Collaborative Route Finding in Semantic Mazes

Katharine Beaumont[1], Eoin O'Neill[1], Nestor Velasco Bermeo[1] and Rem Collier[1]

[1]*UCD School of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland*

Abstract

The following document describes the submission to the All The Agents Challenge of a system that interacts with the Autonomous Maze Environment Explorer Project (AMEE)[1]. Our solution is a collaborative route finding service for Semantically Defined Mazes. Agents with Reinforcement Learning tools are used to discover paths through the maze. These agents collaborate by sharing schematic knowledge of the maze (information about the maze layout). The code repository and video demonstration locations are detailed in the Online Resources section at the end of the document.

Video: https://youtu.be/b2tecNJc0DE

Source Code: https://gitlab.com/mams-ucd/atac-maze

Keywords

Semantic Web, Semantic Agents, Intelligent Agents, Reinforcement Learning

1. Introduction

A challenge of semantic web technologies is the conceptual integration of heterogeneous sources of data [1]. A possible solution is the creation of ontologies for knowledge representation, but these can be difficult to maintain and reuse [1]. Furthermore, there exists a proliferation of sometimes contradictory ontologies of varying levels of detail [2].

Over the last two decades, in semantic web research there has been a shift of focus from ontologies, to linked data, to knowledge graphs [1]. A major challenge in agent-oriented programming is the complexity of programming agents, and building multi-agent systems. Once deployed, multi-agent systems can be difficult to amend and adapt to new technologies.

Thus another challenge is how to future proof intelligent web agents. When creating a multi-agent system comprised of intelligent web agents, interacting with semantic web resources, there is a need to balance the integration of specific ontologies and schemas with agent reuse.

This paper proposes a starting point to providing this balance: using state of the art web technologies, we present a system of BDI agents programmed with abstract goals and plans, combined with modules that provide goal and plan implementations: integrating schemas and machine learning algorithms, whilst allowing for knowledge sharing between agents. This modular approach allows for a system of agents that can be augmented with different learning algorithms and/or different schemas, with very little change to the agent code, whilst preserving the abilities of the agents to interact effectively with a semantic web environment.

[1] https://all-agents-challenge.github.io/atac2021

All the Agents Challenge (ATAC 2021)

katharine.beaumont@ucdconnect.ie (K. Beaumont); eoin.o-neill.3@ucdconnect.ie (E. O'Neill); nestorvb@ucd.ie (N. V. Bermeo); rem.collier@ucd.ie (R. Collier)

1.1. Technologies

The system provides an integration of semantic web, reinforcement learning, agent-oriented software and web technologies. It is a multi-agent microservices system (MAMS) comprised of Agent-Oriented MicroServices (AOMS)[3], and Plain-Old MicroServices (POMS). AOMS are microservices exposed through a well-defined interface modelled as a set of REpresentational StateTransfer (REST). They are built using MAS technology [3]. The AOMS is the Maze Navigation Service as presented in Figure 1.

The BDI programming language ASTRA is used to create BDI agents that are reactive and responsive to their environment, continually receiving environment events and updating their beliefs, which are incorporated into flexible plans [4]. They offer significant advantages in developing autonomous systems and allow for the integration of a range of AI techniques [5].

Using ASTRA allows us to take advantage of the module mechanism for building internal libraries that implement custom terms, formulae, sensors, actions and events [6]. A single agent can create several copies of the same module, with different names and states [6]. The modules are written in Java and accessed via the Module API.

In this system, the aim is to strike a balance between knowledge-rich and knowledge-lean techniques to explore the environment, by using a module (Jena Module) to interface with semantic web technologies in order to provide domain knowledge, and a module (Navigator Module) to process data gained from exploring the environment. The modular composition avoids a strong commitment to one type of knowledge representation.

Reinforcement learning capabilities are provided in the Navigator Module. The module contains an implementation of the Q Learning algorithm. Q-Learning uses Temporal Difference learning which samples from the environment and performs updates based on estimates that are revised over multiple episodes: learning is on-line and incremental [7]. It is policy based: policy learning can be seen as goal-directed [8]. It is suitable to dynamic, interactive environments. Integrated with the agents, there are essentially two tiers of goals: a reinforcement learning goal which is triggered by user interaction, and agent-level goals in the traditional BDI sense.

2. System architecture

The Maze Navigation Service comprises of two BDI agent types: a main managing agent (Navi), and multiple goal-based path finding subagents (PathFinder). The subagents employ a KnowledgeStore module that incorporates Apache Jena, and a Navigator Module that provides reinforcement learning-based navigation capabilities to support the exploration of semantic mazes. Other modules model relevant ontologies.

Agents should be viewed as BDI agents with a reinforcement learning tool, rather than as hybrid BDI-reinforcement learning agents [9]. The programmed goal and plans associated with BDI architecture are more abstract, and the reinforcement learning module provides implementation details required to navigate the maze and learn a policy regarding the path to a goal room. The (BDI) goal of the agents is to respond to user input, by locating a path to whichever room is required, using the reinforcement learning module. The reinforcement learning goal is the required room. The (BDI) plans provide iterative interaction with the environment, according to direction from the reinforcement learning module, which maps it.

