Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Lifelong Learning of concepts in CRAFT

présenté par:

Nithin Vasishta

a été évalué par un jury composé des personnes suivantes:

Irina Rish,	président-rapporteur
Liam Paull,	directeur de recherche
Christopher Pal,	membre du jury

Mémoire accepté le: .....

Sommaire

La planification à des niveaux d'abstraction plus élevés est essentielle lorsqu'il s'agit de résoudre des tâches à long horizon avec des complexités hiérarchiques. Pour planifier avec succès à un niveau d'abstraction donné, un agent doit comprendre le fonctionnement de l'environnement à ce niveau particulier. Cette compréhension peut être implicite en termes de politiques, de fonctions de valeur et de modèles, ou elle peut être définie explicitement. Dans ce travail, nous introduisons les *concepts* comme un moyen de représenter et d'accumuler explicitement des informations sur l'environnement.

Les concepts sont définis en termes de transition d'état et des conditions requises pour que cette transition ait lieu. La simplicité de cette définition offre flexibilité et contrôle sur le processus d'apprentissage. Étant donné que les concepts sont de nature hautement interprétable, il est facile d'encoder les connaissances antérieures et d'intervenir au cours du processus d'apprentissage si nécessaire. Cette définition facilite également le transfert de concepts entre différents domaines. Les concepts, à un niveau d'abstraction donné, sont intimement liés aux *compétences*, ou actions temporellement abstraites. Toutes les transitions d'état suffisamment importantes pour être représentées par un concept se produisent après l'exécution réussie d'une compétence. En exploitant cette relation, nous introduisons un cadre qui facilite l'apprentissage tout au long de la vie et le raffinement des concepts à différents niveaux d'abstraction. Le cadre comporte trois volets:

- Le sytème 1 segmente un flux d'expérience (par exemple une démonstration) en une séquence de compétences. Cette segmentation peut se faire à différents niveaux d'abstraction.

- Le sytème 2 analyse ces segments pour affiner et mettre à niveau son ensemble de concepts, lorsqu'applicable.
- Le sytème 3 utilise les concepts disponibles pour générer un *graphe de dépendance de sous-tâches*. Ce graphe peut être utilisé pour planifier à différents niveaux d'abstraction.

Nous démontrons l'applicabilité de ce cadre dans l'environnement hiérarchique 2D CRAFT [**6**]. Nous effectuons des expériences pour explorer comment les concepts peuvent être appris de différents flux d'expérience et comment la qualité de la base de concepts affecte l'optimalité du plan général. Dans les tâches avec des dépendances de sous-tâches complexes, où la plupart des algorithmes ne parviennent pas à se généraliser ou prennent un temps impraticable à converger, nous démontrons que les concepts peuvent être utilisés pour simplifier considérablement la planification. Ce cadre peut également être utilisé pour comprendre l'intention d'une démonstration donnée en termes de concepts. Cela permet à l'agent de répliquer facilement la démonstration dans différents environnements. Nous montrons que cette méthode d'imitation est beaucoup plus robuste aux changements de configuration de l'environnement que les méthodes traditionnelles. Dans notre formulation du problème, nous faisons deux hypothèses: 1) que nous avons accès à un ensemble de compétences suffisamment exhaustif, et 2) que notre agent a accès à des *environnements de pratique*, qui peuvent être utilisés pour affiner les concepts en cas de besoin. L'objectif de ce travail est d'explorer l'aspect pratique des concepts d'apprentissage comme moyen d'améliorer la compréhension de l'environnement. Dans l'ensemble, nous démontrons que les concepts d'apprentissage peuvent être un moyen léger mais efficace d'augmenter la capacité d'un système.

Mots clés: Segmentation des compétences, segmentation de démonstration, apprentissage conceptuel, planification, apprentissage par renforcement hiérarchique, apprentissage par renforcement

Summary

Planning at higher levels of abstraction is critical when it comes to solving long horizon tasks with hierarchical complexities. To plan successfully at a given level of abstraction, an agent must have an understanding of how the environment functions at that particular level. This understanding may be implicit in terms of policies, value functions, and world models, or it can be defined explicitly. In this work, we introduce *concepts* as a means to explicitly represent and accumulate information about the environment.

