from an organisational entity	3
Environment 10	
Environment properties	**Description**
Organisational structure	3-level depth hierarchy (1, 10, 100)
Locations	1000
Accessible locations from a location	3
Exit points from an organisational entity	3

Table 6.2: Environment 4–10

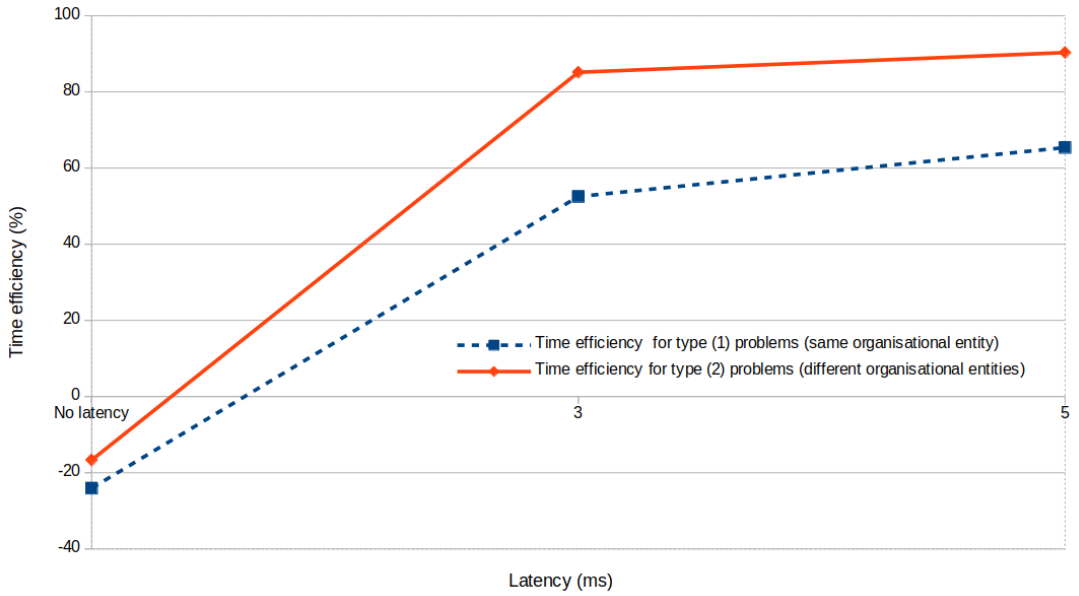

Figure 6.5: Runtime efficiency of collaborative uniform-cost compared to uniform-cost based on problem types

Problem type	Collaborative uniform-cost		Uniform-cost	
	Expands	Requests	Expands	Requests
Same organisational entity	112	229	95	475
Different organisational entities	6884	12617	5950	30043

Table 6.3: Expands and requests based on problem types

the request, and thus decides whether to send the request to a resource agent or to drop the request. In addition, collaborative and parallel search leads to a rapid discovery of the destination, irrespective of its optimality. Once the destination is found, the goal verification procedure is initiated, informing all search agents about the destination. While the destination is under local verification, search agents use the knowledge about the found destination to filter unpromising nodes and requests such as the nodes having higher cost than the found destination and the requests whose start node is more expansive than the destination under verification.

With latency, collaborative uniform-cost performs better than uniform-cost even though collaborative uniform-cost expands more nodes. There are 3 main reasons behind this result. First, node expansion is executed in parallel by multiple search agents. Second, node expansion is critical because of the latency resulting from the requests to resource agents. In collaborative uniform-cost, the handling of requests to resource agents is done in an asynchronous manner, so agents are not blocked while waiting for the response. Third, collaborative uniform-cost sends significantly less requests compared to uniform-cost.

6.2.3 Experiment 2: Evaluating the search model over different environment structures

In the second experiment, we compared collaborative uniform-cost with uniform-cost based on environment structures. We use 2 criteria for differentiating environment structures, namely node distribution and connectivity. Node distribution is how nodes are distributed among the organisational entities of an environment. Connectivity refers to the connections between nodes and between organisational entities.

To measure the performance of the search model in function of node distribution, we experimented collaborative uniform-cost in 3 environments, namely Environment 5, 6, and 7. These 3 environments are of the same size, but have different organisational structures. The organisational structure of Environment 5 is a 3-level depth hierarchy consisting of 1 organisational entity at the first level, 10 at second, and 100 at third, while that of Environment 6 and Environment 7 are a 3-level depth hierarchy consisting of 1 organisational entity at the first level, 5 at second, and 50 at third and a 3-level depth hierarchy consisting of 1 organisational entity at the first level, 2 at second, and 10 at third, respectively. Nodes are more distributed in Environment 5 (100 per leaf organisational entity) than in Environment 6 (200 per leaf organisational entity) and in Environment 7 (1000 per leaf organisational entity). The results, as depicted in Figure 6.6, suggest that collaborative uniform-cost is most efficient in Environment 5 in which nodes are the most distributed among the 3 environments. In Environment 7, each 1000 nodes are grouped under the same organisational entity. Therefore, collaborative uniform-cost can mostly only distribute the workloads among search agents exploring the search space under the same organisational entity. The algorithm has to expand many nodes to find an exit to other parts of the search space allowing the search process to spread. In such a condensed environment, uniform-cost finds the result quickly as the search is simple and there are few exits. Collaborative uniform-cost starts to surpass uniform-cost from 3 ms of latency.

