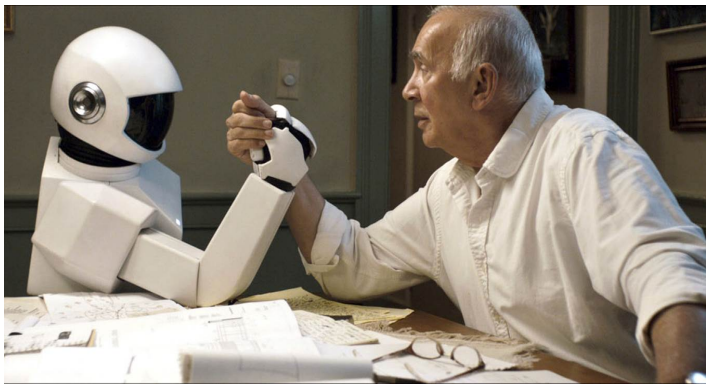
TÉCNICO LISBOA



On Planning in Human-Robot Collaboration

Guilherme de Mascarenhas Belo Antunes

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. José Alberto Rosado dos Santos Víctor
Researcher Paul Rudolph Schydlo

Examination Committee

Chairperson: Prof. João Manuel de Freitas Xavier
Supervisor: Prof. José Alberto Rosado dos Santos Víctor
Member of the Committe: Prof. Luís Manuel Marques Custódio

December 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

”A good human plus a machine is the best combination”.

Garry Kasparov

Acknowledgments

First of all, I would like to thank my co-supervisor, Paul Schydlo, for the dedication shown in helping me understand complex concepts and resolve technical issues. This thesis was arduous, but with his help, the path was more straightforward.

Secondly, I thank my friends for always trying to make this process easier, whether it was by reviewing my work, giving me their feedback, or simply keeping me focused.

A special thank you goes to my girlfriend's grandparents, Margarida and António, for showing immeasurable kindness in the final stages of my thesis. Thank you for giving me a space where I could isolate and focus on my work.

Lastly and most importantly, I would like to thank my girlfriend for sticking beside me through thick and thin. These last months have been nothing short of chaos. Thank you for helping me deal with all the anxiety and pressure. Thank you for constantly listening to me mumble about my work. This thesis is also yours.

Abstract

As robots leave highly structured factory environments and enter human-populated areas, autonomous systems should learn how to adapt and cooperate with humans. The key to designing such systems is accurate action prediction. Previous human action prediction research focused predominantly on two approaches: 1) collecting data about a given problem and making the algorithm “learn” its inherent patterns (neural networks); 2) describing the problem’s rules and constraints and letting the algorithm find a sequence of actions that takes us from a given initial state to a goal state (planning). The first approach completely ignores the structure of the problem. Additionally, its heavy data dependency makes this strategy hard to employ when little to no information is available. On the other hand, the second approach struggles to find an action sequence when the problem is too complex. This work proposes a model-based reinforcement learning strategy, combining the previously mentioned approaches. By describing the rules of the environment to the algorithm, it learns what actions the robot agent and the human agent can carry out. It then estimates what option brings the most joint reward to both entities with the help of a policy network. We implemented multiple models to study how knowledge about future possibilities can increase the cooperative behavior of the robot agent. Our work shows that focusing on an agent’s objective may lead to better human-robot collaboration than targeting the human’s next move.

Keywords

Planning, Joint action, Prediction, Planning heuristics

Resumo

À medida que robôs saem de ambientes altamente estruturados e entram em áreas habitadas por humanos, os sistemas autónomos devem aprender a adaptar-se e a cooperar com humanos. A chave para projetar tais sistemas é prever ações com precisão. Pesquisas anteriores focaram-se predominantemente em duas abordagens: 1) recolher dados sobre um determinado problema e fazer o algoritmo perceber os padrões existentes (redes neuronais); 2) descrever as regras do problema e deixar que o algoritmo encontre uma sequência de ações que nos leve de um dado estado inicial a um estado objetivo (planeamento). A primeira abordagem ignora completamente a estrutura do problema. Além disso, a sua forte dependência de dados torna essa estratégia difícil de empregar quando não existe informação disponível. Por outro lado, a segunda abordagem tem dificuldade em encontrar uma solução quando o problema é demasiado complexo. Esta tese propõe uma estratégia de aprendizagem por reforço baseada em modelos, combinando as abordagens mencionadas anteriormente. Ao descrever as regras do ambiente ao algoritmo, ele aprende as ações que o agente robô e o agente humano podem realizar. Em seguida, estima qual opção traz uma maior recompensa conjunta para as entidades com a ajuda de uma "rede de políticas". Implementamos vários modelos para estudar como o conhecimento sobre possibilidades futuras pode melhorar o comportamento cooperativo do agente robô. O nosso trabalho mostra que focar no objetivo pode levar a uma melhor colaboração humano-robô do que prever o próximo movimento do humano.

Palavras Chave

Planeamento, Ação conjunta, Previsão, Heurística de Planeamento

Contents

List of Figures

List of Tables

1

Introduction

Contents

1.1 Motivation

With the growing number of autonomous systems leaving structured environments, they must be able to adapt and collaborate with humans. Collaborative robots (or cobots) are machines specifically designed for close human-robot interaction within a shared physical space. Human-robot collaboration's applications extend throughout multiple areas such as autonomous driving [6], healthcare [7], risk assessment [8, 9], among others.

To achieve collaboration, robots must learn how to act like humans, which in this context, means predicting and adapting to others' behavior.

As people, we rely on verbal communication and, more importantly, non-verbal cues to understand each other's goals and coordinate [10]. If we were to give this "social intuition" to autonomous systems, combined with their precision, they would be able to perform human tasks better than humans.

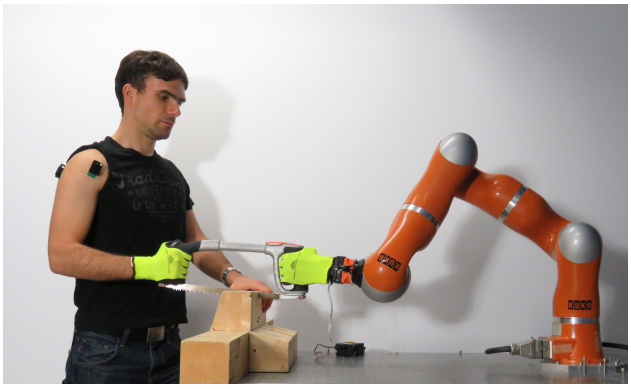


Figure 1.1: Collaborative human-robot sawing experiment [1]

.

In this thesis, we focus on the importance of action prediction as a tool to achieve collaboration. Human behavior is not always predictable, as humans often act according to their individual beliefs and experience. This fact makes it harder for machines to anticipate our actions accurately.

The objective here is to create an artificial agent that can collaborate with humans by understanding human intentions and predicting their future actions. Here, we narrow down the problem to a simple 2D scenario.

1.2 Topic Overview

This section briefly summarizes the concepts explored throughout the thesis. In later chapters, we will go more in-depth into each one of them.

1.2.1 Collaboration

Collaboration refers to an activity where two or more individuals work together in a cooperative way to achieve a common goal.

According to Sebanz, cooperation is the combination of three elements: having a shared perception of the objective and the means to achieve it, anticipating what action our counterpart will take, and acting according to our prediction of our partner's action [11].

Anticipating the actions of others requires us to perceive what is going on in their minds. Research shows that humans can understand mental states, i.e., comprehend that each person has their intentions and beliefs, and they can differ from everyone else's [12].

With the help of information available in the environment, we can infer the mental state behind a given action. For example, seeing someone looking at a store's window might indicate an interest in buying something.

The Collaboration chapter explains in greater detail the components introduced by Sebanz, as well as this mechanism that allows us to form representations of the outer world in our minds. Furthermore, it also discusses the implementation of said mechanism in robots.

1.2.2 Prediction

The previous section highlighted how humans predict each other's actions. Now the Prediction chapter focuses on how Artificial Neural Networks use data about a specific problem to model it. This approach ignores the structure of the environment and makes predictions about future outcomes based on previous examples.

Neural networks are composed of several artificial neurons connected through weighted edges. We give several samples of problem data to this network, and it learns its patterns achieving an accurate prediction model.

In the context of human action prediction, common neural networks give way to more sophisticated models.

Convolutional Neural Networks are extremely convenient for image recognition. They pose as a tool for robots to understand visual information about the environment. Architectures of this model usually do one of two things: identify the action in an image or video or predict the next frame in a video.

The next model is the Recurrent Neural Network which can process temporal information, i.e., recognize patterns that repeat in time. Its architecture differs from the simple neural network and the Convolutional Neural Network.

Last but not least, we present Reinforcement Learning which is related to forcing the algorithm to interact with the environment, making it learn through experience. This approach solves the issue of collecting large amounts of data to create accurate models.

1.2.3 Action

The previous section briefly described how we use data to create predictive models. The Action chapter focuses on the structure of the environment, i.e., the existing relationships and constraints in the problem domain.

Artificial Intelligence planning aims to autonomously solve problems where we have an initial state and want to reach a goal state by performing a set of actions. This approach sequentially evaluates which actions are possible and moves toward the objective.

Classical planning problems are solved with the help of a variety of algorithms. Some are simple techniques that explore all possibilities, while others use knowledge about the problem to narrow the search and make it more efficient.

Some models combine learning from data with the environment structure. We call this approach model-based reinforcement learning. This strategy has been successfully implemented before, mainly in game-playing scenarios.

1.2.4 Proposed model

The Proposed model chapter discusses how we combine the previous models and theoretical concepts to create an artificial intelligence agent that collaborates with humans.

We start by defining our problem as a sequential decision-making scenario, where we know the set of actions we can take at a given state, but we don't know the reward of performing either. To visually analyze the process of action prediction, we decided to use a 2D grid world.

Throughout this chapter, we present several models we used to study collaboration. The first ones focus on predicting the action of the human agent and taking an anticipative action. The last ones try to interpret the task in terms of what sub-goals compose the main goal. Instead of predicting the next move, these models understand the human agent's objective. We show that they achieve better levels of cooperation.

1.3 Objective

Science has looked into the problem of human action prediction through multiple approaches. As previously mentioned, neural networks receive data and create a prediction model for the problem. The second approach, planning, involves computing the actions that will take us to our objective.

Each of these approaches has its advantages and disadvantages. For instance, we might not have sufficient data about the problem to train the network, which is very common in real-world applications. On the other hand, planning problems struggle when the problem is too complex and fail to compute the path to reach the goal.

The objective of this thesis is to explore how we can combine the structural nature of planning problems with the pattern recognition ability of neural networks to create a human action prediction model.

2

Collaboration

Contents

2.1 Collaboration

As technology advances, robots are becoming more than simple order followers. Automated machines are progressively leaving highly structured environments like factories and moving into human-populated areas. They do not only have to operate efficiently and securely when alone but also be able to cooperate (same as collaborating) with humans.

This problem introduces the research field of Human-Robot Collaboration, which studies the collaborative processes that allow human and robot agents to work together and achieve shared goals.

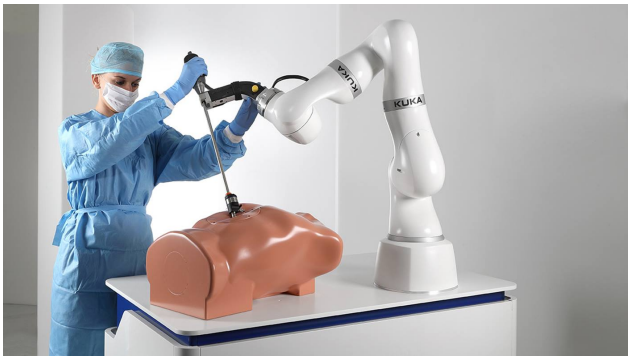


Figure 2.1: Human-Robot collaboration in the medical sector

To understand how we can program robots to collaborate with humans, we must first understand how humans can collaborate among themselves.

Research shows that signs of the ability to coordinate with others are present in humans from an early age. Through interaction with other people, children become capable of engaging in more complex joint actions [13].

In this section we explain what collaboration entails.

Sebanz, Natalie, et al. define cooperation as the interaction of three components: shared representation, action prediction, and anticipative action [11]. Shared representation relates to establishing a common goal and structure for the joint action. Action prediction is related to predicting the co-actors' next move. Anticipative action, in turn, involves acting according to the goal and our predictions of the other agents' behaviors.

The following sub-sections detail each component in greater depth.

2.1.1 Shared Representation

Shared representation is more than sharing the same goal. It requires co-actors to have a common conceptual background. This is important because multiple individuals may adopt different strategies to achieve the same objective.

A simple illustration is two people attempting to prepare a meal together. Not only do they have to agree on the dish they are going to cook, but also on the recipe they will use. To put it in more abstract terms, partners must agree on their goal and the small steps necessary to reach it.

Joint attention is another critical aspect of shared representation. It involves processing information about the attention of self and others [14]. This ability allows us to 1) understand where others

are focusing their attention; 2) orient our co-actors' attention to a specific location [15, 16].

The final part of shared representation is the environment portrayal. Partners need to have the same views on how the shared workplace is currently functioning.

Returning to the example of two people preparing a meal, to cooperate, both must be aware of the ingredients that are available and where they are, as well as the kitchen equipment that they can use (the oven, for instance, may already be in use).

Note that two people can cook together without having the same information on the environment, e.g., helping a friend cook at his house. However, in this case, we don't classify the interaction as "collaboration" since one agent would have to provide instructions to the other.

In short, shared representation involves having the same goal, strategy, and knowledge of the surroundings.

2.1.2 Action Prediction

The ability to anticipate others' actions is closely related to collaboration.

Action anticipation refers to predicting our partner's possible next moves using non-verbal cues. Given our knowledge about the goal, we can infer what subgoals the other agent is trying to finish.

Research has shown that human brain activity is similar when we observe someone acting and performing that action ourselves [17]. The causes for this activity are mirror neurons.

Additionally, Kohler E. et al. prove that these mirror neurons also discharge when hearing the sound related to an action. For example, if we listen to the sound of keys hitting a door, some neurons will activate as if we were the ones opening the door [18].

The most accepted theory is that "mirror neurons" help humans understand the goal of observed actions [19].

The suggested mechanism states that individuals know the effect of their actions. Furthermore, as previously described, observing the actions of others activates one's neural system. Thus, when the "mirror neurons" that symbolize an act trigger by looking at another individual, the observer is capable of understanding its goal. That happens because the neural discharge is the same as when the observer tries to achieve the same objective [20].

Prinz interprets this "mirror response" as a form of action simulation [21]. This theory suggests that when observing an agent acting, we tend to perform the same action.

Other authors, like Sebanz, divide prediction into the sub-elements, "what", "when", and "where" of an action [22].

The "what" aspect refers to the type of action the agent is performing and the intention behind it. Perceiving the motives of others falls under the "Theory of mind" mechanism, which we will describe later. For now, we will simply accept that there is plenty of evidence suggesting that humans perceive the actions of others in terms of their goals [23].

For instance, if we were watching climbing up the stairs, we would easily understand the aim of that action and ignore how the agent was performing it (e.g., one step at a time or two steps at a time).

The “when” aspect is critical for joint actions that require close temporal coordination. The main challenge here is predicting the timing of others’ actions. A good example of timing coordination is a band playing together.