Concepts are defined in terms of a state transition and the conditions required for that transition to take place. The simplicity of this definition offers flexibility and control over the learning process. Since concepts are highly interpretable in nature, it is easy to encode prior knowledge and intervene during the learning process if necessary. This definition also makes it relatively straightforward to transfer concepts across different domains wherever applicable. Concepts, at a given level of abstraction, are intricately linked to *skills*, or temporally abstracted actions. All the state transitions significant enough to be represented by a concept occur only after the successful execution of a skill. Exploiting this relationship, we introduce a framework that aids in lifelong learning and refining of concepts across different levels of abstraction. The framework has three components:

- System 1 segments a stream of experience (e.g. a demonstration) into a sequence of skills. This segmentation can be done at different levels of abstraction.
- System 2 analyses these segments to refine and upgrade its set of concepts, whenever applicable.
- System 3 utilises the available concepts to generate a *sub-task dependency graph*. This graph can be used for planning at different levels of abstraction.

We demonstrate the applicability of this framework in the 2D hierarchical environment CRAFT [**6**]. We perform experiments to explore how concepts can be learned from different streams of experience, and how the quality of the concept base affects the optimality of the overall plan. In tasks with complex sub-task dependencies, where most algorithms fail to generalise or take an impractical amount of time to converge, we demonstrate that concepts can be used to significantly simplify planning. This framework can also be used to understand the intention of a given demonstration in terms of concepts. This makes it easy for the agent to replicate a demonstration in different environments. We show that this method of imitation is much more robust to changes in the environment configurations than traditional methods. In our problem formulation, we make two assumptions: 1) that we have access to a sufficiently exhaustive set of skills, and 2) that our agent has access to *practice environments*, which can be used to refine concepts when needed. The objective behind this work is to explore the practicality of learning concepts as a means to improve one's understanding about the environment. Overall, we demonstrate that learning concepts can be a light-weight yet efficient way to increase the capability of a system.

Keywords: Skill segmentation, demonstration segmentation, concept learning, planning, hierarchical reinforcement learning, reinforcement learning

Table des matières

Liste des tableaux

Liste des figures

Remerciements

I would like to thank Prof. Liam Paull for giving me the opportunity to be a part of his lab, for giving the freedom and space to explore literally every idea that crossed my mind. The amount of patience and support that I received from him is unbelievable. I feel very lucky to have been a part of the REAL lab, Mila, and Université de Montréal. The work atmosphere was filled with friendly and approachable faces, and at no point of time in my stay did I feel lacking for anything.

I would like to thank my dear friends Mohammad Amini, Sarath Chandar, and Andreea Ioana Deac for always being there for me. Mamal and Andreea, for me, were the main catalyst for growth. In so many ways they were an inspiration and made me want to work on myself constantly. Sarath Chandar, for being such a dependable and comforting presence, for being so magnanimous with his time and resources. I feel very lucky to be his friend. I would also like to thank Elizabeth Ericson and James Ough for making our apartment a home, something I would look forward to come back to. James's work ethic and ambition remains an inspiration. I would like to thank Dr. Rami Baz for providing me with a safe and a highly conducive atmosphere to grow and pursue my spiritual practices. What I learnt from him during our interactions are invaluable and something I will cherish forever.

And finally, my parents for all their sacrifices, some which I am not even aware of. They have always encouraged me to pursue my passions, and have never made me feel lacking for anything. It is only through their support that I have the confidence to face any situation that comes my way.

Chapitre 1

Introduction

Most of the tasks in the real world require planning across different levels of abstraction, or *hierarchies*. Let us consider the example of preparing dinner. This task can be broken down into a higher level plan of doing groceries, preparing the cooking utensils, preparing the ingredients, cooking, setting the table, and serving the food. Here, each individual step can be further broken down into a finer level of plan. Doing groceries, for instance, would involve collecting one's wallet and grocery bag, stepping out the house, locking the door, navigating to the grocery store, etc. For humans, this is a very natural way to plan and execute long horizon tasks. If one were to observe carefully, the levels of hierarchy that get the most attention during planning are the ones with higher degree of uncertainty about its transitions, and the ones that are more immediate in terms of execution. Therefore, for such planning to be successful an agent must have the ability to seamlessly switch between different levels of hierarchy and, along with this, have the ability to continually refine its understanding about the environment at each level.