To evaluate the performance of the collaborative search model in terms of connectivity, we experimented collaborative uniform-cost in 3 environments of different connectivity, namely Environment 4, 5, and 8. Figure 6.7 shows time efficiency we gain using collaborative uniform-cost compared to uniform-cost. Collaborative uniform-cost performs best in Environment 4 because there are more connections among nodes and more exits to other organisational entities. The collaborative search model does not use a predefined method to separate the search space. It dynamically and progressively distributes the search space among agents based on the structure of the environment. More exits allow agents to reach more parts of the graph faster, and thus finding the destination faster. More connections among nodes also lead to a more efficient search since more nodes can be explored by parallel search agents. Regardless of the connectivity, uniform-cost performs better than collaborative uniform-cost when there is no latency. However, collaborative uniform-cost overtakes uniform-cost when latency is introduced.

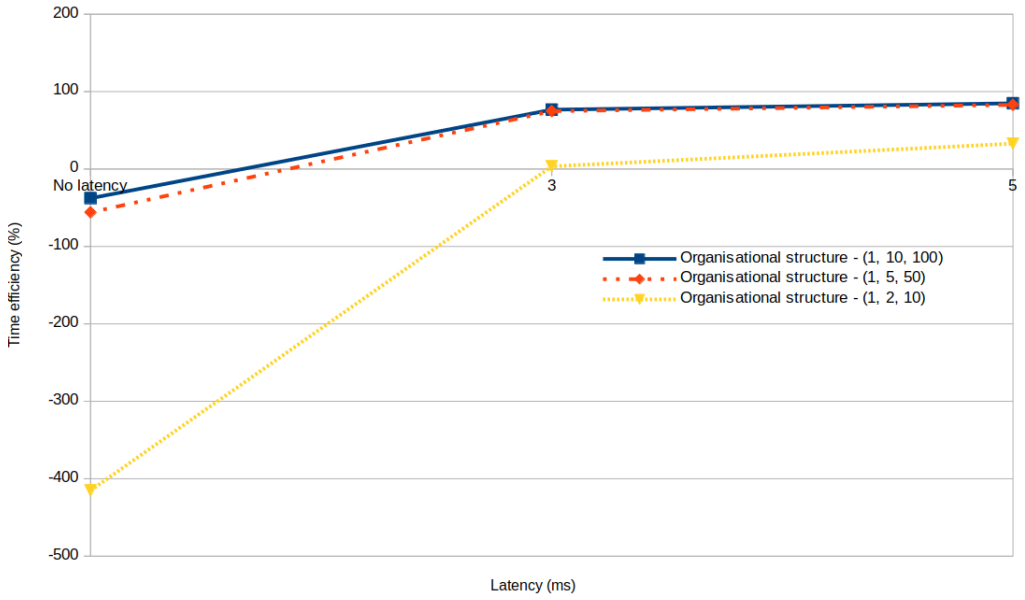

Figure 6.6: Runtime efficiency of collaborative uniform-cost compared to uniform-cost over different node distributions

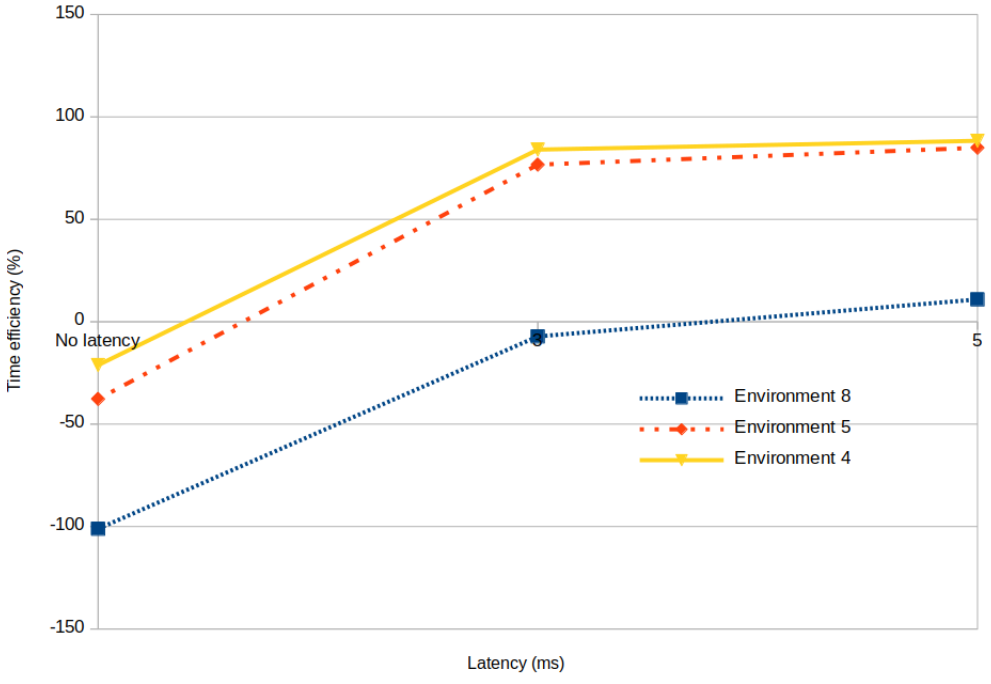

Figure 6.7: Runtime efficiency of collaborative uniform-cost compared to uniform-cost over different connectivity