As a curiosity, we introduce the theories of Davidson and Wolpert. According to them, humans generate timed sensory predictions of their actions [24], which allow us to filter sensory information. If the outcome matches the prediction, “sensory cancellation” occurs [25]. This theory explains that we cannot tickle ourselves since we can accurately predict the timing of the action.

The “where” of a prediction focuses on the physical environment shared by the co-actors. We highlight two main aspects, predicting the spatial implications of the actions of others, and shared attention.

Working in a shared space requires the agents to be aware of the location of others, e.g., not bumping into other cooks while working in a kitchen.

A similar challenge is predicting where objects will end once the other has set them in motion with their actions.

Knoblick and Flach conducted a study where participants watched clips of themselves or somebody else throwing darts at a board and tried to predict the landing location [26]. They verified that the predictions were more accurate when people watched themselves acting, which supports the idea of action simulation.

The other component of the “where”, shared attention, is concerned with joint action requiring agents to attend to the same object. For example, when two people move a sofa together.

There is a universal consensus that shared attention is more than simple gaze following [22]. It involves agents being aware they are paying attention to an equal location.

Furthermore, partners must comprehend that the relationship between one and the object is identical to that of the other.

In summary, action prediction consists of assessing what the next possible moves are. With that in mind, we then predict which possibility is more likely with the help of non-verbal cues. Finally, we anticipate “where” and “when” the agent executes the action.

2.1.3 Anticipative action

The final element of collaboration is anticipative (or anticipatory) action. It is concerned with choosing the move that will bring the team closer to the goal.

By using knowledge about the environment and other agents, we can adjust our actions to ensure the best possible outcome. In other words, shared representation and action prediction provide the information we can use to make an educated decision. The question now becomes how we can give this intuition to robots.

Collaboration can be seen as the interplay between different parts. Creating a representation of the environment is commonly achieved by collecting data about the physical space, e.g., using cameras.

In order to predict an agent's action we need to comprehend what is guiding it. The difficulty here is understanding how automated machines can "assign" a goal to an agent.

The philosopher Daniel Dennet introduced the concept of intentionality, a core component of the cognitive science field. Dennet states that actions are goal-directed and derive from unique beliefs and desires [27]. This collection of psychological features is what we call a mental state.

The following section will outline the theory of mind mechanism, which is responsible for assigning mental states.

2.2 Theory of Mind

The Theory of Mind mechanism (ToMM) is an essential field of research in understanding and reasoning about goal-directed entities. This mechanism refers to the ability underlying the capacity to reason about one's own and others' mental states. Possessing a functional theory of mind is crucial for successful everyday social interactions, as it allows us to make sense of the actions of others and communicate efficiently.

If we were able to design systems that are capable of understanding the intents of others and attributing mental states to them, i.e., desires, thoughts, or intentions, we would achieve better human-robot cooperation.

In order to understand how humans come to develop a theory of mind, authors like Simon Baron-Cohen and Alan M. Leslie have focused on the study of children on the autism spectrum.[28, 29]

Previous studies had suggested that these children were unable to comprehend other people's mental state. The false belief test clearly illustrates this idea. In this test, a child observes the scenario presented in figure 2.2.

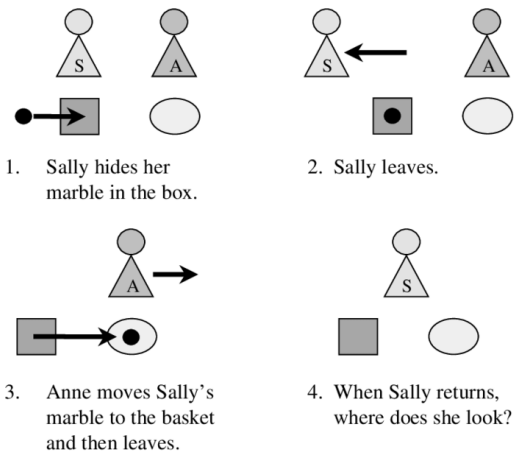


Figure 2.2: False Belief test from Brooks, Rodney et al.[2]

The experiment demonstrated that children around a certain age were able to understand that when Sally returned she would look for the marble in the box, and therefore hold a false belief. However, children on the autism spectrum around the same age thought Sally would look for the marble in the basket, since they could not comprehend that other people have their own beliefs (mental states).

This experience raises the question: How limited is the perception that children on the autism spectrum have? By answering this question, we hope to explain theory of mind a bit further.

2.2.1 Primary Representation vs Metarepresentation

In the previous section, we introduced the concept of theory of mind and that children on the autism spectrum show evident flaws in this mechanism, which manifest as a limited perception of their surroundings. In this section we focus on what perception entails.

Alan M. Leslie proposes a distinction in the development of the normal functioning cognitive system, between primary representation and metarepresentation [30]. Primary representations allow humans to understand and store literal information about the world. They refer to simple predicates like “There is a snake in the bush”. Metarepresentations, in contrast, allow our cognitive system to create descriptions of other representations, hence their name, e.g.,“Bob thinks there is a snake in the bush”. According to Leslie, metarepresentation is the mechanism that enables humans to represent mental states.

2.2.2 Metarepresentation’s role in joint attention

Now that we have defined metarepresentation, we can discuss its significance.

When it comes to children on the autism spectrum, research has shown that their primary representation mechanism is intact, but the metarepresentation one is not. This conclusion comes from observations where joint attention activities (e.g., pointing at an object to get someone to hand it to you) are considerably less common in children on the autism spectrum than in control groups [31]. Children on the autism spectrum can’t understand joint-attention behaviors initiated by others and are incapable of conducting such acts themselves.

Does this mean that joint attention behaviors require metarepresentation? Some authors seem to agree that the answer is true [28, 32]. The consensus is that collaboration requires communication, whether it is verbal or non-verbal (e.g., gaze or pointing). Communication (i.e., each person understanding the intents of the other) requires each individual to have the theory of mind mechanism. Theory of mind needs metarepresentation. Therefore, we can conclude that joint attention requires metarepresentation.

However, other scholars claim that joint attention skills develop before the cognitive capacity for metarepresentation [33].

Simon Baron-Cohen argues that these early manifestations of joint attention are a symptom of having metarepresentation. [34].

2.2.3 Non-verbal communication

The last sections explained what representations are and how crucial they can be for understanding attention in others.

Primary representations allow us to say things like “John is looking at the cake”. Metarepresentations enable us to think critically and add, e.g., “John is looking at the cake, so he must want a

slice".

Thus far, we have omitted the role of communication in joint actions. Humans may use verbal conversation to agree on the objective and strategy they will employ. However, the majority of communication will be at a non-verbal level. Giving small signals to our co-actor while performing a joint action is part of human nature.

In this section, we focus on the relevance of non-verbal communication for social interactions.

Non-verbal cues can take many forms, e.g., facial expressions, gestures, paralinguistics, and more.

The eye gaze, in particular, plays a central role. Humans use actions like staring and blinking to transmit a ton of information.

Eyes are a unique tool. While ears allow us to hear and lips enable us to speak, eyes can both signal and perceive [35, 36].

Signaling and perceiving are always hand in hand. If we imagine a social context where one person signals something, others perceive it (figure 2.3).

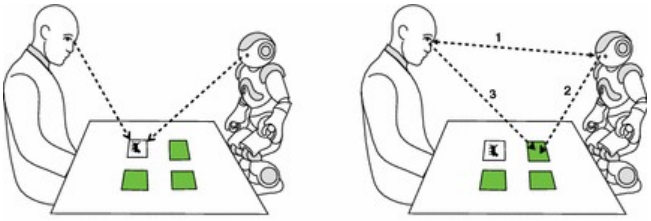


Figure 2.3: Directing attention through gaze hints

Gaze studies in robotics target the design of architectures of gaze behavior for robots [37]. With a solid structure, we can make machines more predictable and communication-prone. To prove the dual function of eye gaze, Wu, David W. L. et al. studied a natural food consumption situation [38]. They hypothesized that people tend to look away when someone begins to bite. They observed that participants were significantly more likely to look down at their food before taking a bite when eating with another person than when eating alone. This behavior shows an unconscious tendency to signal our actions. Furthermore, they also noticed that when one person looked down, the other was more inclined to look away. This behavior indicates that people can read this type of cues unconsciously.

This study noticeably proves the existence of "eye communication". However, it does not study the connection between vision and mental state attribution in great detail.

Huang, Chien-Ming, et al. directed their efforts to examine gaze patterns to predict intentions [39]. Their exercise considers a sandwich-making scenario where a customer requests ingredients, and the worker adds them to the sandwich. They verified that, generally, the customer would look at an item before asking for it. The results lead us to the conclusion that eye gaze manifests an intention. This idea extends to other forms of non-verbal cues.

In this thesis, we focus on information like distance to objects or direction of movement, which are more suitable for the spatial nature of the problems we intend to solve.

2.2.4 Theory of mind for Robots

So far, we have looked at how the theory of mind mechanism helps people assign mental states to each other.

If we were building machines that interact naturally with people, they must interpret both the environment and the animate agents. They must also display social cues so that co-actors can predict their moves.

To implement the theory of mind mechanism in robots, we consider Baron-Cohen's model and its implications (2.4).

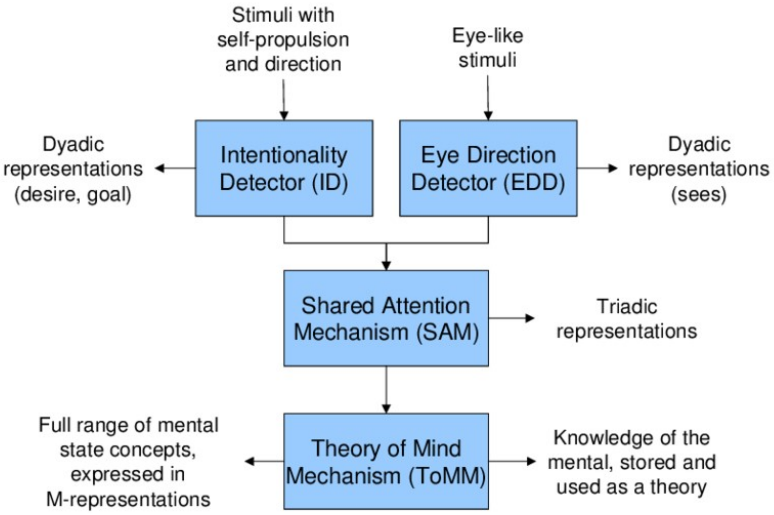


Figure 2.4: Block diagram of Baron-Cohen's model of the development of theory of mind [3].

This model considers two forms of input.

Firstly, we have the raw perceptions that come from our senses (e.g., vision, hearing, touch).

The intentionality detector (ID) filters this data and leaves only self-launched actions, i.e., actions that have an intention behind them. These motions allow it to distinguish between animate (agents) and inanimate (objects) entities [40]. The ID module then generates representations of the partner's intentions, e.g., while looking at someone lifting a sofa, "He is doing it alone" or "He is waiting for something".

The second set of inputs are visual stimuli. The eye direction detector (EDD) builds representations specifying if the gaze of another agent is concentrating on you. This module then identifies if you are the target of that agent's attention [41].

The third module, the shared attention mechanism (SAM), combines the representations of ID and EDD to create more complex portrayals. Imagine John is looking at us while he tries to lift the sofa. We build descriptions such as "John knows that I see him looking at me" or something along those lines.

The fourth and last module, the theory of mind mechanism (ToMM), is what transforms SAM representations into mental states [42]. For example, by combining "John is trying to lift the couch",

“John is looking at me”, and “John knows that I see him looking at me”, we can get “Jonh wants my help lifting the couch”.

Now that we have described this theory of mind model, we discuss its practical implementation.

An automated system programmed with this mechanism would be able to perform high-level collaboration with humans.

The robot would learn social cues from humans, just like a child does, and use them to assign mental states. It would also learn to comprehend and express its intentions (or mental states in general).

The difficulty in implementing this model is that every module requires analyzing several complex processes.

There are numerous works in mental state attribution and gaze following that provide frameworks for the modules presented above. However, we won’t look at them in depth as they are outside the scope of this thesis.

Nonetheless, the model described in this section highlights some valuable ideas for modulating human behavior. This dissertation utilizes them in an algorithm to predict actions.

2.3 Conclusion

In this chapter, we summarized the theoretical foundations behind human collaboration.

We have seen that cooperative interactions combine three factors: shared representation, action prediction, and anticipative action.

We then described the mechanisms that allow humans to interpret the behavior of themselves and others. A considerable amount of research in this field focuses on autism. Robots, in a way, can be seen as “autistic” since they cannot understand people’s intentions, emotions, or beliefs.

Finally, we presented Baron-Cohen’s model for the theory of mind mechanism and discussed the implications of its implementation in automated machines.

In the following chapters, we describe techniques that achieve collaboration. One group of methods solely focuses on predicting the next move, while the other tries to understand the steps that compose each action.

3

Prediction

Contents

3.1 Action prediction

As mentioned in the previous chapter, one of the approaches for predicting actions heavily focuses on the prediction itself. Prediction, in this context, means the ability to anticipate an observed act [43].

Focusing on prediction, as opposed to focusing on the goal, suggests that the predictive system does not know the global structure of the environment. Instead, the algorithm analyzes samples, i.e., data about previous iterations of the problem, and gives a prediction.

This corresponds to, e.g., choosing the best move in a chess game without knowing the rules but having seen the game multiple times before.

The concept of playing chess without knowing the rules previously may seem empirically wrong. However, several authors applied it, achieving convincing results. [44, 45]

These approaches save us the trouble of formulating the environment's rules, which implies we can reuse them in several scenarios with relatively little effort.

Getting predictions by studying data introduces the machine-learning field of artificial neural networks (ANNs or NNs).

3.2 Artificial Neural Networks

Neural networks (same as artificial neural networks) are algorithms inspired by the neurons in the biological brain. We'll quickly go over their architecture.

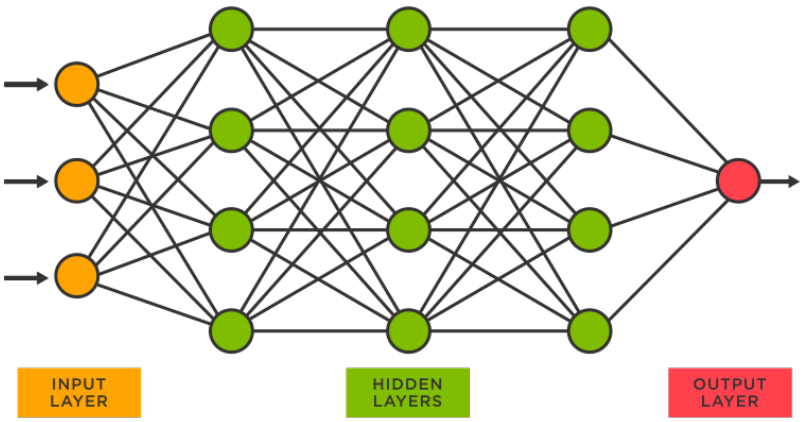


Figure 3.1: Example of artificial neural network structure

NNs are composed of several "artificial neurons" with connections between them and divided into three layer types: input layer, hidden layer, and output layer. (figure 3.1). In the figure, each node is a neuron.

The input layer receives the input data and forwards it to the hidden layers. The hidden layers are intermediate layers between the input and output layers and are the ones that perform all the computation. The number of hidden layers may vary from network to network. Additionally, networks with several hidden layers can be called deep neural networks (DNNs). Finally, the output layer receives information from the last hidden layer and produces a result (prediction).