In this work, we focus on continually improving an agent's understanding of the environment across different levels of abstraction. Abstraction can be along two dimensions – state abstraction (related to the environment state and observations) and temporal abstraction (related to the action space of the agent). In this work, we explore the relationship between these two dimensions of abstraction and introduce *concepts* as means to represent and accumulate information about the environment.

Reinforcement learning (RL) has been the framework of choice for developing intelligent agents that can function in complex scenarios [**1, 10, 176**]. Traditional ways of learning and representing information in this setting have been in terms of policies (Def. 3) [**129, 163**], value functions (Def. 2) [**130, 17**], or a combination of both [**119, 75, 60**]. These learned entities, usually represented as parameterised functions, are implicit ways of representing information about the environment. For example, in the case of the Atari2600 game Pong [**18**], the value function implicitly captures the fact that if it lets the ball go past the pad it is controlling, it will incur a negative reward [**130**]. With the help of powerful function approximators like deep neural networks [**103**], these methods have the ability to capture intricate details about the environment even from high-dimensional state spaces.

Recent advances have made it possible to achieve several remarkable feats like outperforming human level benchmarks in Atari2600 games [**130, 12**], mastering landmark strategy games like Go, Chess, and Shogi [**167, 168**], and multi-player real-time strategy games like StarCraft II and Dota 2 [**20, 184**]. It has also fared well in continuous control tasks like solving a rubik's cube with a single hand [**144**], flying autonomous aerial vehicles, [**198**], among many others [**202, 99, 105, 129, 163, 34**]. However, these methods have very low sample efficiency (and therefore high infrastructure cost), and it is highly non-trivial to re-use the learnt information in a different context. These methods also frequently suffer from generalisation issues, training instability, and lack of reproducibility. In certain cases, it is hard to design an appropriate reward function, and therefore learning useful behaviours becomes challenging [**93**]. These problems are exacerbated as environments increase in complexity and when the tasks have dependencies at different levels of hierarchy. However, there are several promising lines of research that work towards overcoming these issues.

Model based reinforcement learning (MBRL) methods work towards learning the model of the environment. This is then used to either plan directly [**13**], or in conjunction with model free methods [**50, 74**]. Depending on how the model is learnt, certain aspects of the information may be transferable. In multi-task settings, for example, training the model against a variety of tasks can make the model robust for transfer [**110**]. One can also specifically design algorithms that extract transferable features from the environment, like successor representations [**15**], skills (Def. 4) [**80, 53, 180**], or priors [**61**] (See section 2.3.1

for details). Compared to model free methods, MBRL methods have higher sample efficiency, and the learnt information is comparatively more reusable. These methods, however, have their own set of limitations. The MBRL solutions are prone to model bias, i.e. depending on the training data and the methodology with which they are trained, they may learn characteristics about the environment that may not correspond to the actual environment dynamics. This can derail the process of planning and execution. These methods, like their model free counterparts, are susceptible to catastrophic forgetting (the phenomenon of overwriting previously learned information while learning new information). As we get access to new information, we usually have to re-learn the model as there is no infrastructure to easily refine previously learnt information. This aspect of the problem is addressed in the domain of lifelong learning.

Lifelong learning (LL) studies the problem of continually refining an agent's understanding about the world and upgrading its capabilities from various sources of information. Advanced LL systems should be able to use the accumulated knowledge to help future learning, discover new tasks, and be able to learn even at the time of deployment [**32**]. Since, extracting and re-using knowledge across different domains is a core component of LL, there is huge emphasis on knowledge extraction and representation techniques. Research in LL has produced some remarkable systems in the domain of Natural Language Processing (NLP) like Lifelong Topic Model (LTM) [**31**], Never Ending Language Learner (NELL) [**128**], Lifelong Interactive Learning and Inference (LILI) [**121**]. Simultaneously, in the domain of RL there have been systems like Horde [**177**]. Majority of the progress in the context of RL, however, have been in closely related domains like transfer learning, multi-task learning, online learning, and meta-learning[1]. RNNs are widely used in the domain of meta learning as they can support two layers of learning – slow learning and accumulation of knowledge that is implicitly stored in network weights, and fast learning and adaption to current task through activation functions [**86, 87, 7, 30**]. Several attempts have been made to increase the scope of these methods through explicit use of memory [**157, 156**]. Significant progress has been made when it comes to few-shot learning with the advent of MAML and related works [**51, 138, 127**]. We explore this further in section 2.3.2. However, in terms of real world deployment there is a

[1]Lifelong Learning is a super-set of these domains, and does not make certain assumptions that the other domains make

lot of work that needs to be done in terms of improving knowledge representation techniques, gauging the correctness and applicability of learnt information, multi-modal learning, among others [**32**].