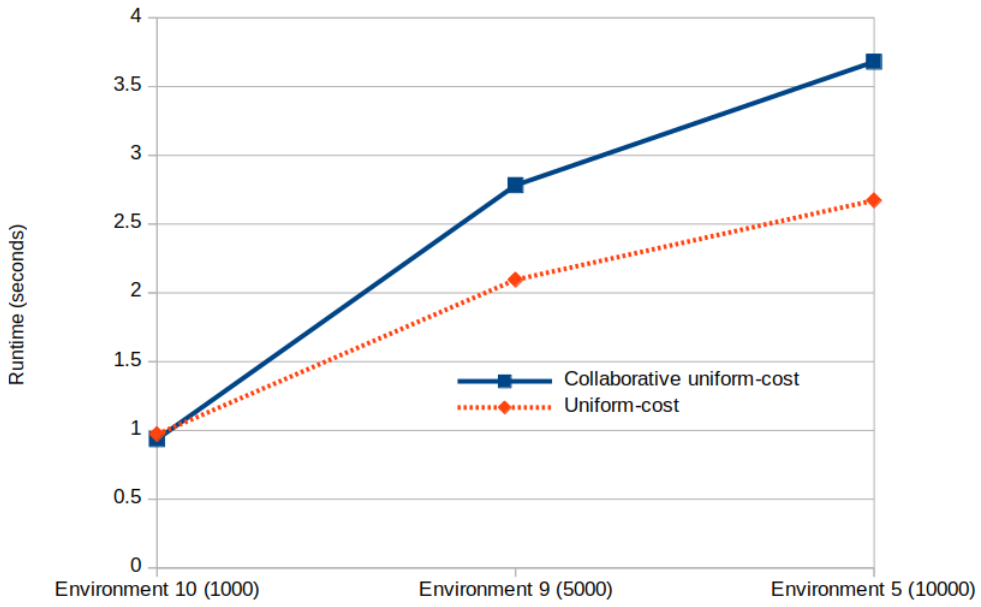

Figure 6.8: Runtime over different graph sizes – No latency

6.2.4 Experiment 3: Evaluating the scalability of the search model

The objective of the third experiment is to examine the scalability of the collaborative search model in function of environment sizes and latency. In this experiment, we compared collaborative uniform-cost with uniform-cost in 3 environments of the same structure but different sizes, namely Environment 5, 9, and 10.

Figure 6.8 shows the order of growth of both algorithms in function of environment sizes, without latency. The results suggest that uniform-cost scales better as the size of the environment grows. As mentioned in previous sections, this is due to the overheads required in the collaboration among agents in collaborative uniform-cost. When latency is introduced, collaborative uniform-cost outperforms uniform-cost in all the environments, as illustrated in Figure 6.9. Remarkably, this result also shows that collaborative uniform-cost scales much better than uniform-cost in function of latency.

6.2.5 Discussion

Based on these experiments, uniform-cost performs better than collaborative uniform-cost in all cases when there is no latency. However, in most cases, collaborative uniform-cost starts to outperform uniform-cost when a latency of around 2 ms is present. In addition to latency, problem types and graph structures have a significant influence on the collaborative search model. As shown in previous sections, collaborative uniform-cost is more efficient in solving complex problems that involve various organisational entities in well-connected and more distributed environments.

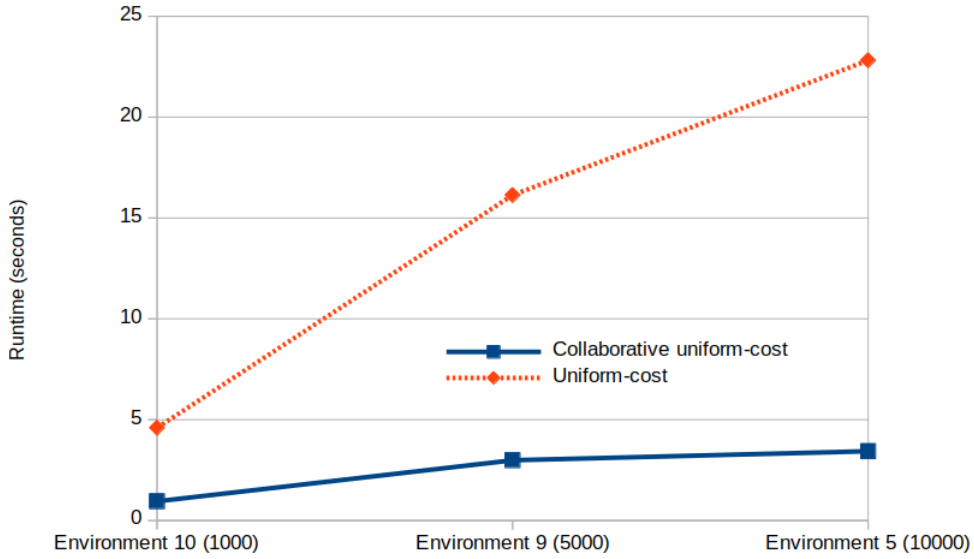

Figure 6.9: Runtime over different graph sizes – 5 ms latency

6.3 Conclusion

In this chapter, we have presented the validation of the knowledge model along with the heuristics and the collaborative multi-agent search model. The results show that the heuristics are efficient when used to solve complex problems in a well-connected and distributed environment. The collaborative search model performs better than the serial counterpart in most cases when latency is present. Furthermore, like the heuristics, the search model is also more efficient in solving complex problems in well-connected and distributed environments. We may conclude that both the heuristics and the collaborative search model are suitable for solving problems in complex environments where there are numerous options to choose at each step, especially when there is latency. The factors including the types of problem, the complexity of the environment, and the latency can be used to decide whether the heuristics and the collaborative search model are suitable for a given search problem.

Part IV

CONCLUSION AND PERSPECTIVES

Chapter 7

Conclusions and Perspectives

Contents

The aim of this dissertation is to address multi-goal pathfinding in ubiquitous environments. Such environments are dynamic and open. The entities residing within the environments can be mobile, and their state may be dynamic. Consequently, the state of the environments may evolve over time. Solving problems in such environments requires up-to-date information about the environments. To address multi-goal pathfinding and, at the same time, the complexity of the environments, we proposed an approach that exploits data from relevant sources such as the Web and various entities located in the environments to compute the solution and to adapt the solution according to the evolution of the environments.

The objective of this chapter is to provide a global conclusion of our work presented in the dissertation. The rest of the chapter is divided into two sections. First, we summarise our contributions. Second, we present the directions for our future work to address the limitations of our approach as well as to improve the approach.

7.1 Contributions

To summarise our contributions, we categorise them according to the four research questions addressed in this dissertation.