As we stated before, layers are composed of several artificial neurons. Neurons connect to neurons in the adjacent layers through weighted edges. This way, neurons serve as inputs to neurons in the next layer. The weight, w_{kx}, represents the connection's strength, i.e., how strongly one neuron affects the other.

Each neuron can be activated through the edges. The activation of a neuron depends on all the neurons in the previous layer that connect it. More specifically, the sum of all input contributions (plus a bias, b_x) goes through an activation function that decides whether the neuron is activated or not.

The neurons' layout is summarized in figure 3.2

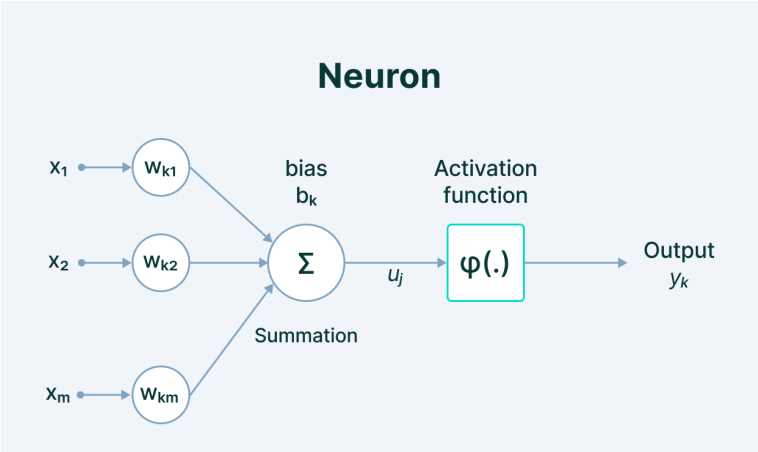


Figure 3.2: Artificial neuron's layout

So far, we have defined the basic architecture of a neural network. Now we describe how it "learns" to generate predictions from data.

Consider we were trying to predict tomorrow's weather based on today.

The learning happens in a process called training. In this process, we take data samples from our problem and input them into the network. In the example, the data samples would consist of measures from a given day and the observed weather the following day.

The inputted information propagates throughout the nodes in the network (as shown in figure 3.2) until the output layer.

The prediction would most likely be wrong since we randomly initialize all the weights. However, the algorithm compares the value of the output in the sample (correct value) to the predicted one. It uses a lost function to evaluate how wrong the prediction was. Then, it calculates how the weights of the edges must change to make the prediction slightly more accurate.

The neural network uses this process (back-propagation) and updates all the relevant weights for that input.

The algorithm repeats these steps for each sample on the training set and leaves us with a trained network. Then we use a testing (or validating) set to check if the network is able to accurately predict the outputs given the inputs.

This section introduced neural networks as a tool for generating predictions. In the context of action prediction, the commonly used models have slightly different architectures.

3.3 Convolutional Neural Networks (CNNs)

The first model we look at is the convolutional neural network (CNN). CNNs are the most commonly applied algorithm in image analysis/recognition.

Their success comes from the ability to identify patterns (or features) in images.

For example, if we wanted to classify a dataset of animal photos, the network would be able to recognize characteristics like tails or beaks. The output layer would then classify the species with some degree of certainty.

As we have said in the previous section, CNNs are also used in the action prediction field. They utilize images of agents performing actions instead of measures of some kind (like in the weather prediction example).

For starters, we are going to highlight the main components of convolutional neural networks. Then, we will overview some of the literature concerning CNNs applied to action prediction.

3.3.1 Architecture

The inputs are the first difference between a CNN and a regular artificial neural network. While simple models of ANNs receive information like the temperature in a day, convolutional ones process complete images.

Imagine our network takes 10x10 images as input. The flattening operation converts these dimensions (height and width) into a single one. This way, the network interprets the 10x10 image as a vector of size 100[1].

Consider a 1920x1080 (1080p) image. It is composed of a little bit over 2 million pixels. If we connect each neuron in the input layer to all the neurons in the second layer, we obtain around 4 trillion connections. Note that our network may have many more layers that add more complexity. So as we show, it is computationally unfeasible to train this number of edges.

Now, we are going to describe the layers that make convolution on images feasible.

The first big innovation is the introduction of convolutional layers.These layers are the ones responsible for highlighting features. They get their name from the convolution operation they perform. Consider the 2D input matrix representing an image in figure 3.3.

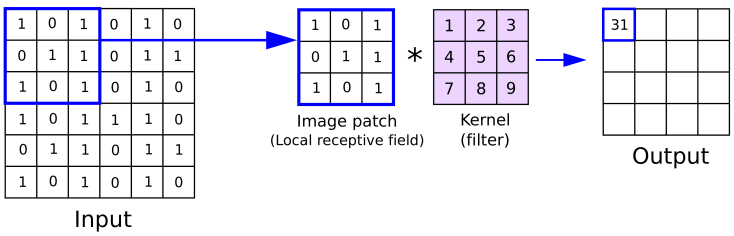


Figure 3.3: Convolution layer - Applying a kernel [4]

[1]We feed each pixel to a neuron in the input layer, so if the network receives 10x10 images, it means that the NN has 100 neurons in the input layer

By sliding the kernel across the image, we obtain the convolved feature (output). The values of the convolved feature matrix are the sum of elementwise multiplications between the image patch and the kernel. Considering the kernel at the location presented in the figure, we calculate:

$$1 * 1 + 0 * 2 + 1 * 3 + 0 * 4 + 1 * 5 + 1 * 6 + 1 * 7 + 0 * 8 + 1 * 9 = 31. \tag{3.1}$$

CNNs learn how kernels can identify specific features through the learning process previously described. Each kernel detects a distinct characteristic.

The first convolutional layers will capture low-level details like edges and corners. As we progress on the network, we get more high-leveled features, e.g., if the image contains a dog.

The second novelty is the pooling layers. Usually, these layers always follow a convolutional layer.

Their purpose is to reduce the size of the convolved feature. This way, they reduce the computational power required for processing the data.

Additionally, they also extract dominant features formerly highlighted by the kernels.

The most common operation performed by pooling layers is “max pooling”. This operation calculates the maximum value in a convolved feature patch.

We get the output by sliding a matrix through the convolved feature, similar to what we do in the convolutional layer.

Besides reducing computational power, Max Pooling layers also have the advantage of being noise suppressants, meaning that they discard randomly activated neurons.

Since CNNs are not the direct focus of this thesis, we gave a general perception of the processes they use while trying not to get too technical.

We now analyze the application of convolutional neural networks in action prediction.

3.3.2 Applications

Computer vision is an artificial intelligence field that teaches machines to comprehend visual information. In the context of understanding human action, this field utilizes CNNs in one of two ways.

On the one hand, action recognition concerns processing an action within a video and generating a label. On the other hand, action prediction or anticipation is related to recognizing an act as early as possible, i.e., predicting a human action from a temporally incomplete video.

Due to the growing real-world applications, these problems have become more popular [46]. In this section, we will describe relevant research in each field.

Security surveillance is a frequent application of action recognition. Areas under surveillance often prohibit some human actions, e.g., jumping over a fence.

Ji et al. developed a 3D CNN model that extracts spatial and temporal features by applying 3D convolutions [47]. This way, the model captures motion information in sequential frames and labels the video.

Duan et al. study the use of “skeleton-based” approaches in action recognition. Their model estimates the pose in a 2D human image. Then, they stack heatmaps of joints or limbs along several frames, creating 3D heatmaps. Finally, a 3D-CNN classifies these 3D heatmaps.

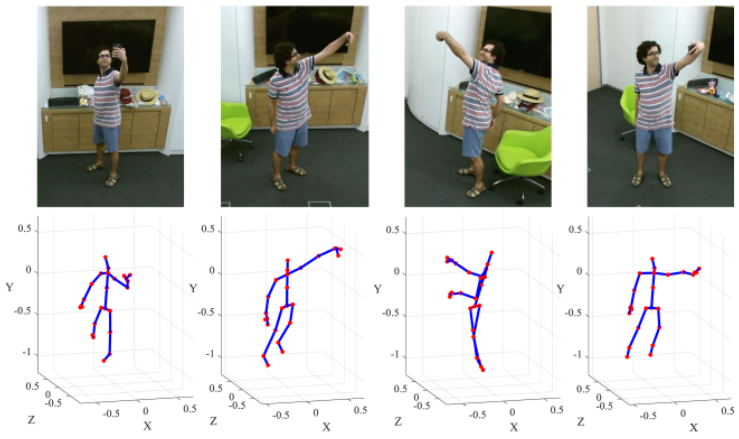


Figure 3.4: Skeleton representations in human action frames

Gkioxari et al. introduce the idea of using cues in an image to classify the action [48]. They argue that contextual information may help improve the performance of action recognition models. For example, when observing a jogger, the scenario (e.g., the road or trail) and the presence of other joggers can be an additional source of information. Their model outperforms previous approaches in the field.

Due to significant improvements in computer vision and machine learning, image analysis problems have progressed from inferring the current state to predicting the future state.

Correctly classifying an act, watching only its initial frames, is crucial in applications like autonomous driving. By analyzing the body motion and predicting the future actions of pedestrians, CNNs can prevent collisions.

Kong et al. propose a model for predicting actions in videos containing incomplete acts [49]. Their approach uses sequential context information to complement the extracted features of partial videos. It reconstructs missing info by learning from fully observed videos of the corresponding action.

Due to the temporal nature of the movement, models that combine spatial and temporal features demonstrated better results than models that only consider one.

Nonetheless, using CNNs alone has proven to have its limitations. These networks are considerably convenient when it comes to raw image classification. However, to identify patterns in sequential data, such as actions, there is another class of neural networks that stands out.

3.4 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are a class of neural networks that allow previous outputs to be used as inputs. The main difference between a CNN and an RNN is the ability to process temporal information.

This capacity makes them very appropriate in the fields of speech recognition and natural language processing. Data in these fields come in sequences, e.g., a sentence. As described in the previous section, we can represent an action as a collection of sequential frames. In the case of speech recognition, the problem might be predicting the succeeding words in a sentence, whereas in

action prediction, we can use RNNs to anticipate the next set of frames.

3.4.1 Architecture

The ability to recognize patterns in sequential data and predict the most likely output comes from their architecture. Utilizing previous results to elaborate the current prediction makes RNN cells differ from simple neural network cells 3.5.

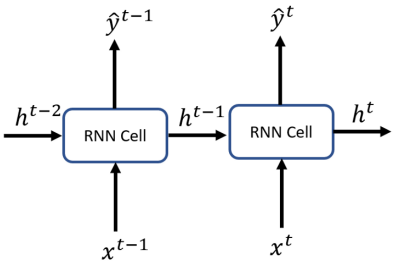


Figure 3.5: RNN cell sequence

RNN cells receive the input vector x^t, where t is the time, and projects it to an output vector $\hat{y}^t$. As we can observe in figure 3.5, to make a prediction, the cells also consider what is called the network hidden state, h_{t-1} (hidden state in the previous timestep), which depends on the previous outputs of the network.

At each timestep, the RNN calculates the output vector and the hidden state as a some function of the input and the previous hidden state. It then propagates the current hidden state to the next cell.

In short, an RNN cell is defined by an expression for the output at the current timestep, $\hat{y}^t$, and the next internal state, h_t (equation 3.2). Here W_x, U_x, and b_x are model parameters.

$$\begin{aligned} h_t &= softmax^2(W_h h_{t-1} + U_h x_t + b_h) \\ \hat{y}_t &= softmax(W_{\hat{y}} h_t + U_{\hat{y}} x_t + b_y^t) \end{aligned} \tag{3.2}$$

The process of training an RNN is similar to the one in simple neural networks. Information is fed forward in the network until it reaches the output layer. Then, by computing the lost function the model understand how it should update the values of its parameters.

The shortcoming of RNNs is they cannot remember long-term dependencies in data due to the vanishing gradient problem. The gradient refers to the update the networks parameters get (in the training phase), which is proportional to how much that parameter affects the lost function. In some cases, this value is very small, which means that as we back-propagate it, some gradients become increasingly small. The consequence is some parameters not updating, which in the worst case stops the network from training.

3.4.2 Long Short-Term Memory

To solve this, Sepp Hochreiter and Jurgen Schmidhuber invented Long Short-Term Memory (LSTM) [50]. RNNs built with LSTM cells divide data into short-term and long-term categories. RNNs choose

[2]The softmax function is used as an activation function in multiple neural network layouts.

whether data is valuable enough to remember and loop back into the network or if it is irrelevant.

LSTM cells consist of three parts, also known as gates: the forget gate, the input gate, and the output gate (figure 3.6).

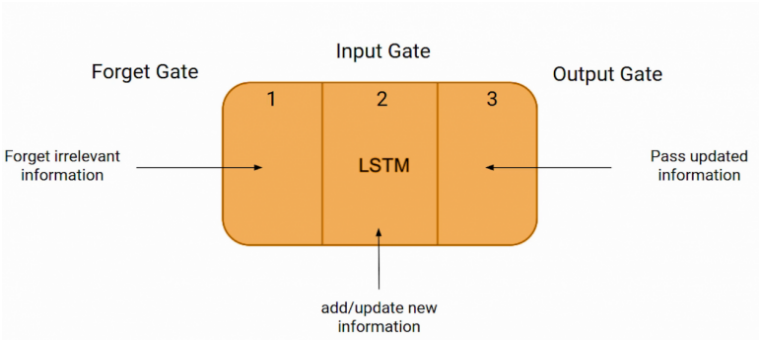


Figure 3.6: Long short-term memory cell [5]

The forget gate decides if the cell should store the information from the previous timestep or ignore it. Consider the following example: “Alice likes the beach. Bob likes the pool”. The first sentence talks about Alice and ends with a fullstop. After detecting Bob’s name, the network should forget about Alice.

The input gate evaluates the importance of the input information and decides if it should be stored. Now consider the example: “ Bob is going to the football game. He texted me saying he knows the owner of the stadium.” Both sentences refer to Bob. However, in the second sentence, the relevant information is Bob knows the owner of the stadium. Ideally, the network should forget that “He texted me”.

The final gate is the output gate which determines the value of the next hidden state. Imagine we want to predict the next word in the “Bob got a sunburn, but he applied sunscreen. Poor _____”. In this example, the network should understand that “poor” is related to Bob and update the hidden state accordingly.

By regulating the flow of information into and out of the cell, the three gates allow the RNN to have a short-term memory that can last several time steps, thus the name long short-term memory.

3.4.3 Applications

Recurrent Neural Networks are one of the state of the art approaches in machine learning. However, its applications remain limited due to the difficulties in training them. In sequential data with complex patterns, the number of parameters in the model can surpass the millions. Nonetheless, there are still some applications that deserve mention.

Graves et al. combined a multidemensional LSTM with temporal classification to create a handwriting recogniser [51]. They were able to successfully apply their system to English and Arabic. Additionally, they anticipate that this model can be used for any supervised sequence labelling problem.

Sutskever et al. tried using RNNs to model language at the character level [52]. Previous studies

showed that morphemes[3] were the appropriate units for speech prediction. They argue that converting words into morphemes is more troublesome than treating them as groups of characters. With this approach, their model can adapt to words it has never seen (provided they are a composition of already-known morphemes). Additionally, predicting the next word by making a sequence of character predictions is much more efficient than computing the probabilities in a dictionary of known words.