The use of modules in the system goes beyond augmenting the agents with additional knowledge representation technologies as certain behaviours (for example choosing the next room to enter) are delegated to the module, and are not directly controlled by the agent. The agent controls the learning cycle, but not the learning process. The modules do not alter the beliefs, desires or intentions of the agent, but provide a learning tool or resource. The system also provides for knowledge sharing between agents using Java objects which can be manipulated in the modules, via the main agent.

A Spring Boot Application (a POMS, the Query Manager) provides a visual interface between the user and the Maze Navigation Service [10]. Navi has the url of the Query Manager as an initial belief, and on startup, it sends a post request to a registration endpoint. When the Query Manager gets an incoming user request for a room from the User Input, it forwards it to the registered main agent, which has REST endpoints to receive the request.

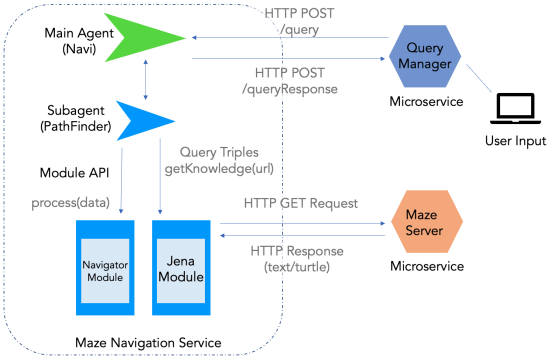

Figure 1: System Diagram

2.1. Maze Navigation System

Navi acts as a manager, and creates PathFinder subagents as required. When created, beliefs are added to Navi about the subagents available and the cumulative schematic knowledge collected by them. PathFinder subagents use individual copies of key modules that allow them to both interrogate the RDF schema for the maze, and interact with it.

There are two types of communication in the system. Inter-agent communication occurs between Navi and the PathFinder subagents communicate using ASTRA, which implements a FIPA ACL - based messaging infrastructure. Human-system communication occurs via a REST API, as described at the end of the last section.

2.2. Jena Module (RDF Knowledge Store)

PathFinder subagents have individual in-memory RDF knowledge stores. The Apache Jena library, an open source framework Semantic Web and Linked Data application[11], was integrated into the subagents via the Jena module. This enables the subagents to interact with the Maze Server over HTTP via GET requests. They initialise a Jena model which is an interface for the creation, parsing and storage of triples.

2.3. Collaborative Semantic Navigation Process

Throughout successive queries, PathFinder subagents collaborate via the agent Navi as they share schematic knowledge about the maze, which is then communicated with any new subagents.

On receiving a query, Navi loops through its list of existing subagents and communicates with each of them in turn to see if they have any knowledge of the room in question. If the room in question is their goal room, their belief about the path to the room is returned. If the room in question is a step in their path to a different goal room, the subpath is returned.

If no subagents have knowledge of the requested room, or no subagents exist yet, a new PathFinder subagent is created. Any schematic knowledge of the maze is shared with the new subagent by Navi in the `LocationStore` object. PathFinder has the belief "goal room" initialised as the requested room. It also sets a goal room variable in the Navigator module. It has additional beliefs concerning the starting url of the maze, current url, and the current and maximum number of iterations over which to perform the learning process.

One iteration of learning involves PathFinder going from the starting room of the maze to the goal room, or reaching the maximum number of steps (not finding the room). The subagent controls the length of the learning process through associate beliefs.

Starting at the maze entry point, the Apache Jena module reads the current url, which informs it of adjacent rooms. These are passed to the PathFinder subagent, which passes them in turn to the Navigator module. The Navigator module performs a step of the Q Learning algorithm. It calculates the score of the move from the previous room into the current one. The score is calculated using awareness of the objective (arrival at the user-input goal room) instead of environmental reward signals. Once the goal room is reached, an artificial score is allocated. This differs from the Q Learning algorithm which takes environmental feedback as the score.

The Navigator module contains code that decides whether to explore a new room next (pick randomly) or pick the adjacent room with the highest score (exploit environmental knowledge). The url of the next room to move to is passed to the PathFinder subagent, which updates it's belief on the current url and begins the next step of the current iteration.

The Navigator module also checks whether the goal room has been reached, or whether the number of steps taken to try and find the goal room is too high, and ends the process. This may happen in more complex environments, or when the room does not exist. It signifies the end of the process to the PathFinder subagent by passing back a url of "end", which resets the current url belief to the starting url, and increments the number of iterations. Over successive iterations, the path is refined. Once complete, PathFinder gets the path from the Navigator module and informs Navi via a message. Navi posts the path (`Path`) to the Query Manager. PathFinder sends it's schematic knowledge (`LocationStore`) to Navi, to share with any new PathFinder subagents. If the goal room was not found over all iterations, an error is returned to Navi, which is posted to the Query Manager.