Research question 1. *Obtaining pertinent information: How to acquire the information necessary for solving a multi-goal pathfinding problem?*

To answer to this research question, we proposed a conceptual knowledge model (see Section 3.1) that can be used to provide an abstraction for a ubiquitous environment. Via the knowledge model, we can describe an environment in a way that integrates all the elements necessary for solving multi-goal pathfinding which consist of the followings:

- the spatial dimension of the environment,

- the cybernetic, physical, and/or social entities located in the environment,
- the relevant resources from which useful information about the environment and the entities can be retrieved,
- and the relationships between the entities and the goals that can be used to determine through which entity a goal can be satisfied.

Such an abstraction provides the knowledge necessary for solving multi-goal pathfinding problems. For this first research question in particular, the environment abstraction enables us to determine which information sources should be used to retrieve the necessary data to solve a given problem.

To provide a formal and concrete implementation of the knowledge model, we proposed an ontology, entitled Ubiquitous Environment Abstraction Ontology (see Section 3.2) and abbreviated as `ueao`, to represent the knowledge captured in the model. The ontology `ueao` provides the generic classes and properties for describing ubiquitous environments. We extended `ueao` to design another ontology entitled Smart Airport Activity Ontology `saa` (see Section 3.3) to describe a smart airport by incorporating the types of entities commonly present in a smart airport and the common activities that can be carried out in the airport. To create these ontologies, we use the Web Ontology Language (OWL). Basing the knowledge model on RDF, ontology, and OWL provides us three main advantages (see Section 3.2.3) including the flexibility in the manner in which the description of the environment is stored, the ability to integrate various kinds of data in the description, and to ability to reason upon the knowledge about the environment.

Research question 2. *Handling latency: How to address the latency of accessing data sources and transferring data from the sources?*

Using up-to-date information from relevant sources is essential owing to the dynamics of the environment. However, accessing and retrieving data from various sources creates the overheads resulting from the latency of processing and transferring data. We proposed a collaborative multi-agent search model (see Section 4.1) that is capable of mitigating latency and improving search efficiency.

The collaborative search model separates the process of accessing to resources for information from the search process by employing search agents to perform the search and resource agents to perform the resource-related operations. The interactions between search agents and resource agents are asynchronous and non-blocking. This allows the search to proceed while the information is being retrieved, and thus reduces the impact of latency on the search process.

Furthermore, the proposed search model combines distribution of the search space with parallelism of the search process to improve search efficiency. Different parts of the search space are explored in parallel, and useful information is exchanged among agents. The collaboration among different agents performing distributed searches in parallel allows the algorithm to find the path more efficiently.

Research question 3. *Beyond the classical pathfinding: How to solve multi-goal pathfinding?*

To address this research question, we proposed an approach (see Section 5.1) to solve multi-goal pathfinding. In this approach, we employ our knowledge model to describe the environment (see Section 5.2) providing the knowledge necessary for solving multi-goal pathfinding problems.

Given a multi-goal pathfinding problem, our approach extracts the necessary knowledge from the description of the environment (see Section 5.3) such as in which locations a given goal can be satisfied. The extracted knowledge is used to generate a goal-space graph (see Section 5.4) that represents the relations between the given goals and the locations where the goals can be carried out in the given order.

We construct a multi-layered graph by combining the goal-space graph, as the first layer, with the environment graph, as the second (see Section 5.5). We use a search algorithm on the first layer to determine at which location each goal should be satisfied. On the second layer, we employ a search algorithm to find the optimal path to connect the selected locations from the first layer. The collaborative multi-agent search model can be applied to the search algorithm used, especially on the second layer where the graph is large and there are accesses to resources to compute the path cost in order to address the latency and improve the search process.

Research question 4. *Dynamics, mobility, and openness: How to address the dynamic state of the environment during path computation and path execution?*

To adapt the solution according the evolving state of the environment, we proposed a mechanism (see Section 5.6) that continuously detects the changes in the environment that affect the solution and incrementally updates the solutions accordingly. The mechanism uses the multi-agent infrastructure constructed during the initial search using the collaborative multi-agent search model to detect changes and to update the solution.

7.2 Perspectives

In this section, we present our perspectives to address the limitations of our approach as well as to improve the approach.

7.2.1 Perspectives in the improvement of the approach

Updating the environment description

In our approach, the description of the environment is indispensable for solving multi-goal pathfinding problems. It is the representation of the environment and provides the necessary information required to solve the problems. As the environment evolves, the description needs to be updated to provide a valid representation of the environment. Currently, updating the enviroment description has not been

addressed in our approach. It is noteworthy that there is a body of work in the literature such as [Ciortea 2015] that addresses the changes of the environment in the context of Internet of Things. Building upon such work, we may be able to detect the changes in the environment and update the description of the environment accordingly.

Handling the heterogeneity of data and resources

A research challenge that we do not currently address in our approach is how to handle the heterogeneity of data and resources. The way to access resources differs from one resource to another. Data retrieved from various resources are of heterogeneous formats and structures. To be able to access the resources and make use of the data in our approach, we need a way to address their heterogeneity. This is an important problem, but it is not the main focus of our approach. Therefore, our next step is to explore the literature for existing solutions to tackle this issue.

Associating goals to activities

In the proposed knowledge model, among other notions, we capture the concept of activity to model the relationships between entities and the activities that can be carried out via the entities (see Section 3.1). However, the notion of goals is not represented in our model. In the current state of our approach, we assume that there exits a function that we can use to determine the activity that can satisfy a given goal. For instance, a straightforward solution would be to store a list of common goals that can be satisfied in an environment and the activity that should be carried out to satisfy each goal. In our future work, we would like to explore various alternatives to handle this issue more efficiently and intelligently such as by using ontology and reasoning.