In another area, Seker et al. showed that it is possible to use RNNs for condition monitoring and diagnosis in nuclear power plant systems [53]. By comprehending specific patterns in data, the network can detect anomalies. i.e., results that don't correspond to the prediction. The conclusions of this study make us speculate about utilizing similar techniques to detect human errors.

Predicting stock prices is another application of RNNs. Share values are complex functions that depend on several factors. Minami Shotaro proposes an LSTM-RNN method for predicting a single stock price using corporate action event information and Macro-Economic indices [54], which shows promising results.

In the context of action prediction, the most well-known example is autonomous driving. The challenges in this area include predicting the actions of drivers and pedestrians. Although the proposed architecture was presented in the context of pedestrian crossing prediction, the authors speculate that other applications of similar nature, such as activity recognition, may also benefit from using this approach.

Rasoule et al. introduce a model that collects information from various sources to accomplish pedestrian prediction at the point of crossing [55]. At each time step, the model observes the pedestrian and its surroundings, which allows it to perceive its pose and velocity. They prove that the algorithm outperforms previous architectures, which used motion history to predict future pedestrian trajectories.

Accurate driver action prediction can improve driver assistance systems. Early recognition is critical in this scenario, as it can help detect driver mistakes and prevent accidents. Olabiyi et al. present a model that captures information about the driver and the driving environment with the assistance of cameras [56]. They then use RNNs to determine the correlation between the inputs and the driver's behavior. Their algorithm can predict actions like accelerating, braking, and lane changing a considerable time before it happens.

With this section, we hope to have clarified how RNNs can help tackle problems in different areas. Previous information about an environment can help machines understand connections between processes. However, this type of approach has its disadvantages.

3.4.4 Limitations

The first disadvantage of Neural Networks we present is heavy data dependency. As described throughout this chapter, NNs use considerable amounts of data in the training process. Generally speaking, more data leads to more accurate predictions. This principle prevents us from using NNs

[3] A morpheme is the smallest linguistic part of a word that can have a meaning. For example, the morphemes "un-", "break", and "-able" make up the world "unbreakable"

in environments where little to no information is available.

The second limitation is overfitting. This concept refers to when a statistical model fits exactly the training data.

While training NNs, we want the algorithm to learn patterns in this training set and then accurately perform with unseen data. If the model perfectly adapts to samples, it loses its ability to generalize for new inputs.

Several causes can lead to overfitting, such as a model that is too complex (which means more parameters), a small training set, or the algorithm running for too long.

We can quickly identify overfitting by comparing the loss function of the model on the training set and validation set (3.7). One way to prevent overfitting is early stopping. This method stops training to prevent the model from learning the noise in the data.

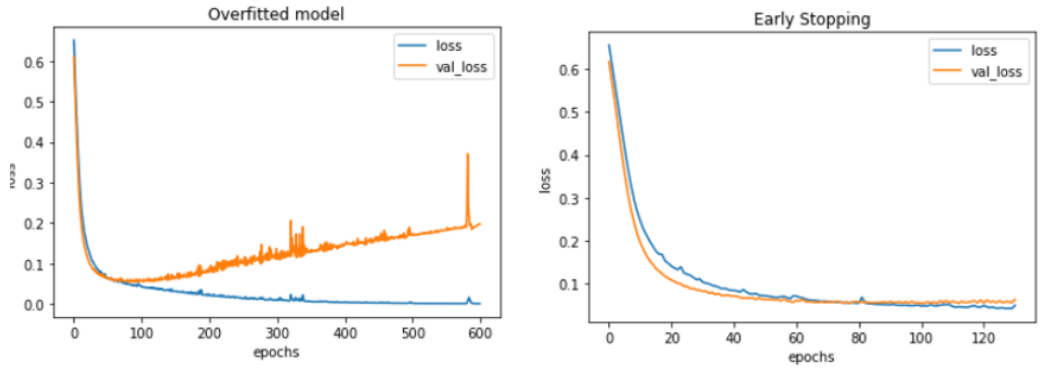


Figure 3.7: Comparison between the loss function of a random model with and without early stop

In figure 3.7, the overfitted model error (loss) is close to zero in the training set (blue line) and much higher in the validation set (orange line). By introducing early stopping, the model generalizes better and has a low loss value for both collections.

This approach is risky since if we stop the training process too soon, the network may underfit, i.e., not learn patterns in data. The goal here is to find a balance between the two.

Following this we introduce a different paradigm, reinforcement learning.

3.5 Reinforcement Learning

Reinforcement Learning (RL) is an area of machine learning that focuses on choosing the action that maximizes a reward. It differs from regular neural networks as it learns from experience, i.e., trial and error, in an interactive environment.

Reinforcement learning is usually modeled as a Markov Decision Process (MDP), which we will better describe later. In this model, the agents interact with the environment through actions. The environment returns a state and a reward for that corresponding action (3.8). Randomly selecting moves and observing the environment's response allows the model to discover what actions are better. This process is equivalent to training in neural networks. The purpose is for the agent to learn

an optimal policy that maximizes the expected cumulative reward of a sequence of actions.

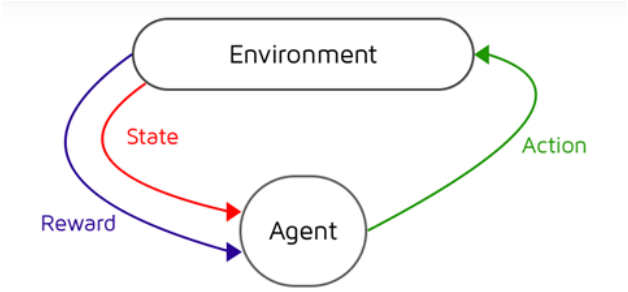


Figure 3.8: Reinforcement learning model

The Monte Carlo method is a way of learning about these states and rewards by interacting with the environment. The principle of this approach is to deal with multiple uncertain variables by generating a large number of random observations and creating a joint distribution for them. This way, the system learns about the expected reward of each action.

This method can be employed in several fields of science. The most common illustrative example is to determine the value of pi (π). Imagine we draw a circle inside a square with a length of 1 on each side, as shown in figure 3.9. We know that the area of the circle is defined by the expression

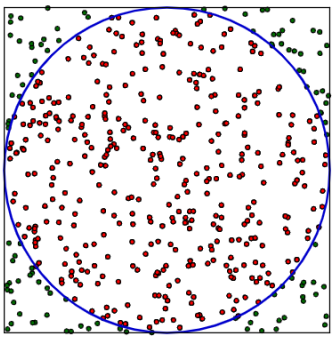


Figure 3.9: Estimating π with Monte Carlo simulations

$Area = \pi r^2$, which we can rewrite as $\pi = 4 * Area$ (since $r = \frac{1}{2}$). Then, we can randomly generate the coordinates and mark them in the figure. The Monte Carlo method tells us that the area of the circle is approximately the fraction between the number of points inside the circumference (392) and the total number of samples (500). By substituting the value of the circle's area, we get the estimate for π:

$$\pi \approx \frac{392}{500} * 4 = 3.136 \tag{3.3}$$

The law of large numbers states that as we increase the number of points, its mean gets closer to the average of the whole population, i.e., as the number of samples goes to infinity, the value of 3.3 approximates the actual value of π.

3.6 Conclusion

In this chapter, we introduced several classes of Neural Networks. All of them have the characteristic of using data to learn patterns.

We started by outlining the basic structure of a NN and the learning process they go through. We then introduced the Convolutional Neural Network (CNN), which is used for image processing. We provided some applications of CNNs for action recognition.

Next, we presented the Recurrent Neural Network (RNN), which finds patterns in sequential information. We described Long Short-Term Memory (LSTM), the mechanism that prevents RNN parameters from not updating.

Then, we presented several applications of RNNs across multiple fields, including an action prediction scenario. Additionally, we also displayed some limitations of neural network approaches.

Following, we introduced Reinforcement Learning which is an approach to sequential decision-making that focuses on learning through experience. The algorithm "plays" with the environment several times, thus discovering which actions lead to the maximum reward.

The Theory of Mind mechanism allows humans to understand the intentions of others. This mechanism also specifies that humans perceive each other's actions in terms of their goals. The following approach focuses on what set of moves takes us close to this goal, considering the environmental restrictions.

4

Action

Contents

4.1 Planning

The previous chapter described a set of approaches that use data samples about the problem to learn how to predict future cases. We now focus on a method that utilizes the rules of the environment to predict future states.

AI planning is a field of Artificial Intelligence (AI) that explores the process of using autonomous techniques to solve planning and scheduling problems [57].

A planning problem is one in which we have some initial state, which we want to transform into the desired goal state by applying a set of actions.

Imagine we wanted to serve a fruit salad. Let's assume our ingredients are an apple, a banana, and a melon. In the kitchen, we have the required fruits, knives, bowls, etc. To assemble the fruit salad, we need to perform small actions like peeling each fruit, cutting them into smaller pieces, and putting everything in a bowl. The problem is simply deciding the order in which we will perform these actions. Clearly, some tasks cannot happen before others, e.g., we cannot put the melon in the bowl before cutting it into smaller portions or cut the banana before peeling it. This example, although very simple, displays some core aspects of planning problems.

AI planning performs something known as a state space search. If we consider the initial state as the root, we can successively apply actions to get a tree of possible states. Figure 4.1 provides a visual example of this search.

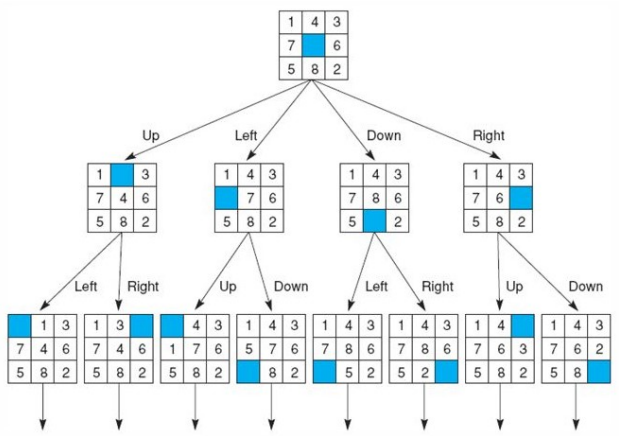


Figure 4.1: Example of state space search in 8-puzzle

The following section outlines the tools used for modeling planning problems.

4.1.1 Planning Domain Definition Language

Planning Domain Definition Language, also known as PDDL, is a group of languages that allows us to define planning problems.

The idea behind PDDL is domain-independent planning, which means that the algorithm we use doesn't need to know anything about the problem it is solving [58]. This ideology enables us to apply the same algorithm in multiple scenarios instead of developing domain-specific algorithms. The difficulty in this type of problem becomes describing the domain.

PDDL domains mainly consist of predicates and actions. A predicate is a function that tests for some condition involving its arguments and returns true or false, e.g., “Is Bob holding a spoon?”.

Actions, in PDDL, are manipulations of the predicates. In its basic form, an action has a parameter and an effect. In this thesis, we will also consider that actions have preconditions.

Let’s look at the example “Bob puts the spoon down”. To put down the spoon, Bob must be holding it in the first place. Additionally, by putting down the spoon, Bob will no longer carry it. Although this may appear overly simplistic, it accurately captures how actions modify predicates.

Action	Parameters	Preconditions	Effects
put down	person, object	person holding object? = True	person holding object? = False
pick up	person, object	person holding object? = False	person holding object? = True

Table 4.1: Behaviour of actions in PDDL

The examples in table 4.1 show another layer of abstraction we can add to our domain. We replaced “Bob” and “spoon” with vaguer terms like “person” and “object”, respectively. Of course, this implies the need to declare “Bob is a person” and “Spoon is an object”. The reason for this change is we might not want actions to be confined to a single person or object. This way, we can introduce new information to our domain simply by affirming, e.g., “Alice is a person” or “Fork is an object”.

Newer versions of PDDL introduce helpful features. Timed initial literals (available in PDDL 2.2 [58]) allow us to express that a fact is “true” only after some time has passed, e.g., if you put food in the oven, it will only be cooked after a while.

Soft constraints (available in PDDL 3.0 [59]) are conditions we would prefer to see fulfilled but aren’t strictly necessary, e.g., Bob should wait for 2h30 after lunch before going to the pool, but it’s ok if he doesn’t, as long as he awaits for at least 2 hours.

Processes and events (available in PDDL+ [60]) represent uncontrollable changes in our environment. A Process is a procedure that will always happen when its preconditions are satisfied, e.g., if Bob leaves the pool, he starts to get dry. An event serves as an instantaneous version of a process. It is a consequence of fulfilling its precondition, e.g., if Bob enters the pool, he will get wet.

At the moment, we know how we can model planning problems, but there’s still a piece missing. We require something to look at our modeled domain and attempt to find the desired goal, the planner.

4.1.2 Planner

Planners allow us to solve PDDL problems. An AI planner reads the PDDL domain and uses it to break down and solve the problem. As we said previously, the solver doesn’t know anything about the problem it is solving. PDDL describes the existing relations and constraints in the problem’s domain, and the planner uses it to find a solution.

When the planner finishes solving, it provides a plan (list of actions) that achieves our goal.

The efficiency of the planner depends directly on the efficiency of the algorithm it uses. The nature of the domain influences the algorithm’s performance. Some algorithms might be well suited for complex environments but perform poorly in simple ones. Later, we will describe a few algorithms used by planners.

It's important to note that not all planners support the full PDDL feature set.

At his moment, we have defined the parts that allow us to solve planning/scheduling problems. Following this, we introduce a twist to planning problems, partial-order planning.

4.2 Partial-order Planning

Partial-order planning is an approach to planning that follows a "least commitment" strategy [61]. This approach only forces an order between actions when necessary, giving us a partially ordered plan. Hence the name partial-order planning. By contrast, planners that provide solutions with a total ordering between all actions are called total-order planners.

To exemplify, let's say we are going to the supermarket to buy ingredients for a cake. Our recipe needs eggs, flour, and milk. Image 4.2 shows how the solutions from each planner differ.

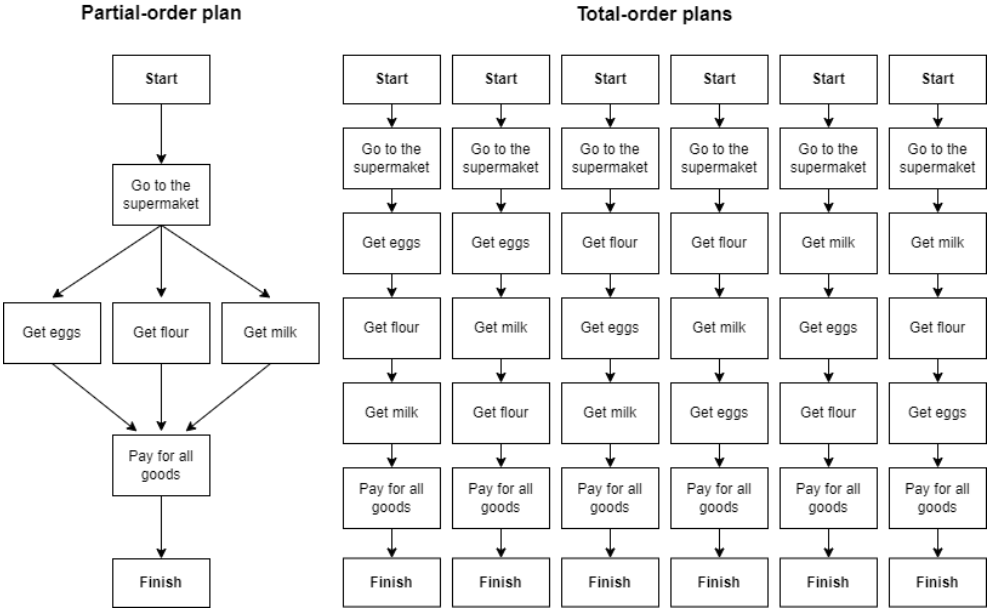


Figure 4.2: Differences between partial-order plans and total-order plans

As we can observe, the partial-order plan aggregates in the same layer actions that can be performed in any order.