3. Knowledge and Learning

Knowledge and learning in the agents is in memory and not persisted beyond the lifespan of the program. In addition to objects used in the RDF Knowledge Store, three key objects for navigation are defined and shared across agents, `Location`, `LocationStore` and `Path`.

`Location` represents a room in the maze, storing the room url, name and map of adjacent locations (direction-room url pairs). It also has a map which represents it's score in the reinforcement learning process. The object is created in the subagent's instance of the Navigator module, which caches the objects during the learning process. At the end of learning, the `Location` objects have their scores removed (as they are individual to subagent goal rooms) and as a collection are passed from the module to the subagent in the `LocationStore` object. This is stored as a belief, and sent to Navi via a message. Navi amalgamates it's existing copy (if it has one) with new `Location` objects, and shares it's `LocationStore` with any new PathFinder subagents. The result is that a URL need only be read once. The `Location` object thus has a dual purpose: acting as both global schematic knowledge and facilitating the individual learning process.

`Path` object: At the end of the learning process, the Navigator module returns a `Path` to the PathFinder subagent, which stores it as a belief. This is an ordered list of pairs of direction: room name, starting with the entry point. Printing the list forms a human-readable description of steps with directions from the entry point to the goal room. This is shared with Navi, and returned to the user.

4. Existing work and challenges

The combination of BDI agents with semantic web technology has been broached with JASDL [12]. This uses an annotation system in a way that is similar to the way our approach uses ASTRA modules. However this can be seen as a tighter integration, with a strong coupling with an agent's beliefs. In the system presented in this paper, the use of modules is more akin to adding a skill to the agent, that informs but does not integrate with, nor enforce the consistency of beliefs. It is a more loosely coupled integration. Further, as it does not interact with the agent's belief base, it can be used on bigger datasets, and could be extended to use external databases to further this capability.

There is much existing work on the incorporation of learning into BDI agents (for example detailed in [4]), and in particular reinforcement learning (for example [9]).

A key challenge of the system is the scalability of the reinforcement learning process. It is met in part by the agents sharing schematic knowledge, which reduces the need to re-map the maze. In terms of the size of the data, a potential solution would be to introduce a module that interfaces with a database in place of the `LocationStore`. Another issue is the complication of configuring the reinforcement learning process. In addition, reinforcement learning doesn't scale well when agent needs to perform diverse set of tasks [13]. These present a challenge if reusing the agents for different environments. A potential solution is the dynamic insertion of modules at runtime, to explore the semantic environment and preform ontology discovery, learn which schemas are required, then download and parse them from a schema repository.

Acknowledgments

This research is funded under the SFI Strategic Partnership Pro-gramme (16/SPP/3296) and is co-funded by Origin Enterprises plc.

References

[1] P. Hitzler, A review of the semantic web field, Communications of the ACM 64 (2021) 76–83.

[2] A. Bernstein, J. Hendler, N. Noy, A new look at the semantic web, Communications of the ACM 59 (2016) 35–37.

[3] R. Collier, E. O'Neill, D. Lillis, G. O'Hare, Mams: Multi-agent microservices, in: Companion Proceedings of The 2019 World Wide Web Conference, 2019, pp. 655–662.

[4] S. Airiau, L. Padgham, S. Sardina, S. Sen, Incorporating learning in bdi agents, in: Workshop AAMAS: adaptive and learning agents and MAS (ALAMAS+ ALAg), ACM Estoril, 2008, pp. 49–56.

[5] R. H. Bordini, A. El Fallah Seghrouchni, K. Hindriks, B. Logan, A. Ricci, Agent programming in the cognitive era, Autonomous Agents and Multi-Agent Systems 34 (2020) 1–31.

[6] R. W. Collier, S. Russell, D. Lillis, Reflecting on agent programming with agentspeak (l), in: International Conference on Principles and Practice of Multi-Agent Systems, Springer, 2015, pp. 351–366.

[7] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[8] M. Bosello, A. Ricci, From programming agents to educating agents–a jason-based framework for integrating learning in the development of cognitive agents, in: International Workshop on Engineering Multi-Agent Systems, Springer, 2019, pp. 175–194.

[9] A.-H. Tan, Y.-S. Ong, A. Tapanuj, A hybrid agent architecture integrating desire, intention and reinforcement learning, Expert Systems with Applications 38 (2011) 8477–8487.

[10] V. Inc, Spring boot, 2021. URL: https://spring.io/projects/spring-boot.

[11] T. A. S. Foundation, Apache jena, 2021. URL: https://jena.apache.org/.

[12] T. Klapiscak, R. H. Bordini, Jasdl: A practical programming approach combining agent and semantic web technologies, in: International Workshop on Declarative Agent Languages and Technologies, Springer, 2008, pp. 91–110.

[13] C. Florensa, D. Held, X. Geng, P. Abbeel, Automatic goal generation for reinforcement learning agents, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 1515–1528. URL: https://proceedings.mlr.press/v80/florensa18a.html.