Reasoning to enhance the environment description

Currently, our approach functions under the assumption that the provided description of the environment is complete and accurate. However, in real practice, we may very well encounter the cases where the environment description is partial or incomplete or both. In such cases, reasoning upon the existing knowledge may allow us to discover more knowledge to complete and improve the description. This is one of the issues that we would also like to address in our future work. It is also noteworthy that the ability to reason upon the knowledge in the description is possible thanks to the use of ontology and OWL.

7.2.2 Perspectives in the validation of the approach

In this dissertation, we have provided the validation of different components of the approach via various experiments. However, due to time constraint, we were not able to complete our experiments on the mechanism for handling the dynamics of the environment. To fully validate our approach, we are currently conducting the experiments on the said mechanism. The next step is to validate the approach as a whole by integrating all of the components.

Furthermore, the experiments were conducted using randomly generated environments. As a part of our future work, we would like to extend the validation of our approach to real environments in order to observe how the approach functions when various particularities of the real environments are introduced.

7.2.3 Perspectives in the application of the approach

Multi-goal pathfinding can be found in different fields such as trip planning, robotics, and logistics. In this dissertation, we proposed an approach to solving multi-goal problems that is tailored for trip planning. Since our field of focus is smart mobility and transportation, for our future work, we would like to apply our approach to develop a trip planner for ubiquitous environments such as smart buildings and smart transits. Nevertheless, we would also like to explore other fields where our approach can be applicable, potentially the logistics and robotics.

Publications

O. Kem, F. Balbo and A. Zimmermann. *Collaborative Search for Multi-goal Pathfinding in Ubiquitous Environments.* In Jan Ole Berndt, Paolo Petta and Rainer Unland, editeurs, Multiagent System Technologies, pages 72–88, Cham, 2017. Springer International Publishing.

O. Kem, F. Balbo, A. Zimmermann and P. Nagellen. *Multi-goal Pathfinding in Cyber-Physical-Social Environments: Multi-layer Search over a Semantic Knowledge Graph.* Procedia Computer Science, vol. 112, pages 741–750, 2017. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 21st International Conference, KES-2017, 6-8 September 2017, Marseille, France.

O. Kem, F. Balbo and A. Zimmermann. *Multi-goal Pathfinding in Ubiquitous Environments: Modeling and Exploiting Knowledge to Satisfy Goals.* In Proceedings of the International Conference on Web Intelligence, WI '17, pages 1147–1150. ACM.

O. Kem, F. Balbo and A. Zimmermann. *Traveler-Oriented Advanced Traveler Information System based on Dynamic Discovery of Resources: Potentials and Challenges.* Transportation Research Procedia, vol. 22, pages 635–644, 2017. 19th EURO Working Group on Transportation Meeting, EWGT2016, 5-7 September 2016, Istanbul, Turkey.

O. Kem, F. Balbo and A. Zimmermann. *A Distributed Approach to Constructing Travel Solutions by Exploiting Web Resources.* In 2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI), pages 713–716, Oct 2016.

Bibliography

[Abdalla 2014] Amin Abdalla, Yingjie Hu, David Carral, Naicong Li and Krzysztof Janowicz. *An ontology design pattern for activity reasoning.* In Proceedings of the 5th International Conference on Ontology and Semantic Web Patterns-Volume 1302, pages 78–81. CEUR-WS. org, 2014. (Cited on page 50.)

[Adler 1998] Jeffrey L Adler and Victor J Blue. *Toward the design of intelligent traveler information systems.* Transportation Research Part C: Emerging Technologies, vol. 6, no. 3, pages 157–172, 1998. (Cited on page 28.)

[Algfoor 2015] Zeyad Abd Algfoor, Mohd Shahrizal Sunar and Hoshang Kolivand. *A comprehensive study on pathfinding techniques for robotics and video games.* International Journal of Computer Games Technology, vol. 2015, page 7, 2015. (Cited on page 12.)

[Asad J. Khattak 1994] Haitham M. Al-Deek Asad J. Khattak. *Concept of an Advanced Traveler Information System Testbed for the Bay Area: Research Issues.* Journal of Intelligent Transportation Systems, vol. 2, no. 1, pages 45–71, 1994. (Cited on page 34.)

[Bektas 2006] Tolga Bektas. *The multiple traveling salesman problem: an overview of formulations and solution procedures.* Omega, vol. 34, no. 3, pages 209–219, 2006. (Cited on page 14.)

[Campbell 2011] P.A. Campbell, J.R. Junger, J.P. Havlicek, A.R. Stevenson and R.D. Barnes. *Pathfinder: Oklahoma's advanced traveler information system.* In Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on, pages 402–407, Oct 2011. (Cited on page 34.)

[Catarci 2006] Tiziana Catarci, Benjamin Habegger and Antonella Poggi. *Intelligent user task oriented systems.* Personal Information Management: Now That We're Talking, What Are We Learning?, page 20, 2006. (Cited on page 50.)

[Chen 1993] J.-Y. Chen and P. Yang. *Shanghai Urban Traffic Route Guidance System.* In Vehicle Navigation and Information Systems Conference, 1993., Proceedings of the IEEE-IEE, pages 217, A8–12, Oct 1993. (Cited on page 30.)

[Chiu 2001] D.K.W. Chiu. *A script language for generating Internet-bots.* In Database and Expert Systems Applications, 2001. Proceedings. 12th International Workshop on, pages 667–671, 2001. (Cited on page 34.)

[Chiu 2005] D.K.W. Chiu, O.K.F. Lee, Ho fung Leung, E.W.K. Au and M.C.W. Wong. *A Multi-Modal Agent Based Mobile Route Advisory System for Public Transport Network.* In System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on, pages 92b–92b, Jan 2005. (Cited on pages 28 and 34.)