In contrast, the total-order planner forces an ordering between every activity. In reality, this planner would only provide one of the total-order plans. Consequently, we wouldn't know that some actions may change their order.

Some authors did a comparative analysis of the two types of planners [62]. Previous studies presumed that total-order planners could never match the efficiency of partial-order planning. However, Minton, Steven et al. showed that the search space of partial-order planners could surpass in size that of total-order planners. They also exposed that a search space advantage doesn't necessarily mean better efficiency.

Their work demonstrates that the search strategy subtly influences the performance of a planner. Algorithms like the depth-first search can benefit more from partial-order planning than distribution-

insensitive techniques can.

For several years, Blum and Furst's Graphlan algorithm [63] was widely used by the planning community. At the time, it emerged as the fastest planner for solving classical planning problems. Graphplan mixes techniques like forward planning (similar to state space searches) and backward search. Backward searches are analogous to forward ones, with the difference that they start from the goal node and work their way back to the initial state.

Following the success of Graphplan, some researchers tried to increase its performance and expand its possible applications.

Kambhampati, Subbarao et al. explained how applying techniques from CSP (Constraint Satisfaction Problem) literature could help with the efficiency of the backward search phase of Graphplan [64].

Gazen, B. Cenk et al. took a different approach and tried to enrich the language used by Graphplan to include features like disjunctions, conditional effects, and others [65].

In the late 90s, partial-order planning became outdated, predominantly due to the exceptional performance increases in forward state space search planners[1]. These new algorithms used an informed heuristic to guide the search. Furthermore, they carried much less processing weight, resulting in a rapid expansion of nodes. We will explain this type of algorithm later.

More recently, the interest in partial-order planning emerged once again. Coles, Amanda et al. explored the potential of forward state search strategies to support partial-order planning in the solving of temporal and numeric problems [66]. Their planner, POPF (Partial Order Planning Forwards), follows the "least commitment" approach, delaying the decision of ordering actions, timestamps, and numeric parameters. While applying an action to a state, it aims to keep only the ordering constraints needed to resolve threats. This way, the planner keeps the possibility of the new action occurring before one already in the plan.

4.3 Algorithms

In the previous sections, we described the components of AI planning. We mentioned that the algorithm we choose influences the search's performance.

Now, we are going to describe some algorithms used for classical planning problems. We divide these algorithms into two classes: uninformed and informed (or heuristic).

4.3.1 Uninformed search methods

Uninformed search methods (also called blind methods) are an approach to state space searches where the algorithm uses brute force operations to find a solution. In a given state, there are a set of possible actions. The algorithm blindly selects one, regardless of whether it is a good option.

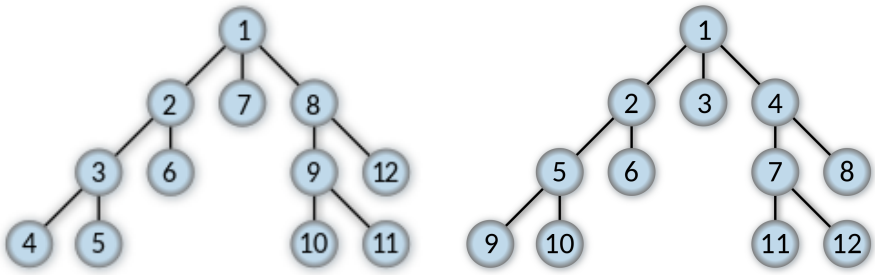
We highlight two uninformed search algorithms: Depth-first search (DFS) and Breadth-first search (BFS).

[1] Forward state space search planners provide a total-order plan. Therefore, they can be considered total-order planners.

DFS is an algorithm that starts at an initial state and proceeds as far down a specified branch of the state space. If the algorithm reaches a dead-end (nodes where no actions are possible), it backtracks until it finds an unexplored path. Picture a sudoku-solving problem. The DFS would start by filling as many positions as it possibly could. When no more moves are possible, it removes the last inserted number and tries another one. This process goes on until it finds a solution.

DFS shows some limitations in environments where the search tree has infinite depth. To solve this, we can impose a limit to the search depth (Depth-limited Search) or consecutively update the maximum search depth (Iterative deepening depth-first search).

The other uninformed search method is the BFS. This algorithm is mainly used in problems where we want to find the shortest path between the initial state and the goal. BFS starts at the root node and checks all states it can reach through a single action. For each of these states, it once again verifies the nodes it can hit. BFS consecutively analyzes nodes at the same depth in the search tree until it finds the goal. The main disadvantage of this algorithm is the memory cost because it requires the computer to store several nodes in memory.



(a) DFS node expansion order **(b)** BFS node expansion order

Figure 4.3: Comparison of expansion order of nodes in DFS and BFS

Uninformed search techniques are appropriate when we have no information on the problem. However, artificial intelligence often requires more refined mechanisms.

4.3.2 Informed search methods

Informed search is a class of algorithms that uses problem-specific knowledge about the domain to achieve a more efficient search. These methods guide the search toward more promising paths. We focus on a specific algorithm, the “A-star” search.

A* (pronounced “A-star”) is a path search and graph traversal algorithm which is frequently used in many areas of computer science due to its completeness[2], cost optimality[3], and efficiency. Unlike other traversal techniques, A* uses information on the goal state which helps with planning ahead at each step so a more optimal decision is made. Like Dijkstra’s algorithm, A* works by making a lowest-cost path tree from the start node to the target node. What sets it apart is that for each node,

[2]An algorithm is complete if it is guaranteed to find a solution when there is one, and to correctly report failure when there is none. [67]

[3]An algorithm is cost optimal if it finds a solution with the lowest path cost of all solutions. [67]

A* uses a function $f(n)$ that gives an estimate of the total cost of a path that uses that node. This estimate is called a heuristic. For a* to work, the heuristic function must be admissible, which means that it must never overestimate the cost of reaching a goal.

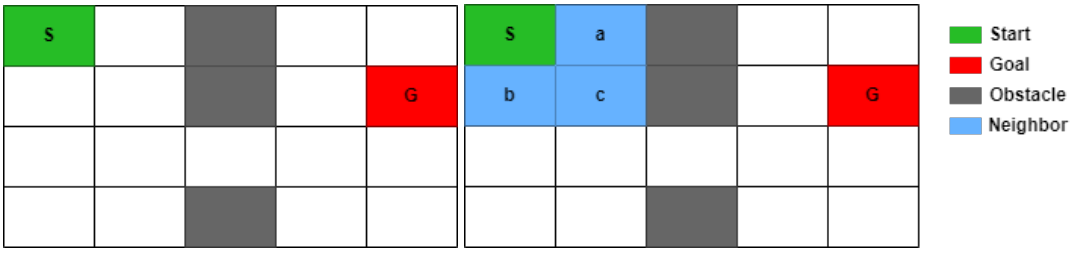
A* expands paths that are less expensive by using the function:

$$f(n) = g(n) + h(n), \tag{4.1}$$

where

- $f(n)$ = total estimated cost of path through node n
- $g(n)$ = cost so far to reach node n
- $h(n)$ = estimated cost from node n to goal (heuristic)

The 2D grid search is a popular and simple to understand example of the A* algorithm. Let's have a look at a 4x5 grid where we are trying to determine the shortest distance between the start position, $S = (1, 1)$ and the goal, $G = (2, 5)$. Additionally, let's assume that there are some obstructions, as shown in 4.4(a).



(a) Initial State **(b)** First positions to be evaluated

Figure 4.4: 2D Grid Example

The algorithm will start by looking at the start node's neighbours (4.4(b) and estimating their distance to the goal state.

We will utilize the Euclidean Distance as a heuristic function, which informs us that the distance from x to y is given by:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}, \tag{4.2}$$

where x_1 and x_2 are the coordinates of point x and y_1 and y_2 are the coordinates of point y.

By applying the heuristic function, considering that $h(n) = d(n, G)$, we get the following values for positions a, b and c:

$$h(a) = d(a, G) = d((1, 2), (2, 5)) = \sqrt{(2-1)^2 + (5-2)^2} = \sqrt{10} \approx 3,16 \tag{4.3}$$

$$h(b) = d(b, G) = d((2, 1), (2, 5)) = \sqrt{(2-2)^2 + (5-1)^2} = \sqrt{16} = 4 \tag{4.4}$$

$$h(c) = d(c, G) = d((2, 2), (2, 5)) = \sqrt{(2-2)^2 + (5-2)^2} = \sqrt{9} = 3 \tag{4.5}$$

The total estimate of the cost of the paths to the goal can be obtained now that we have the heuristic values. Since we are using a grid, vertical and horizontal moves have a cost of 1 and diagonal moves have a cost of $\sqrt{2}$. Therefore, using expression 4.1, the estimated path costs are:

$$f(a) = g(a) + h(a) = 1 + \sqrt{10} \approx 4,16 \tag{4.6}$$

$$f(b) = g(b) + h(b) = 1 + 4 = 5 \tag{4.7}$$

$$f(c) = g(c) + h(c) = \sqrt{2} + 3 \approx 4,41 \tag{4.8}$$

The algorithm will then select $a = (1,2)$ as the next position, since it has the lowest path cost estimate. However, as we can observe in 4.4(b), after moving to position a, we can only move to c, due to the obstacles in $(1,3)$ and $(2,3)$. Consequently, the algorithm will go back and select c as the next position. The following iterations of the algorithm are represented in 4.2. For simplicity purposes, only the positions that are most likely to be the next are shown. For identical reasons, the heuristic values are omitted.

Current Position	Current Cost	Possible Positions	Next Position
$c = (2,2)$	$\sqrt{2}$	$(2,1),(1,2),(3,1),(3,2),(3,3)$	(3,3)
$(3,3)$	$2\sqrt{2}$	$(2,4),(3,4),(4,4)$	(2,4)
$(2,4)$	$3\sqrt{2}$	$(1,4),(1,5),(2,5),(3,4),(3,5)$	(2,5)
$(2,5)$	$1+3\sqrt{2}$	-	-

Table 4.2: Iterations of the A* Algorithm

When the goal is reached or if there are no more positions to test, the search is complete. The path found by the algorithm is optimal and has a cost of $1 + 3\sqrt{2} \approx 5,24$.

The result obtained in expression 4.8 confirms that the heuristic function did not overestimate the cost, since $4,41 > 5,24$.

With this example, we hope to have clarified how the heuristic value guides the search toward the correct paths.

The best-first search (different from BFS) is another informed search method worth mentioning. Just like A*, it successively expands less expensive nodes. The difference is that the estimated path cost is given by $f(n) = h(n)$, which means the algorithm only considers the heuristic value (estimate) and not the actual cost. For this reason, this method is also known as a greedy search.

We now have described a plurality of techniques used in classical planning. Following, we will introduce a relatively new concept that combines planning and reinforcement learning.

4.4 Combined Approaches

In the context of sequential decision-making, there are two approaches: reinforcement learning and planning. The artificial intelligence field has recently started combining both in something called model-based reinforcement learning. This area consists of “any MDP approach that i) uses a model (known or learned) and ii) uses learning to approximate a global value or policy function” [68].

In this section, we focus on a single application, AlphaGo. Google DeepMind's AlphaGo is a computer program that plays the board game Go. It gained popularity by defeating the reigning human world champion of Go, Lee Sedol (match score was 4-1).

The focus of AI research on this strategy game comes from its complexity. Standard games of "Go" use a 19x19 grid. The players take turns placing the pieces on the empty positions of the board. A typical game of Go has 250 possible moves each turn and a game depth of 150 turns. So we are looking at about 10^{360} possible actions in a single game. For comparison, chess only has about 10^{120} possible moves, and there are "only" 10^{82} atoms in the observable universe. These numbers show how unfeasible it is to compute all the possibilities to come up with a solution.

AlphaGo introduces a combination of advanced (heuristic) tree search techniques with deep neural networks.

The Monte-Carlo tree search (MCTS) is a heuristic search algorithm that uses Monte-Carlo rollouts (simulations) to estimate the reward of each state. In every simulation, the algorithm plays the game until the end by selecting random moves. The game result helps assigning weights to each node in the search tree. This way, actions that result in a win are more likely to be played. As we increase the number of Monte-Carlo rollouts, the search converges to perfect play. AlphaGo enhances MCTS with reinforcement learning.

The neural networks take a description of the Go board as an input and process it. One neural network, the "policy network" selects the next move to play. The other neural, the "value network", predicts the winner of the game.

These networks are trained, firstly, by data from human expert games, and secondly, by games of self-play (reinforcement learning) [69].

AlphaGo Zero (a subsequent version of AlphaGo) achieved an even higher level of play training only against itself.

4.5 Conclusions

In this chapter, we started by describing planning. This field uses a state space search to find a sequence of actions (plan) that takes us from the initial state to the goal.

We then introduced PDDL, which is a family of domain-independent planning languages. Its syntax relies heavily on the logical relation between agents and objects.

After that, we defined partial-order planning, which is a planning approach that only forces an order between actions when necessary (least commitment strategy).

Then, we specified some algorithms used in classical planning. The depth-first search (DFS) and breadth-first search (BFS) are uninformed search methods. Later, we moved on to heuristic (informed) searches, like the "a* algorithm". This method uses information about the goal to navigate the search tree more efficiently.

Following, we introduced Reinforcement Learning which is an approach to sequential decision-making that focuses on learning through experience. The algorithm "plays" with the environment

several times, thus discovering which actions lead to the maximum reward.

Finally, we discussed an application of model-based reinforcement learning. The AlphaGo program proved the power of this approach by achieving a super-human level of play in Go, surpassing all previous Go-playing programs.

In the next chapter, we will describe our action prediction model, which takes ideas from AlphaGo's model-based reinforcement learning strategy.

5

Proposed model

Contents

This thesis started by outlining the importance of the human-robot collaboration field. We then described the psychological processes that allow humans to efficiently and effortlessly cooperate. Furthermore, we also exposed how action prediction underlies collaboration. Finally, we presented two approaches for action prediction: one focused on data (neural networks), and the other focused on planning.

We now introduce our model, which intends to explore the combination of the methods described above for action prediction.

5.1 Problem Statement

Human-robot collaboration requires the robot to anticipate human action. This action anticipation relates to predicting possible future actions and selecting the most probable ones with the help of environmental cues. We model this problem as a Markov Decision Process (MDP).

The MDP is a model for decision-making scenarios where the output depends on the decision maker's behavior and some random conditions (stochastic model). In our case, the decision-maker (AI agent) tries to select the action that brings the most reward, considering the human's possible actions (random condition).