Another way to address the issue of interpretability and transferability is to understand the core causal mechanisms of the environment. Causal mechanisms describe the cause-and-effect relationship between different entities in the environment. This, in theory, is the most compact and efficient way to represent environment dynamics. Having access to this would greatly assist in making sense of new information and performing optimally in a situation. This aspect of the problem is explored in a principled way with Causal Reinforcement Learning. Current state-of-the-art methods that use causal inference in RL settings are limited to the domain of multi-armed bandits [**114**]. Another popular approach to extract causal relations is explicitly modelling objects and their interactions in the environment [**44, 162, 118, 96, 63, 201, 69, 188, 183, 4, 106, 120**]. These methods drastically improve the sample efficiency of existing RL algorithms and pave the way to scale standard models to more complex problem settings. However, these methods focus more on high-dimensional state representations than environments having sub-task dependencies. We explore Object-Oriented RL and Causal RL further in section 2.3.3.

Although each thread of research makes some much needed progress towards achieving a lifelong learning system, current methods are lacking in several ways:

1) Systems capable of extracting crucial task relevant details are highly data inefficient;
2) There are no reliable ways to continually extract and store information related to the core causal structure of the environment;
3) Generalisability of the extracted information is limited. This is especially true for hierarchical environments with sub-task dependencies.

Humans are able to formalize concepts with very little interaction with the environment and then are able to learn relevant skills through practice. Consider the task of learning to cook from internet videos. Someone who is an unskilled cook can watch a video and attempt to replicate it but may have to practice several times to master certain concepts (e.g., cutting onions). However, on subsequent demonstrations, the concepts that have been

previously learned can be applied easily in new scenarios (e.g. cutting other things). Over time, through repeated exposure to different cooking examples, the learner gradually builds their database of knowledge. This can be used to replicate previously seen situations, and further combining and generalizing from the experience becomes possible. In this work, we introduce the notion of *concepts* as a means to represent and accumulate information about an environment.

1.1. Concepts: Definition and relation to state and temporal abstractions

There are two dimensions of abstraction for any given environment. One dimension is in terms of state, and another is in terms of action. Let us first consider action abstractions for the task of preparing dinner. At the lowest level of abstraction, actions may be represented in terms of motor commands that the agent executes in the environment. For example, "contract muscle X to position Y". Actions at higher levels of abstraction can be represented as a composition of actions from lower levels of abstraction. For example, the action "pick up object" can be decomposed into "place hand on object", "grasp object", "lift hand". Similarly, "do groceries" can be decomposed into "go to the vegetable section", "pick up carrot", "place carrot into trolley", etc. In order for an agent to plan in terms of higher level actions, the probability of success of its lower level actions should be reasonably high and there must be a clear understanding of the state transitions that will take place. In this work, we define *concepts* as a means to continually develop this understanding across various levels of abstraction.

Definition 1. *A concept is a 2-tuple consisting of an abstracted state transition and the conditions required for that transition to take place.*

This brings us to the second dimension of abstraction, which is the state abstractions. The complete state information about the world would include the dinner table, the kitchen, utensils, furniture, vegetables, along with task irrelevant information like texture of the table cloth, height of the TV stand, traffic situation outside, etc. Therefore, when we want to formulate the concept of "cutting a tomato", we must omit information about other objects in the environment and also certain information about the object itself. For example, the

"color" of the "tomato" is irrelevant for the skill of "cutting". The transition can now simply be described as "fresh tomato" to "sliced tomato". The required conditions for this transition would be that "fresh tomato" is on the "cutting board", "knife" is positioned appropriately in "the dominant hand", the "non-dominant hand" is "empty". When these conditions are satisfied and the high-level action of "cutting" is executed, it leads to the formation of "slices of tomato" from "fresh tomato". This kind of abstraction, being on the level of state information is termed as *state abstraction*.