[Ciortea 2015] A. Ciortea, A. Zimmermann, O. Boissier and A. M. Florea. *Towards a Social and Ubiquitous Web: A Model for Socio-Technical Networks.* In 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), volume 1, pages 461–468, Dec 2015. (Cited on pages 6 and 134.)

[Cyganiak 2014] Richard Cyganiak, David Wood and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014*, February 25 2014. (Cited on page 49.)

[Czogalla 2015a] Olaf Czogalla. *Smart phone based indoor navigation for guidance in public transport facilities.* In Proceedings of the 2nd IFAC Conference on Embedded Systems, Computer Intelligence and Telematics CESCIT, pages 233–239, 2015. (Cited on page 32.)

[Czogalla 2015b] Olaf Czogalla and Sebastian Naumann. *Pedestrian guidance for public transport users in indoor stations using smartphones.* In Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on, pages 2539–2544. IEEE, 2015. (Cited on page 32.)

[Czogalla 2016] Olaf Czogalla and Sebastian Naumann. *Pedestrian indoor navigation for complex public facilities.* In Indoor Positioning and Indoor Navigation (IPIN), 2016 International Conference on, pages 1–8. IEEE, 2016. (Cited on page 32.)

[Dean 1988] Thomas L Dean and Mark S Boddy. *An Analysis of Time-Dependent Planning.* In AAAI, volume 88, pages 49–54, 1988. (Cited on page 23.)

[ETSI 2011] ETSI. *Intelligent Transport Systems (ITS); Testing; Framework for conformance and interoperability testing.* Rapport technique, ETSI, 2011. (Cited on page 28.)

[Evett 1995] Matthew Evett, James Hendler, Ambuj Mahanti and Dana Nau. *PRA*: Massively parallel heuristic search.* Journal of Parallel and Distributed Computing, vol. 25, no. 2, pages 133–143, 1995. (Cited on page 25.)

[Fox 1993] Mark S Fox, John F Chionglo and Fadi G Fadel. *A common-sense model of the enterprise.* In Proceedings of the 2nd Industrial Engineering Research Conference, volume 1, pages 425–429, 1993. (Cited on page 50.)

[Fukunaga 2017] Alex Fukunaga, Adi Botea, Yuu Jinnai and Akihiro Kishimoto. *A Survey of Parallel A.* arXiv preprint arXiv:1708.05296, 2017. (Cited on pages 12 and 19.)

[GDF 2015] *GDF.* `http://wiki.openstreetmap.org/wiki/GDF`, 2015. Accessed: 2015-02-02. (Cited on page 36.)

[Ghallab 2016] Malik Ghallab, Dana Nau and Paolo Traverso. Automated planning and acting. Cambridge University Press, 2016. (Cited on pages 12, 65 and 99.)

[Group 2012] ESG5 Technical Group. *DATEX II V2.1.* Rapport technique, European Commission, Directorate General for Mobility and Transport, May 2012. (Cited on page 35.)

[GTF 2015] *General Transit Feed Specification Reference.* `https://developers.google.com/transit/gtfs/reference`, 2015. Accessed: 2015-02-02. (Cited on page 36.)

[Hart 1968] P. E. Hart, N. J. Nilsson and B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pages 100–107, July 1968. (Cited on page 16.)

[Herbert 2008] W. Herbert and F. Mili. *Route guidance: State of the art vs. state of the practice.* In Intelligent Vehicles Symposium, 2008 IEEE, pages 1167–1174, June 2008. (Cited on page 30.)

[Hitzler 2012] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider and Sebastian Rudolph. *OWL 2 Web Ontology Language Primer (Second Edition), W3C Recommendation 11 December 2012*, December 11 2012. (Cited on page 49.)

[Hu 2002] Mingwei Hu, Yunfei Wang and Qixin Shi. *Developing Beijing traveler information systems framework.* In Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on, pages 381–386, 2002. (Cited on page 33.)

[IRANI 1986] KEKIB IRANI and YI-FON SHIH. *Parallel A-asterisk and AO-asterisk algorithms- An optimality criterion and performance evaluation.* In 1986 International Conference on Parallel Processing, University Park, PA, pages 274–277, 1986. (Cited on page 25.)

[Khanjary 2011] M. Khanjary, K. Faez, M.R. Meybodi and M. Sabaei. *PersianGulf: An Autonomous Combined Traffic Signal Controller and Route Guidance System.* In Vehicular Technology Conference (VTC Fall), 2011 IEEE, pages 1–6, Sept 2011. (Cited on page 30.)

[Khanjary 2012] M. Khanjary and S.M. Hashemi. *Route guidance systems: Review and classification.* In Telematics and Information Systems (EATIS), 2012 6th Euro American Conference on, pages 1–7, May 2012. (Cited on page 30.)

[Kishimoto 2009] Akihiro Kishimoto, Alex S Fukunaga, Adi Botea*et al. Scalable, Parallel Best-First Search for Optimal Sequential Planning.* In ICAPS, 2009. (Cited on page 25.)

[Kishimoto 2013] Akihiro Kishimoto, Alex Fukunaga and Adi Botea. *Evaluation of a simple, scalable, parallel best-first search strategy.* Artificial Intelligence, vol. 195, pages 222–248, 2013. (Cited on page 25.)

[Koenig 2002a] Sven Koenig and Maxim Likhachev. *Improved fast replanning for robot navigation in unknown terrain.* In Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on, volume 1, pages 968–975. IEEE, 2002. (Cited on pages 22 and 23.)

[Koenig 2002b] Sven Koenig and Maxim Likhachev. *Incremental a.* In Advances in neural information processing systems, pages 1539–1546, 2002. (Cited on pages 21 and 22.)

[Korf 1985] Richard E. Korf. *Depth-first Iterative-deepening: An Optimal Admissible Tree Search.* Artif. Intell., vol. 27, no. 1, pages 97–109, September 1985. (Cited on page 18.)