A MDP is a tuple $< S, A, P_a, R_a >$ with:

- Set of possible environment states (state space), S
- Set of possible agent actions, A
- Set of state transition probabilities, P_a. Here $P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$.
- Expected immediate reward function, R_a. The reward after transitioning from state s to state s' by choosing action a is represented by $R_a(s, s')$

5.2 Problem domain

To test the implementation of our model in a context of action prediction, we defined a relatively simple domain with the following characteristics:

- **Environment** - 5x5 (2D) grid world.
- **Objective** - Agents need to collect items scattered around the grid and deliver them at a goal location.
- **Number of Agents** - 1 or 2 agents (AI agent + human agent).
- **Goal location** - A set of coordinates, e.g., goal location $= (2, 2)$
- **Items** - Several vegetables scattered around the grid.

- **Possible Actions** - Agents can move up, down, left, or right if they remain inside the grid. If they are at the location of an item, they can pick it up. If they are at the goal location holding an item, they can put it down. Agents can only hold one item at a time.

Only allowing vertical and horizontal moves may seem unambitious. However, it introduces parallel paths, as there are several ways to get from one point to another in a 2D grid.

Although the environment dimensions are relatively small the search space can quickly explode, i.e., grow so much that it becomes computationally unfeasible to complete the search.

Figure 5.1 gives an example of the type of problems we intend to solve with this algorithm.

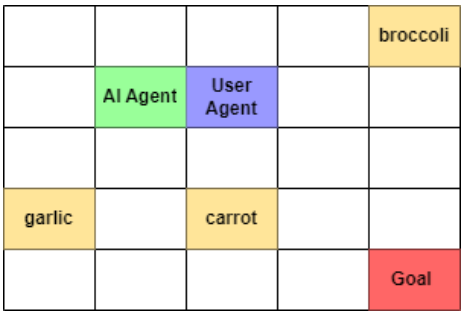


Figure 5.1: Domain Example

The “AI Agent” and the “User-Agent” start at the given position. They aim to pick up all the vegetables (garlic, carrot, and broccoli) and deliver them to the goal. The employed strategy depends on both the user and the AI agent. The user may choose to cooperate or not. The same goes for the AI agent.

Later in this section, we will describe several strategies we implemented in the planning phase and compare them. With this, we hope to find more insight into human action prediction.

5.2.1 Domain Representations

In this section, we describe the steps that lead us to our current domain. As we previously referred, our model’s structure follows an AI Planning ideology.

The core of classical planning resides in PDDL. For an extended period, researchers worked toward building efficient planners. While that happened, the language (PDDL) remained almost the same (with minor improvements described in the planning chapter).

For these reasons, we started by testing simple problems and using PDDL planners to solve them.

Throughout our experiences, we encountered some difficulties that would become less relevant if we moved to a PDDL-like domain constructed by us.

This new domain version conserved the essence of planning. We now describe both domains.

5.2.1.A PDDL

We first modeled the problem as a PDDL kitchen domain in which we were able to test some tools described in the previous chapter (AI planners). They proved convenient when it came to finding

an optimal solution and providing a plan (set of ordered actions to achieve the goal). The planning domain definition languages have the advantage of being defined by simple logical operations.

However, we had no control over the algorithm the planners used. A majority of them used a heuristic search.

Even so, we tried utilizing a planner to obtain a partial-order solution to apply to our model. The partial plan it provided applied the "least commitment" when working with more than one agent [66]. By this, we mean that the planner correctly showed when an action that gets us to the goal state could be performed by either agent:

$$S_2 = S_1 + a_1^{agent1} = S_2 + a_1^{agent2} \tag{5.1}$$

where S_2 is the desired state, S_1 is the current state, a_1^{agent1} is action 1 performed by agent 1, and a_1^{agent2} is action 1 performed by agent 2.

However, when working with only one agent, we wanted the planner to consider all the paths that lead from the current state to the goal state (as shown in figure 5.2). Here, the partial planner failed, showing only one possible (optimal) action.

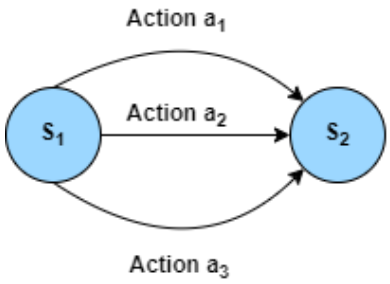


Figure 5.2: Desired solution from the partial planner

A successful collaboration strategy requires understanding all possible actions. If the agent can achieve the same objective in various ways (sub-actions), we want to consider all of them.

To implement a desirable single-agent partial-order plan, we changed our domain's representation.

5.2.1.B Non-PDDL

As we indicated in the last section, the PDDL planners used heuristic search techniques. Ideally, we intended to achieve similar performance, with the addition of getting partial-order plans.

Research speculates that would be possible [70], but it would be hard to implement.

To simplify, we considered a maze-solving scenario, which can include the action prediction aspects we are trying to study. Since planners used heuristic searches, we started by building a simple domain where we could develop the A* ("a star") algorithm. The immediate problem was finding a heuristic function (which was not necessary for PDDL).

The curious aspect of the heuristic is that it should not necessarily be admissible (reminder: admissible means that it never overestimates the cost from a state to the goal) for complex problems. According to the theory of mind, humans interpret actions in terms of their objective ("What am I trying

to achieve by doing this?"). As situations grow in size and complexity, people opt for a "divide-and-conquer" approach, i.e., to plan only their next step (sub-goal) instead of the whole solution (goal). This behavior can translate into humans not acting optimally, which proves that even if A* finds non-optimal solutions, it can still accurately describe human action. We call these semi-optimal solutions.

Assuming A* can find a solution, we now have a total-order plan and its cost. To get a partial-order plan, we rerun the algorithm with the characteristics: 1) If we reach the goal state, we want to record the path of actions that got us there (plan) and the cost; 2) When we reach the goal state, the algorithm must continue exploring states that are already in the search tree; 3) If a state's cost estimate to reach the goal ($f(s)$) exceeds the amount in 1) we don't explore it.

This algorithm returns a list of all the paths (optimal or semi-optimal) that lead from the initial state to the goal state. We then use this list to get a partial-order plan.

The first time we run A*, we obtain the cost in which we can reach the goal state. We can look at this value as the maximum depth for our second search tree. The changed A* can then find all solutions within that depth. These solutions can then train a "policy network", which selects the best action in a given state.

We hypothesize that by randomly navigating through the bounded search tree (Monte Carlo simulations), we will also be able to create a "value network" similar to AlphaGo. This network predicts if we can achieve a goal node by taking a given action.

In short, this section discussed relevant milestones and results that justify our current domain. We first tested a PDDL approach. However, we could not get a partial-order plan of the form we desired.

Moving to a non-PDDL language we found a way of getting the appropriate solution. However, it introduced much more complex logic, such as a heuristic function and defining the domain, which was relatively simple in PDDL.

In the next section we describe the different implementations we considered.

5.3 Model's Approach

Our model's approach reutilizes some "AlphaGo concepts" in the context of human action prediction. The main objective is to develop an AI agent capable of executing shared actions with a user.

We define a starting state and a goal state as classical planning problems do. To ensure the algorithm follows the laws that regulate our domain, we provide some information that allows it to assert possible actions. This way, it doesn't consider impossible moves and gains the ability to plan several timesteps ahead. We can see this description as an attempt to give the algorithm a shared representation.

With this information, the algorithm predicts both the user's possible moves and the AI agent's possible moves.

Then, the action selection process will evaluate what action combination brings the most shared reward to our agents based on a policy. This policy will ideally be the result of training a reinforcement

learning model. The algorithm utilizes this advanced tree search method to move an AI agent while the user inserts its moves. Figure 5.3 presents the model's diagram.

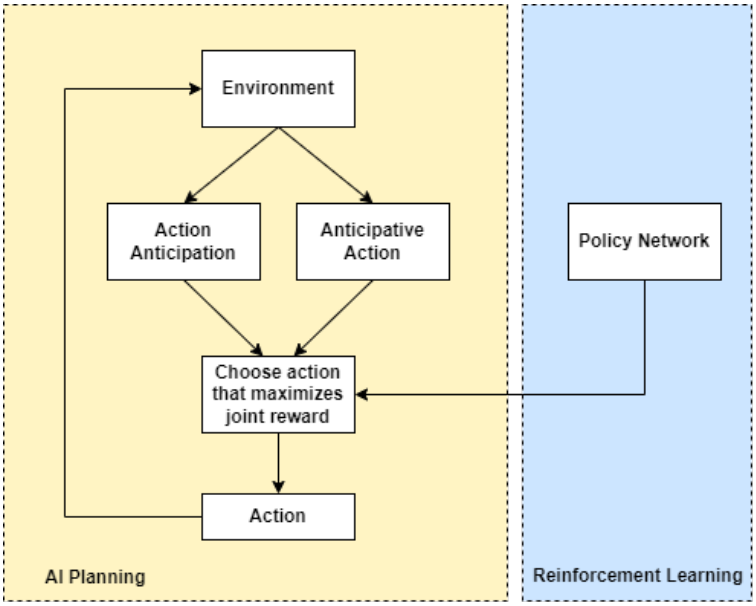


Figure 5.3: Combined approach diagram

This model raises some complex issues that were not present in the AlphaGo case.

For starters, the AlphaGo algorithm applies to a game-playing scenario in which there is always a winner. Board positions are limited, which implies that there is also a maximum number of turns. In terms of the search algorithm, this translates to the search tree having a maximum depth. The bounded search space allows this model to use Monte Carlo simulations to get essential data for the policy network training process.

If we try to use Monte Carlo simulations in an action prediction scheme, there is a possibility of getting an infinite-depth search tree. This case happens because human behavior isn't always intelligent behavior. In the case of AlphaGo, the algorithm's goal is to win. So in the eventuality of the opponent making a mistake (bad move), the algorithm is most likely to reduce the number of turns until victory. In the context of action prediction, "bad moves" mean acting unpredictably. It is also very challenging to predict how often these erratic actions will happen.

However, we argue that humans neither act optimally nor randomly (unpredictably). Our model uses clues to select the most probable states efficiently and thus guide the tree search to the goal.

5.4 Implementation

The objective of this thesis was to create an AI agent that can collaborate with a human agent (user). We described that one of the conditions for cooperation is that both agents must know the goal and what they can do to achieve it. This way, our first model is an agent that could complete the task alone.

5.4.1 Single Agent model

The single agent model, although relatively much more simple than our initially proposed model, requires us to define most of the logic present in the domain. We start by implementing an agent that acts optimally. The algorithm's structure is presented in figure 5.4.

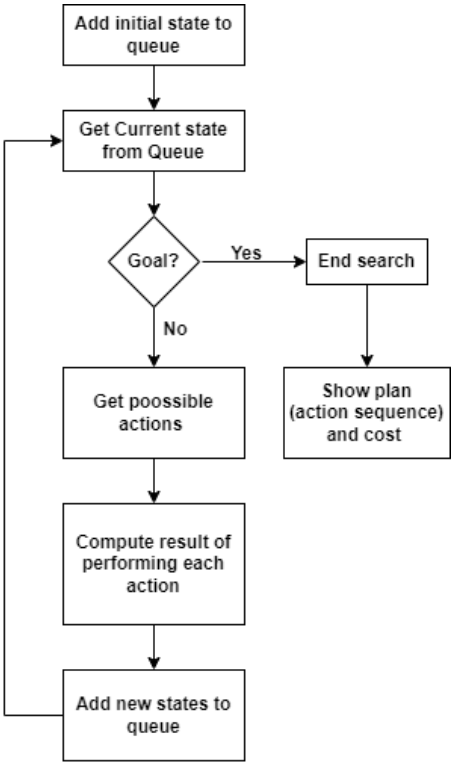


Figure 5.4: Single Agent algorithm structure

The algorithm starts by adding the initial state to a queue that keeps all the non-expanded nodes. Assuming the initial node is not a goal state, the algorithm generates all the possible moves the agent can perform. Next, it calculates the outcome of applying each action to the current state and adds all the resulting states to the queue.

The process of selecting the next state to be evaluated from the queue utilizes the heuristic function, i.e., the algorithm selects nodes according to a heuristic value. This value is equal to the cost of getting to a state (current cost) plus an estimate of the cost from that node to the goal (estimated cost).

Since we want to find a minimal-cost path, the algorithm selects the node with the lowest value from the queue.

By consecutively choosing to expand states with the lowest cost, we ensure that if we find a solution, it is optimal (if the heuristic function never overestimates the distance to the goal state).

Now, we are going to describe the heuristic function the AI agent uses.

5.4.1.A Heuristic Function

The heuristic function provides a rough estimate of the cost of going from a given state to the goal state. The more accurate the heuristic function is, the fewer states the algorithm needs to test, which can considerably improve performance.

If we consider the example in figure 5.5, the goal “Deliver all vegetables at the goal location” can be divided into: “Pick up carrot”, “Deliver carrot at goal location”, “Pick up broccoli”, and “Deliver broccoli at goal location”. These sub-goals imply that the agent must decide which vegetable to pick up first, which supports the claim that humans perceive actions in terms of goals.

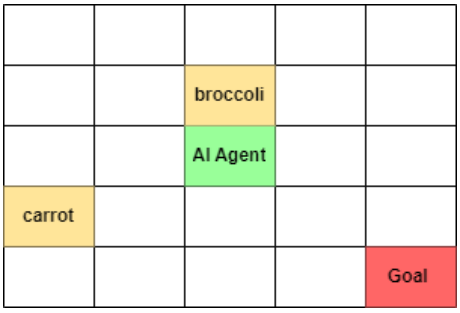


Figure 5.5: Single Agent problem example

The domain’s rules say that to pick up an object, we must first move to its location. From the algorithm’s perspective, this means choosing actions that move the agent toward the object’s position, i.e., actions that minimize the distance between the agent and the item. We introduce the expression we use to compute the distance between two points in this domain.

The Manhattan distance (or taxicab distance) between x and y is given by:

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i) \tag{5.2}$$

Since our environment is a 2D Grid, we can simplify expression 5.3, by plugging in $n = 2$:

$$d(x, y) = (x_1 - y_1) + (x_2 - y_2) \tag{5.3}$$

Now that we’ve established a way to calculate distances, we can focus on the heuristic’s logic. We consider two possible cases: the agent is currently holding an object, or its hands are free.

If the agent is holding an item, the state’s heuristic value, $h(s)$, is given by:

$$h(s) = d(a, g) + 2 * \sum_{i=1}^{n} d(g, o_i) \tag{5.4}$$

where a is the agent, g is the goal, and o is the list of vegetables that are still not in the goal location. This value corresponds to the cost of moving from the current location to the goal and putting down the vegetable, plus the cost of grabbing and delivering every remaining item on the grid.

We would like to highlight that picking up and putting down an object has no associated cost.