In an ideal scenario, for the concept to be as generalisable as possible, the transition and the conditions should be described at the highest level of abstraction. This would enable the concept to be applicable to a wider range of scenarios. Coming up with the perfect state abstraction for a given concept is equivalent to uncovering the core causal principles about the environment. This is an unsolved problem, one that even humans can get wrong at times. In fact, the field of science exists to precisely uncover such relationships between different entities. In routine real world environments, this is still a very difficult problem. Even though one can formulate concepts with a single example, in order to increase its reliability and generalisability, it is best to gain access to as many examples as possible. Another challenge is to identify the state transitions that are significant enough to be formulated as concepts. Formulating too many concepts would increase the system's redundancy through excess memory usage and optimisation procedures. On the other hand, missing crucial concepts can lead to sub-optimal planning (see chapter 7) and in some cases, an absence of solution (see section 6.2.1).

Humans have a complex memory structure that automatically weighs the importance of real world events, abstracts away irrelevant information based on prior knowledge, and stores the information in an organised way. During real world execution, based on the sensory cues and imagination, relevant experiences can be recalled into the conscious memory and can then be used to plan ahead. This stored information can be routinely updated with experience and can gradually lead to the coherent understanding of a phenomenon. In machine learning, there has been a conscious effort to build such capabilities. Works like Memory Augmented Neural Networks [**157**], Neural Turning Machines [**68**] explore the functioning of memory that is compatible with neural networks. Various kinds of priors have been proposed to

simplify and speed-up the process of learning. These priors can be in terms of key transitions themselves (events) [**29**], high-level actions (skills or behaviours) [**80, 53, 180, 61**], or they can have a cognitive basis to their definition like object representations (section 2.3.3). In this work, we aim to roughly replicate the process of human learning using priors. We propose a rudimentary yet effective way of identifying crucial transitions and use that to accumulate relevant information.

Let us first examine the relationship between skills (or temporally abstracted actions) and events. Events can be roughly defined as state transitions that are significant enough to be represented as concepts. One can note that *all the events that occur at a particular level of abstraction, happen only after the execution of a skill at that level of abstraction*. Therefore, the process of improving one's understanding at a particular level of hierarchy can be greatly simplified if we take the preliminary step of identifying the end of skill executions. We use this observation as the basis to design our framework, which has three components:

- System 1: Experience Segmentation
- System 2: Lifelong concept learning
- System 3: Graph based planning

System 1 considers the task of splitting a given stream of experience into useful segments, which are used as inputs to the other components of the framework. The stream of experience may be in the form of expert demonstration or experience collect by the agent itself. In the example of learning to cook from internet videos, our agent would look at a video of a human cooking and split that into segments such as cutting onions, placing pan on the stove, stirring the pot, etc. Assuming that the priors available in terms of an initial set of skills is exhaustive enough, it reduces System 1's function to simply identifying the end of skill execution.

System 2 tackles the problem of formulating concepts using the segments from System 1. The agent continually learns the set of concepts intrinsic to the environment, which can be used for higher level reasoning and planning. It uses a set of practice environments (Def. 9) to practice skills and refine its concepts to whatever extent it can. Segments from System 1 make it easy for System 2 to function, as the system only needs to analyse the states at the point of segmentation.

System 3 uses the concepts learned via System 2 to form a sub-task dependency graph that is specific to the current situation and goal. This drastically simplifies the process of planning and enables the agent to handle obstructions, different initial conditions and thus, effectively move about the environment. Given a demonstration of a task, System 3 also outputs a reward vector summarising the intention of the demonstrator. This, along with the sub-task dependency graph, drastically simplifies the problem of imitating the demonstration across various environment conditions

The success of System 1 depends on the exhaustiveness of the set of skills that the agent possesses. If the set of skills is diverse enough, it can adequately segment any stream of experience into a sequence of skills[2]. Since all changes in the environment are the result of successful executions of skills, System 2's task of formulating concepts from experience becomes simple. The accuracy, completeness, and compactness of the learnt set of concepts, in turn, determines the quality of the overall framework. Being a lifelong learning system, the effectiveness of the system only increases with exposure. After a decent level of maturity, the agent can use the learnt concepts to effortlessly learn from demonstrations and plan across complex hierarchical environments.