[Kumar 1988] Vipin Kumar, K Ramesh and V Nageshwara Rao. *Parallel Best-First Search of State-Space Graphs: A Summary of Results.* In AAAI, volume 88, pages 122–127, 1988. (Cited on page 25.)

[Kumar 2003] Praveen Kumar, Dhanunjaya Reddy and Varun Singh. *Intelligent transport system using GIS.* In 6th Annual International Conference, Map India, pages 28–30, 2003. (Cited on pages 28 and 34.)

[Kumar 2005] Praveen Kumar, Varun Singh and D. Reddy. *Advanced traveler information system for Hyderabad City.* Intelligent Transportation Systems, IEEE Transactions on, vol. 6, no. 1, pages 26–37, March 2005. (Cited on page 34.)

[Laporte 1992] Gilbert Laporte. *The traveling salesman problem: An overview of exact and approximate algorithms.* European Journal of Operational Research, vol. 59, no. 2, pages 231–247, 1992. (Cited on page 14.)

[Likhachev 2004] Maxim Likhachev, Geoffrey J Gordon and Sebastian Thrun. *ARA*: Anytime A* with provable bounds on sub-optimality.* In Advances in neural information processing systems, pages 767–774, 2004. (Cited on page 23.)

[Likhachev 2005] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz and Sebastian Thrun. *Anytime dynamic a*: An anytime, replanning algorithm.* In ICAPS, pages 262–271, 2005. (Cited on pages 23 and 26.)

[Lim 2014] K. L. Lim, L. S. Yeong, S. I. Ch'ng, K. P. Seng and L. M. Ang. *Uninformed multigoal pathfinding on grid maps.* In 2014 International Conference on Information Science, Electronics and Electrical Engineering, volume 3, pages 1552–1556, April 2014. (Cited on page 13.)

[Liu 2014] Xuan Liu, John M Usher and Lesley Strawderman. *An analysis of activity scheduling behavior of airport travelers.* Computers & Industrial Engineering, vol. 74, pages 208–218, 2014. (Cited on page 58.)

[Lovicsek 1998] M. Lovicsek, S. Stewart and B. Delsey. *Integrating the collection, fusion, and dissemination of traveler information in Edinburgh, Scotland.* Mathematical and Computer Modelling, vol. 27, no. 9–11, pages 335–348, 1998. (Cited on page 34.)

[Matai 2010] Rajesh Matai, Surya Prakash Singh and Murari Lal Mittal. *Traveling salesman problem: An overview of applications, formulations, and solution approaches.* Traveling Salesman Problem, Theory and Applications, pages 1–24, 2010. (Cited on page 13.)

[Nissim 2012] Raz Nissim and Ronen I Brafman. *Multi-agent A* for parallel and distributed systems.* In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3, pages 1265–1266. International Foundation for Autonomous Agents and Multiagent Systems, 2012. (Cited on page 25.)

[Peng 2000] Zhong-Ren Peng and Ruihong Huang. *Design and development of interactive trip planning for web-based transit information systems.* Transportation Research Part C: Emerging Technologies, vol. 8, no. 1–6, pages 409–425, 2000. (Cited on page 30.)

[Phillips 2014] Mike Phillips, Maxim Likhachev and Sven Koenig. *PA* SE: Parallel A* for Slow Expansions.* In ICAPS, 2014. (Cited on page 25.)

[Russell 1992] Stuart Russell. *Efficient Memory-bounded Search Methods.* In Proceedings of the 10th European Conference on Artificial Intelligence, ECAI '92, pages 1–5, New York, NY, USA, 1992. John Wiley & Sons, Inc. (Cited on page 18.)

[Russell 2016] S.J. Russell and P. Norvig. Artificial intelligence: A modern approach. Always learning. Pearson Education, Limited, 2016. (Cited on pages 14, 15, 16, 17 and 18.)

[Stentz 1994] Anthony Stentz. *Optimal and efficient path planning for partially-known environments.* In Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, pages 3310–3317. IEEE, 1994. (Cited on page 21.)

[Stentz 1995] Anthony Stentz*et al. The focussed D^* algorithm for real-time replanning.* In IJCAI, volume 95, pages 1652–1659, 1995. (Cited on page 21.)

[Subakti 2016] Hanas Subakti and Jehn-Ruey Jiang. *A marker-based cyber-physical augmented-reality indoor guidance system for smart campuses.* In 2016 IEEE

18th International Conference on High-Performance Computing and Communications, IEEE 14th International Conference on Smart City, and IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pages 1373–1379. IEEE, 2016. (Cited on page 32.)

[Tra 2012] *Les développements liés à Transmodel*. http://www.billettique.fr/spip.php?article313, 2012. Accessed: 2015-01-28. (Cited on page 35.)

[transport 2007] Technical Committee CEN/TC 278 "Road transport and traffic telematics". *Road traffic and transport telematics — Public transpot — Identification of fixed objects in public transport*. Rapport technique, CEN/TC 278, December 2007. (Cited on page 36.)

[transport 2009] Technical Committee CEN/TC 278 "Road transport and traffic telematics". *NeTEx — Network and Timetable Exchange — Part 1: Network Topology*. Rapport technique, CEN/TC 278, August 2009. (Cited on page 36.)

[U.S. Department of Transportation 1998] U.S. Department of Transportation. *Developing Traveler Information Systems Using the National ITS Architecture*. Rapport technique, Mitretek Systems and TransCore, Inc., August 1998. (Cited on pages 28 and 29.)

[Vidal 2010] Vincent Vidal, Lucas Bordeaux and Youssef Hamadi. *Adaptive k-parallel best-first search: A simple but efficient algorithm for multi-core domain-independent planning*. In Third Annual Symposium on Combinatorial Search, 2010. (Cited on page 25.)