If the agent is not holding any item, the algorithm needs to choose a sub-goal. Looking at figure 5.5, we can observe that the goal is at the same distance from the carrot and the broccoli. However,

the agent is closer to the broccoli than to the carrot. For this reason, the algorithm decides to move in the direction of the broccoli first. The calculations in equation 5.5 confirm our assessment. h_b is the path cost estimate when the agent picks up the broccoli first. h_c corresponds to picking up the carrot first.

$$\begin{aligned} h_b &= d(a, broccoli) + d(broccoli, g) + 2 * d(g, carrot) = 1 + 5 + 2 * 5 = 16 \\ h_c &= d(g, carrot) + d(carrot, g) + 2 * d(g, broccoli) = 3 + 5 + 2 * 5 = 18 \end{aligned} \tag{5.5}$$

We can generalize the expressions for more items, getting that the cost estimate of choosing to pick up a given item as the first sub-goal, $h(i)$, is equal to:

$$h(i) = d(a, o_i) + d(o_i, g) + 2 * \sum_{j=1}^{n} d(g, o_i), \qquad \text{with } i \neq j \tag{5.6}$$

Choosing the best sub-goal is equal to selecting x that minimizes expression 5.6

At this moment, we have described all the tools that allow the algorithm to find a solution. It follows the A* search technique structure (5.4), utilizing the heuristic function we just outlined. When the algorithm finds a solution, it returns a plan of the agent's actions as represented in figure 5.6. Remember that only "move" actions have an associated cost, that's why there are 20 timesteps, but the cost of the plan is only 16.

```
Plan: (cost=16)

Timesteps    Actions
-----------  --------------------------
T=0          AI move UP
T=1          AI pick-up broccoli (1, 2)
T=2          AI move DOWN
T=3          AI move DOWN
T=4          AI move DOWN
T=5          AI move RIGHT
T=6          AI move RIGHT
T=7          AI put-down broccoli (4, 4)
T=8          AI move UP
T=9          AI move LEFT
T=10         AI move LEFT
T=11         AI move LEFT
T=12         AI move LEFT
T=13         AI pick-up carrot (3, 0)
T=14         AI move DOWN
T=15         AI move RIGHT
T=16         AI move RIGHT
T=17         AI move RIGHT
T=18         AI move RIGHT
T=19         AI put-down carrot (4, 4)
```

Figure 5.6: Plan of AI agent's actions

Now that we have an AI Agent capable of solving the problem on its own, we can begin to incorporate collaboration with a human.

5.4.2 Simple Multi-Agent model

After getting the AI agent to learn how to act alone, we need to insert it in a shared space. If we implement no modification to the single-agent model algorithm, the agent will be unaware of the user's presence and the impact of the user's actions in the environment.

The previous model elaborates a plan at T=0 and follows that sequence of actions. However, to achieve cooperation, we need the agent to predict what actions the user will perform and adapt if the prediction is wrong. After all, as Grosz says, “Collaboration is not just sums of individual plans” [71]. Since the simple multi-agent model isn’t more than two agents acting alone, we won’t spend more time describing it.

Just picture two agents at different starting positions, executing the plan-making algorithm described in the previous section. We expect that one agent’s actions will interfere with the other’s, not in a cooperative way. In the results section, we register the experimental observations of this model.

For now, we set it aside and focus on more appealing approaches.

5.4.3 Replanning Agent model

5.4.3.A Description

The Replanning Agent is our first legitimate attempt at collaboration between an AI agent and the user agent.

In this model, the AI understands that the relationship between the environment and each agent is fundamentally the same. In other words, the algorithm considers that the other agent can also change the state of the shared physical space. We can look at this as some form of Theory of Mind.

Figure 5.7 displays this model’s simplified architecture.

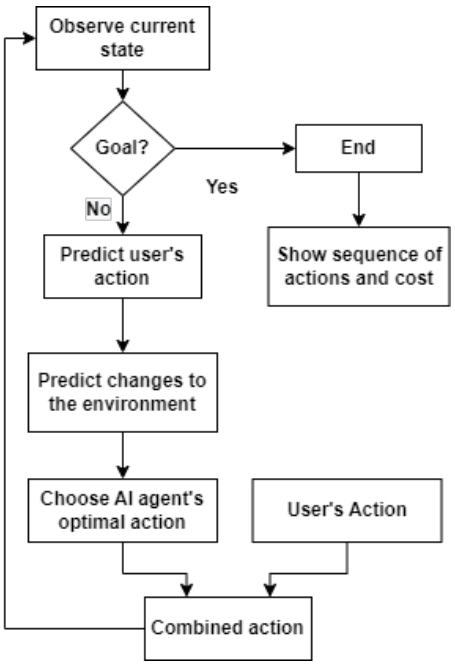


Figure 5.7: Replanning Agent algorithm’s structure

Let’s have a quick overview of what happens in each part of the algorithm.

Starting at a state that is not the goal, the AI agent will predict the partner’s move. This prediction uses the algorithm described in the single agent model. The algorithm will provide an optimal plan of user actions in a scenario where the user is working alone. The replanning agent extracts the user action prediction from this plan.

Assuming it has the correct prediction, the agent internally computes the environment changes that result from that user action.

Finally, the AI agent chooses its move by computing the shortest path from the predicted state (after the user's move) to the goal state.

The algorithm receives the user's action and, together with the AI's action, updates the environment. Whenever it reaches a solution, it returns the sequence of recorded moves and the associated cost.

5.4.3.B Model limitations

Replanning allows the agent to adapt to changes in the environment. However, this model still has some imperfections.

Firstly, predicting the user's move only relies on the first solution found. The algorithm must consider that there might be more than one optimal solution to become more accurate. We can do this by applying a technique described before. First, we find the minimal cost from the current state to the goal. Second, we do a bounded tree search and find all paths within that cost that reach a goal state. With this, we get a tree containing all optimal paths. In this case, the rule for selecting the AI agent's move is to choose the action that leads to the biggest number of optimal paths. The idea is to introduce some uncertainty in the prediction, keeping several possibilities (least-commitment strategy).

Secondly, the AI agent knows that the other agent can interact with the environment but doesn't know it has its own mind. The user agent might be closer to an object than the AI agent, and the AI still tries to move to that location and pick it up. The algorithm only changes its sub-goal when the user finally picks up that object, which causes a change in the environment.

5.4.4 Goal-Predicting Agent model

The next model we introduce is the goal-predicting agent. While the previous approaches focused on predicting the partner's next move, this model targets the sub-goal the human intends to complete. We base this strategy on the theoretical belief that humans act according to their goals instead of planning each action.

This approach gets rid of the previous limitations since it considers that the other agent (human) has a mind of its own and, therefore, each move is part of a plan to achieve a given objective.

At the beginning of the problem execution, we enumerate the sub-tasks that compose the final goal. In a real-world application, this would be like having a recipe. Each agent selects a sub-task from the list according to a strategy, e.g., complete tasks that take longer first.

Here, we define the AI agent's strategy, and it assumes the human will follow it as well. However, if the partner acts in order to complete a sub-goal different from the predicted one, the AI also adapts by picking another assignment.

We present the model for the AI agent's behavior in figure 5.8.

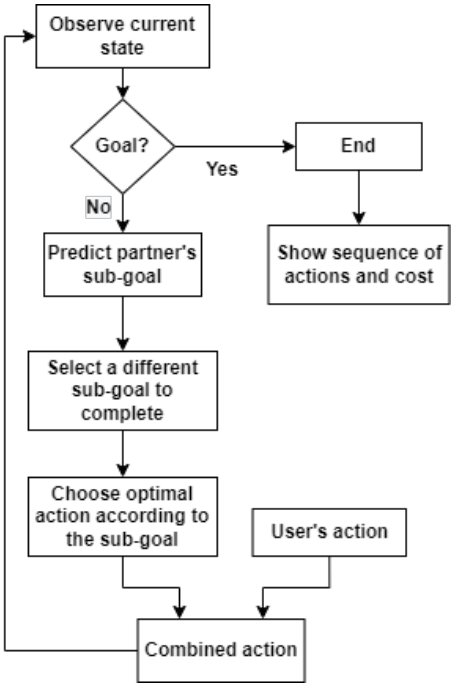


Figure 5.8: Goal-predicting Agent algorithm's structure

The present model shows a significant improvement in the area of collaboration. It takes advantage of the fact that the goal is composed of smaller parts. In some real-world situations, the task's decomposition may not be trivial. As far as this thesis is concerned, problems are always pre-divided.

This approach, although more complex than the others, still follows an individual strategy. Like in the replanning agent model, this algorithm adapts to the partner. The difference is it focuses on the sub-goal instead of each action.

The other approach is combined (or joint) strategies, which make decisions that maximize the shared reward, i.e., select actions that bring the most immediate shared reward for both agents. This idea follows that there are no individual actions. Instead, it considers a single joint activity at each timestep.

5.4.5 Best Shared-Reward Agent model

The previous models only evaluated the individual reward of a move, i.e., they chose actions that were most beneficial to one of the agents

The best shared-reward model introduces the idea of joint planning. This concept considers that there are no individual moves, only a single collective action. The difficulty of this approach is creating a function that correctly evaluates the shared reward of this combined action (human action + AI action). In the AlphaGo algorithm, a trained neural network was in charge of providing this value.

Since creating the function that estimates the shared reward of a joint action proved extremely challenging, we implemented a model with "hardcoded" values to demonstrate the power of this approach (5.9.

We consider this model to be the best representation of the collaboration between agents. The

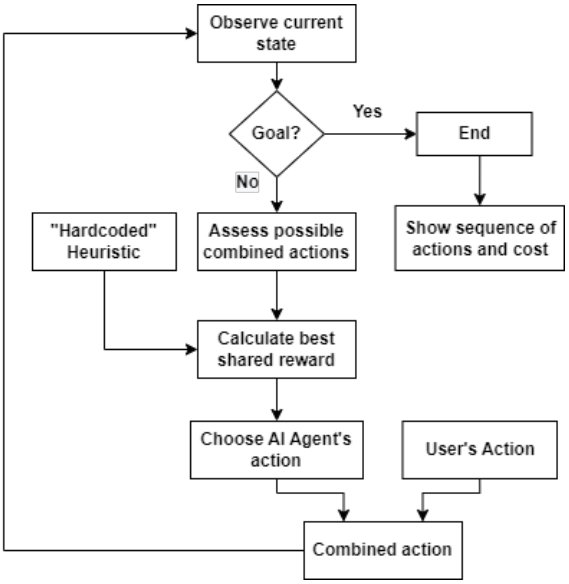


Figure 5.9: Best Shared-Reward algorithm's structure

hardcoded heuristic allows us to evaluate the model's performance and define it as the maximum collaboration ceiling that the other models try to achieve.

5.5 Conclusions

In this chapter, we presented the ideas from which our model took inspiration. Google DeepMind's AlphaGo pioneered approaches that combined AI planning and reinforcement learning. We wanted to replicate this method and apply it to a human-robot collaboration problem.

Right away, we predicted that some issues that weren't present in the Go-playing environment would arise. Since we are trying to achieve cooperation, we don't necessarily want to act optimally (like when trying to win the game). Our focus is to predict our co-actor's next move.

We then described the modifications our domain suffered and the causes that led to them. At last, we presented the group of models we used to study action prediction. The first ones only considered individual plans, while the others introduced some aspects of collaboration.

The first big step was building the replanning agent, which adapted to environmental changes. Although it performs better than a single-plan-following agent, it still lacks some collaborative skills.

The goal-predicting agent introduces the idea of focusing on the prediction of sub-goals (sequence of actions) instead of targeting single steps.

The final model was the best shared-reward agent, which employed a joint planning approach. Ideally, we would use reinforcement learning to get a function that evaluates each combined action, just as proposed in figure 5.3.

6

Results

Contents

The previous section described all the models we implemented to understand human action prediction.

In this section, we document the experimental results obtained by our models. We describe the set of problems we used and the defined metrics. We then compare the performance of each model and draw some conclusions.

6.1 Problem set

The performance of our models is evaluated on a randomly generated set of 2D grid search problems. All the items in this set have 5x5 dimensions to ensure the search space doesn't grow too large.

Working with a 2D grid has the advantage of us being able to visually observe the environment's evolution and compare it with our empirical assumptions.

Each problem is a combination of an initial state and a goal state on the grid world. We insert agents and objects in a given position in the grid. The AI agent either works alone (single agent model) or with the help of a human agent. The goal is to pick up all items and deliver them in a given position. The number of items (n) varies between 2 and 7.

The user inputs (human agent actions) depend on the model we use. For example, in the simple multi-agent model, the human acts according to an optimal individual strategy, while in the goal-predicting algorithm, the user tries to collaborate.

6.2 Performance Metrics

The objective of this thesis is to build an AI Agent model that can collaborate with a human. As detailed in the domain's description section, we focus on maze-solving problems. Since we only allow vertical and horizontal moves, there are multiple ways of moving from point A to point B. As the number of actions to achieve the goal state grows, more parallel optimal paths exist.

This fact indicates that it would be **wrong** to measure our model's performance with classical accuracy formulas, such as:

$$Accuracy\ \ (\%) = \frac{correct\ predictions}{total\ predictions} * 100 \tag{6.1}$$

Instead, we evaluate the models using a much more suitable metric. If the models correctly identify the sub-goal of the user agent, the AI agent will try completing a different sub-goal. If, on the other hand, the models wrongly identify the user's sub-goal, the AI agent may try to complete that goal itself. This error will translate to an increase in the combined cost of solving the problem.

To give an estimate of the model's performance, we compare the timesteps it takes to solve the problem with that of an ideal model. In this context, the "ideal model" means an agent that follows the same strategy as the user (best shared reward model) and not necessarily an optimal agent.

We estimate how capable of collaborating a model is by using the expression:

$$C = \frac{t_{ideal\ model}}{t_{current\ model}} \tag{6.2}$$

where C is the "estimation of collaboration", t_x is the number of timesteps required to find a solution with model x, and the ideal model is the best shared reward model. C has no physical meaning, but it allows us to compare different models.

The second metric we used is the elapsed time to find the solution, t. Here we only consider the time the algorithm spends searching for the AI agent's action. With this we intend to get a view of the computational cost of each model.

We present the experimental results in tables 6.1 and 6.2, where n is the number of items in the problem.

	Single Agent	Simple Multi-Agent	Replanning Agent	Goal-Predicting Agent
n=2	0,6	0,67	1	1
n=3	0,62	0,68	0,87	1
n=4	0,51	0,53	0,83	0,83
n=5	0,44	0,48	0,85	1
n=6	0,5	0,53	0,86	0,97
n=7	0,49	0,62	1	1
n=8	0,57	0,70	1	1

Table 6.1: Values of C for each model in each problem

	Single Agent	Simple Multi-Agent	Replanning Agent	Goal-Predicting Agent
n=2	0,06s	0,06s	0,63s	0,07s
n=3	0,01s	0,01s	0,23s	0,13s
n=4	0,10s	0,10s	1,54s	0,40s
n=5	0,07s	0,07s	1,67s	0,6s
n=6	0,10s	0,11s	3,51s	1,33s
n=7	0,10s	0,11s	3,52s	1,34s
n=8	0,06s	0,08s	2,23s	1,07s

Table 6.2: Values of t for each model in each problem

Firstly, we can observe that both the single-agent and simple multi-agent models get considerably worst value of C than models that adapt their strategy considering the partner. Furthermore, we can also notice that the second model, although having multiple agents, doesn't collaborate.

The replanning agent results are better than initially expected. This model achieves relatively high values of C, even when n increases. We presume its success comes from the existence of several sub-goals per problem. We mean that since there are many tasks that we can complete concurrently if the algorithm makes a wrong prediction of the partner's action, it doesn't always translate into an increase in cost in the final solution. If we were to use a larger grid with fewer items on it, we expect the algorithm to be heavily penalized for the wrong action predictions. Additionally, we see that the collaboration achieved by this model comes with a relatively high computational cost.

The goal-predicting agent shows the best performance as expected. Just like the replanning agent, it re-evaluates the state of the environment at each time step, achieving high indices of collaboration. However, we demonstrate that predicting the partner's goal can be more efficient than precisely anticipating its next move. We speculate that this increase in performance comes from us defining the set of sub-goals before the agents start acting. Being true, this shows the need for an effective partial-order planner to decompose the goal into smaller task.

7

Conclusions and Future Work

Contents

The Human-robot collaboration field is gaining more relevance due to its increasing number of applications, which emphasizes the need for proper collaborative systems.

We started this thesis with an introduction to the theoretical concepts that define cooperation between humans. Collaboration requires agents to have an equal representation of the environment, so they can anticipate each other's actions and correctly adapt to them.

We showed that mental states (such as goals) guide human action, with the Theory of Mind being the mechanism that allows humans to comprehend mental states. We also discussed the implementation of this mechanism in robots.

Having highlighted the importance of action prediction, we introduced two approaches for action prediction.

Neural networks are algorithms that use data about the problem to learn patterns. These networks have several possible architectures, of which we chose to highlight three. Convolutional Neural Networks show convincing results in image processing. Recurrent Neural Networks perform exceptionally well with problems with a sequential nature (like natural language processing). Last but not least, we presented reinforcement learning, in which the algorithm learns to predict by attempting to solve the problem several times (trial and error).

The second approach we introduced was planning. This methodology uses the structure of the environment (rules and constraints) to choose the sequence of actions that take us closer to a given goal. Here, we also described some algorithms used in classical planning problems. Finally, we introduced a combined application of planning and neural networks and discussed their importance.

We then presented our model to achieve collaboration. We define it as a Markov Decision Process, i.e., a sequential decision-making scenario. We used 2D grid search problems to study action prediction. Our proposed model re-uses the idea of combining planning and neural networks, this time in the context of action prediction.

We built several models to study how we can achieve better human-robot collaboration. The obtained results demonstrate that it is better to focus on predicting our partner's sub-goal than it is to anticipate its next move.

The main contribution of this thesis is a new approach to human-robot collaboration. Our model-based algorithm utilizes planning to assess possible actions. We choose the best action with the help of a heuristic function. Ideally, the heuristic function should be substituted for a policy network (reinforcement learning) that evaluates the shared reward of joint actions (as shown in figure 5.3). In this thesis, we "hardcode" the heuristic values to simulate the performance of that model. Our observations confirm that the model-based reinforcement learning model can achieve better collaboration than previous models. We use this model as a benchmark to test all models.

7.1 Future Work

The model this thesis proposes can be expanded in two ways:

- Implement the policy network to estimate heuristic values. We propose that the algorithm uses

the Monte Carlo method to train itself. This method can create a joint distribution for the actions of both agents and assess which combination is the most beneficial to the group. The results of this thesis show that goal prediction leads to better cooperation than action prediction. With this in mind, we also propose training a second policy network, where the Monte Carlo method would create a joint distribution for the sub-goals of the agents. This approach mirrors human behavior in decomposable tasks. Instead of forming an optimal plan, humans focus on the next few sub-tasks they must complete (making the decision process faster). We predict that this method will achieve better collaboration with humans than the one targeting action prediction.

- Incorporate a partial-order planner. This thesis assumes that the problem's goal is always pre-decomposed into sub-goals. This principle might work with objectives that resemble recipes, where there is a known list of tasks to complete. However, in real-world applications, that is not always the case. We need to make our algorithm understand the goal composition to make it collaborate with humans in a new environment. We suggest the development of a partial-order planner to solve the issues of adapting the model to different problems.

Bibliography

[1] L. Peternel, N. Tsagarakis, and A. Ajoudani, “Towards multi-modal intention interfaces for human-robot co-manipulation,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2663–2669.

[2] R. Brooks, A. Smith, and B. Scassellati, “Foundations for a theory of mind for a humanoid robot,” 07 2001.

[3] B. Scassellati, “Theory of mind for a humanoid robot,” *Autonomous Robots*, vol. 12, no. 1, pp. 13–24, 2002.

[4] A. H. Reynolds. (2019) Convolutional Neural Networks (CNNs). [Online]. Available: https://anhreynolds.com/blogs/cnn.html

[5] S. Saxena, “Introduction to Long Short Term Memory (LSTM),” https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/, 2021.

[6] S. Aoki, C.-W. Lin, and R. Rajkumar, “Human-robot cooperation for autonomous vehicles and human drivers: Challenges and solutions,” *IEEE communications magazine*, vol. 59, no. 8, pp. 35–41, 2021.

[7] J. Holland, L. Kingston, C. McCarthy, E. Armstrong, P. OâDwyer, F. Merz, and M. McConnell, “Service robots in the healthcare sector,” *Robotics*, vol. 10, no. 1, p. 47, 2021.

[8] J. A. Marvel, J. Falco, and I. Marstio, “Characterizing task-based humanârobot collaboration safety in manufacturing,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 2, pp. 260–275, 2015.

[9] R. Inam, K. Raizer, A. Hata, R. Souza, E. Forsman, E. Cao, and S. Wang, “Risk assessment for human-robot collaboration in an automated warehouse scenario,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 743–751.

[10] D. Archer and R. M. Akert, “Words and everything else: Verbal and nonverbal cues in social interpretation.” *Journal of personality and social psychology*, vol. 35, no. 6, p. 443, 1977.

[11] N. Sebanz, H. Bekkering, and G. Knoblich, “Joint action: bodies and minds moving together,” *Trends in Cognitive Sciences*, vol. 10, no. 2, pp. 70–76, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364661305003566

[12] S. M. Carlson, M. A. Koenig, and M. B. Harms, “Theory of mind,” *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 4, no. 4, pp. 391–402, 2013.

[13] C. A. Brownell, “Early developments in joint action,” *Review of Philosophy and Psychology*, vol. 2, no. 2, pp. 193–211, Jun 2011. [Online]. Available: https://doi.org/10.1007/s13164-011-0056-1

[14] P. Mundy and L. Newell, “Attention, joint attention, and social cognition,” *Current directions in psychological science*, vol. 16, no. 5, pp. 269–274, 2007.

[15] A. Frischen, A. P. Bayliss, and S. P. Tipper, “Gaze cueing of attention: visual attention, social cognition, and individual differences.” *Psychological bulletin*, vol. 133, no. 4, p. 694, 2007.

[16] P. Nuku and H. Bekkering, “Joint attention: Inferring what others perceive (and donât perceive),” *Consciousness and Cognition*, vol. 17, no. 1, pp. 339–349, 2008.

[17] J. GrÃ¨zes, J. Armony, J. Rowe, and R. Passingham, “Activations related to âmirrorâ and âcanonicalâ neurones in the human brain: an fmri study,” *NeuroImage*, vol. 18, no. 4, pp. 928–937, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811903000429

[18] E. Kohler, C. Keysers, M. UmiltÃ , L. Fogassi, V. Gallese, and G. Rizzolatti, “Hearing sounds, understanding actions: Action representation in mirror neurons,” *Science (New York, N.Y.)*, vol. 297, pp. 846–8, 09 2002.

[19] G. Rizzolatti, L. Fogassi, and V. Gallese, “Neurophysiological mechanisms underlying the understanding and imitation of action,” *Nature reviews neuroscience*, vol. 2, no. 9, pp. 661–670, 2001.

[20] G. Berntson and J. T. Cacioppo, “Handbook of neuroscience for the behavioral sciences,” vol. 1, 2009.

[21] W. Prinz, “Perception and action planning,” *European journal of cognitive psychology*, vol. 9, no. 2, pp. 129–154, 1997.

[22] N. Sebanz and G. Knoblich, “Prediction in joint action: What, when, and where,” *Topics in cognitive science*, vol. 1, no. 2, pp. 353–367, 2009.

[23] H. Bekkering, A. Wohlschlager, and M. Gattis, “Imitation of gestures in children is goal-directed,” *The Quarterly Journal of Experimental Psychology: Section A*, vol. 53, no. 1, pp. 153–164, 2000.

[24] P. R. Davidson and D. M. Wolpert, “Motor learning and prediction in a variable environment,” *Current opinion in neurobiology*, vol. 13, no. 2, pp. 232–237, 2003.

[25] D. M. Wolpert and J. R. Flanagan, “Motor prediction,” *Current biology*, vol. 11, no. 18, pp. R729–R732, 2001.

[26] G. Knoblich and R. Flach, “Predicting the effects of actions: Interactions of perception and action,” *Psychological science*, vol. 12, no. 6, pp. 467–472, 2001.

[27] D. C. Dennett, “Intentional systems in cognitive ethology: The âpanglossian paradigmâ defended,” *Behavioral and Brain Sciences*, vol. 6, no. 3, pp. 343–355, 1983.

[28] S. Baron-Cohen, “Precursors to a theory of mind: Understanding attention in others,” *Natural theories of mind: Evolution, development and simulation of everyday mindreading*, vol. 1, pp. 233–251, 1991.

[29] A. M. Leslie, O. Friedman, and T. P. German, “Core mechanisms in theory of mind,” *Trends in cognitive sciences*, vol. 8, no. 12, pp. 528–533, 2004.

[30] A. M. Leslie, “Pretense and representation: The origins of” theory of mind.”,” *Psychological review*, vol. 94, no. 4, p. 412, 1987.

[31] K. A. Loveland and S. H. Landry, “Joint attention and language in autism and developmental language delay,” *J Autism Dev Disord*, vol. 16, no. 3, pp. 335–349, Sep. 1986.

[32] T. Vierkant, “What metarepresentation is for,” *Foundations of metacognition*, pp. 279–288, 2012.

[33] P. Mundy and M. Sigman, “The theoretical implications of joint-attention deficits in autism,” *Development and psychopathology*, vol. 1, no. 3, pp. 173–183, 1989.

[34] S. Baron-Cohen, “Joint-attention deficits in autism: Towards a cognitive analysis,” *Development and psychopathology*, vol. 1, no. 3, pp. 185–189, 1989.

[35] M. S. Gobel, H. S. Kim, and D. C. Richardson, “The dual function of social gaze,” *Cognition*, vol. 136, pp. 359–364, 2015.

[36] M. Argyle, R. Ingham, F. Alkema, and M. McCallin, “The different functions of gaze,” 1973.

[37] B. Mutlu, “The design of gaze behavior for embodied social interfaces,” in *CHI’08 extended abstracts on human factors in computing systems*, 2008, pp. 2661–2664.

[38] D. W.-L. Wu, W. F. Bischof, and A. Kingstone, “Natural gaze signaling in a social context,” *Evolution and Human Behavior*, vol. 35, no. 3, pp. 211–218, 2014.

[39] C.-M. Huang, S. Andrist, A. Sauppé, and B. Mutlu, “Using gaze patterns to predict task intent in collaboration,” *Frontiers in psychology*, vol. 6, p. 1049, 2015.

[40] S. Baron-Cohen, *Mindblindness: An essay on autism and theory of mind*. MIT press, 1997.

[41] ——, “The eye direction detector (edd) and the shared attention mechanism (sam): Two cases for evolutionary psychology.” Lawrence Erlbaum Associates, Inc, 1995.

[42] K. A. McCabe, V. L. Smith, and M. LePore, “Intentionality detection and âmindreadingâ: Why does game form matter?” *Proceedings of the National Academy of Sciences*, vol. 97, no. 8, pp. 4404–4409, 2000.

[43] C. Monroy, C.-H. Chen, D. Houston, and C. Yu, “Action prediction during real-time parent-infant interactions,” *Developmental science*, vol. 24, no. 3, p. e13042, 2021.

[44] O. E. David, N. S. Netanyahu, and L. Wolf, “Deepchess: End-to-end deep neural network for automatic learning in chess,” in *International Conference on Artificial Neural Networks*. Springer, 2016, pp. 88–96.

[45] G. Haworth and M. Velliste, “Chess endgames and neural networks,” *ICGA Journal*, vol. 21, no. 4, pp. 211–227, 1998.

[46] Y. Kong and Y. Fu, “Human action recognition and prediction: A survey,” *International Journal of Computer Vision*, vol. 130, no. 5, pp. 1366–1401, 2022.

[47] S. Ji, W. Xu, M. Yang, and K. Yu, “3d convolutional neural networks for human action recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.

[48] G. Gkioxari, R. Girshick, and J. Malik, “Contextual action recognition with r* cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1080–1088.

[49] Y. Kong, Z. Tao, and Y. Fu, “Deep sequential context networks for action prediction,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1473–1481.

[50] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[51] A. Graves and J. Schmidhuber, “Offline handwriting recognition with multidimensional recurrent neural networks,” *Advances in neural information processing systems*, vol. 21, 2008.

[52] I. Sutskever, J. Martens, and G. E. Hinton, “Generating text with recurrent neural networks,” in *ICML*, 2011.

[53] S. Şeker, E. Ayaz, and E. Türkcan, “Elman’s recurrent neural network applications to condition monitoring in nuclear power plant and rotating machinery,” *Engineering applications of artificial intelligence*, vol. 16, no. 7-8, pp. 647–656, 2003.

[54] S. Minami, “Predicting equity price with corporate action events using LSTM-RNN,” *Journal of Mathematical Finance*, vol. 08, no. 01, pp. 58–63, 2018. [Online]. Available: https://doi.org/10.4236/jmf.2018.81005

[55] A. Rasouli, I. Kotseruba, and J. K. Tsotsos, “Pedestrian action anticipation using contextual feature fusion in stacked rnns,” *arXiv preprint arXiv:2005.06582*, 2020.

[56] O. Olabiyi, E. Martinson, V. Chintalapudi, and R. Guo, “Driver action prediction using deep (bidirectional) recurrent neural network,” *arXiv preprint arXiv:1706.02257*, 2017.

[57] “What is ai planning?” https://planning.wiki/guide/whatis/aip, accessed: 2022-09-3.

[58] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, “Pddl - the planning domain definition language,” 1998.

[59] A. Gerevini and D. Long, “Plan constraints and preferences in pddl 3 the language of the fifth international planning competition,” 2005.

[60] A. Coles and A. Coles, “Pddl+ planning with events and linear processes,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 24, 2014, pp. 74–82.

[61] D. S. Weld, “An introduction to least commitment planning,” *AI magazine*, vol. 15, no. 4, pp. 27–27, 1994.

[62] S. Minton, J. Bresina, and M. Drummond, “Total-order and partial-order planning: A comparative analysis,” *Journal of Artificial Intelligence Research*, vol. 2, pp. 227–262, 1994.

[63] A. L. Blum and M. L. Furst, “Fast planning through planning graph analysis,” *Artificial intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[64] S. Kambhampati, E. Parker, and E. Lambrecht, “Understanding and extending graphplan,” in *European Conference on Planning*. Springer, 1997, pp. 260–272.

[65] B. C. Gazen and C. A. Knoblock, “Combining the expressivity of ucpop with the efficiency of graphplan,” in *European Conference on Planning*. Springer, 1997, pp. 221–233.

[66] A. Coles, A. Coles, M. Fox, and D. Long, “Forward-chaining partial-order planning,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 20, 2010, pp. 42–49.

[67] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach, 4th edition*. Pearson, 2021.

[68] T. M. Moerland, J. Broekens, and C. M. Jonker, “A framework for reinforcement learning and planning,” *arXiv preprint arXiv:2006.15009*, 2020.

[69] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[70] A. Barrett and D. S. Weld, “Partial-order planning: Evaluating possible efficiency gains,” *Artificial Intelligence*, vol. 67, no. 1, pp. 71–112, 1994.

[71] B. J. Grosz, “Collaborative systems (aaai-94 presidential address),” *AI Magazine*, vol. 17, no. 2, p. 67, Mar. 1996. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/1223