[Weiser 1993] Mark Weiser. *Some computer science issues in ubiquitous computing*. Communications of the ACM, vol. 36, no. 7, pages 75–84, 1993. (Cited on page 3.)

[Weiser 1999] M. Weiser, R. Gold and J. S. Brown. *The origins of ubiquitous computing research at PARC in the late 1980s*. IBM Systems Journal, vol. 38, no. 4, pages 693–696, 1999. (Cited on page 3.)

[Werner 2011] M. Werner. *Selection and Ordering of Points-of-Interest in Large-Scale Indoor Navigation Systems*. In 2011 IEEE 35th Annual Computer Software and Applications Conference, pages 504–509, July 2011. (Cited on pages 13 and 32.)

[Wu 2003] Chun-Hsin Wu, Da-Chun Su, Justin Chang, Chia-Chen Wei, Jan-Ming Ho, Kwei-Jay Lin and D. T. Lee. *An advanced traveler information system with emerging network technologies*. In Proc. 6th Asia-Pacific Conf. Intelligent Transportation Systems Forum, pages 230–231, 2003. (Cited on page 34.)

[Yamaguchi 1999] M. Yamaguchi, T. Kitamura, S. Jinno and T. Tajima. *The interactive CDRG using infrared beacons.* In Intelligent Transportation Systems, 1999. Proceedings. 1999 IEEE/IEEJ/JSAI International Conference on, pages 278–283, 1999. (Cited on page 30.)

[Zhou 2002] Rong Zhou and Eric A Hansen. *Multiple Sequence Alignment Using Anytime A*.* In AAAI/IAAI, pages 975–977, 2002. (Cited on page 23.)

[Zilberstein 1995] Shlomo Zilberstein and Stuart Russell. *Approximate reasoning using anytime algorithms.* In Imprecise and Approximate Computation, pages 43–62. Springer, 1995. (Cited on page 23.)

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : 2018LYSEM023

Oudom KEM

MODELLING AND EXPLOITING KNOWLEDGE OF THE ENVIRONMENT: A MULTI-AGENT APPROACH TO MULTI-GOAL PATHFINDING IN UBIQUITOUS ENVIRONMENTS

Speciality: Computer science

Keywords: Search algorithms, Multi-agent systems, Semantic Web, Cyber-physical systems

Abstract:

From intelligent artificial personal assistants to smart cities, we are experiencing the shifting towards Internet of Things (IoT), ubiquitous computing, and artificial intelligence. Cyber-physical entities are embedded in social environments of various scales from smart homes, to smart airports, to smart cities, and the list continues.

This paradigm shift coupled with ceaseless expansion of the Web supplies us with tremendous amount of useful information and services, which creates opportunities for classical problems to be addressed in new, different, and potentially more efficient manners. Along with the new possibilities, we are, at the same time, presented with new constraints, problems, and challenges.

Multi-goal pathfinding, a variant of the classical pathfinding, is a problem of finding a path between a start and a destination which also allows a set of goals to be satisfied along the path. The aim of this dissertation is to propose a solution to solve multi-goal pathfinding in ubiquitous environments such as smart transits.

In our solution, to provide an abstraction of the environment, we proposed a knowledge model based on the semantic web technologies to describe a ubiquitous environment integrating its cybernetic, physical, and social dimensions. To perform the search, we developed a multi-agent algorithm based on a collaborative and incremental search algorithm that exploits the knowledge of the environment to find the optimal path. The proposed algorithm continuously adapts the path to take into account the dynamics of the environment.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : 2018LYSEM023

Oudom KEM

MODÉLISATION ET EXPLOITATION DES CONNAISSANCES DE L'ENVIRONNEMENT : UNE APPROCHE MULTI-AGENTS POUR LA RECHERCHE D'ITINÉRAIRES MULTI-OBJECTIFS DANS DES ENVIRONNEMENTS UBIQUITAIRES

Spécialité: Informatique

Mots clefs : Algorithmes de recherche, Systèmes multi-agents, Web sémantique, Systèmes cyber-physiques

Résumé :

L'utilisation des téléphones intelligents, le recours aux assistants personnels intelligents ou encore le développement des maisons intelligentes sont autant d'exemples illustrant le développement toujours plus rapide de l'informatique ubiquitaire, de l'Internet des objets et de l'intelligence artificielle. Le croisement des résultats issus de ces domaines de recherche contribue à changer notre quotidien et constitue un environnement fertile pour de nouveaux travaux. Ainsi, l'intégration des entités cyber-physiques dans des environnements sociaux de différentes échelles allant des maisons aux villes intelligentes amène de très nombreuses perspectives.

Ce changement de paradigme met à notre disposition une énorme quantité d'informations et de services utiles, offrant ainsi la possibilité de traiter les problèmes classiques de manière nouvelle, différente et potentiellement plus efficace. Si les solutions à construire bénéficient de ces possibilités, elles doivent également répondre à de nouvelles contraintes et nouveaux défis.

La recherche d'itinéraires multi-objectifs est un sous-cas du problème classique de recherche d'un chemin entre un lieu de départ et une destination auquel s'ajoute la contrainte de passage par un ensemble de lieux permettant de satisfaire un ensemble de buts. L'objectif de cette thèse est de proposer une solution pour la résolution de la recherche d'itinéraires multi-objectifs appliqués aux environnements cyber-physiques tels que les Smart Transits.

Dans notre solution, nous avons proposé une méthode fondée sur les technologies du web sémantique pour modéliser de manière intégrée un environnement cyber-physique dans toutes ses dimensions, i.e., cybernétiques, physiques et sociales. Pour la recherche de chemin, nous avons proposé une approche multi-agents, exécutant un algorithme de recherche collaborative et incrémentale, qui utilise les connaissances de l'environnement pour trouver le chemin optimal. Cet algorithme adapte aussi le chemin en prenant en compte la dynamique de l'environnement.