We demonstrate the applicability of this framework on the 2D hierarchical environment CRAFT **[6]**, where it is easy to construct configurations tasks that are dependent on the successful execution of several other tasks. We assume access to a set of initial skills diverse enough to learn the necessary concepts[3]. We also assume access to a sufficient number of *practice environments* (Def. 9), where the agent can practice different skills to discover and refine concepts. The main motivation behind the use of prior knowledge is the reduction in effort needed to upkeep the agent. Our priority lies in the ease with which the agent is able to adapt to new situations, discover and refine concepts, and recover from failures. We want the agent to achieve this with minimal training data and interventions. Given a mature set of concepts, we show that this framework requires *a single demonstration*, to uncover the

[2]One can note that getting access to an initial set of skills as a prior is easy. Since there is no restriction on the way we want to define and implement skills, all the policies and techniques from the world of classical robotics become available to use

[3]In this work, we don't work upon improving the skill set. In an ideal scenario, both these aspects are improved upon concurrently, but we assume that the set of skills are fixed, and don't need improving.

motivation of the demonstrator, and replicate the demonstration in different instantiations of the environment.

The components of the framework bear similarity with the three rungs in the "Ladder of Causation" [**148**] – association, intervention, and imagination. System 1's main function is to *perceive* and identify important junctions in a given stream of experience. System 2's function is to build concepts or understand the causal mechanisms. It aims to refine and understand the concepts by executing various skills in practice environments which is equivalent to performing various interventions. And finally System 3 builds a sub-task dependency graph according to the given situations using the concepts it understands about the environment. This resembles the counterfactual planning aspect in the "Ladder of Causation".

1.2. Design considerations

1.2.1. Streams of Experience - Demonstrations and Self-Play

A stream of experience refers to the sequence of states or observations that the agent encounters. It can be expert demonstrations from various points of view, or it can be states that the agent itself encounters during exploration. In complex hierarchical environments or hard exploration environments, it is hard to encounter all the critical states through random sequencing of skills and actions. Demonstrations, therefore, are a convenient way gain access to relevant information.

There are cases where it may be hard to obtain demonstrations, like unexplored or unsolved environments where there are no optimal controllers, or environments with high degrees of freedom (DoF) where it is not intuitive for experts to give guidance. In these cases, self-exploration is the only way. In most cases, however, demonstrations are either readily available or the cost of obtaining them is relatively cheap. Demonstrations obtained may be from a different point of view or may have different environment configurations than what the agent is familiar with. We can tackle this through various domain adaptation techniques [**195, 92**]. In our work, we consider demonstrations to be partially observable. In the case of CRAFT, the internal state of the agent, which represents the items gathered so far, is

not visible. But the data gathered using practice environments via self-exploration is fully observable.

1.2.2. Choice of environment

The objective of our work is to provide a potentially complete solution to tasks with hierarchical dependencies. Thus one of the foremost criteria is that the tasks should have non-trivial dependencies, which are easy to understand from a human perspective. It should be easy to setup tasks of varying levels of difficulty and arbitrary length. The environment should be close to real world scenarios without having too many complications (e.g. high dimensional state space) that can be dealt with separately. The emphasis is on demonstrating the agent's ability to learn the core concepts of the environment either from demonstrations or self-exploration, which can be utilised in different instantiations of the environment.

We found the environment CRAFT to be appropriate for our use (See section 3.2 for a detailed description). An example of hierarchical task dependency in this environment is as follows: in order to break a "stone", you need an "axe". To make an axe you need to combine "stick" and "iron" at "workshop #1". To make a stick, you need to visit "workshop #0" with "wood". These dependencies can be different for different initial conditions and environment configurations, and task length can be arbitrarily long. So unless the agent understands the core concepts related to the objects in the environment, it can be impossible to solve certain tasks.

1.2.3. Contributions

We summarize the claimed contributions of this work as follows:

(1) We introduce the notion of concepts as a means to continually improve an agent's understanding about the environment at various levels of abstraction
(2) We introduce a framework to continually extract relevant concepts based on the relationship between skills and the concepts

(3) We demonstrate the applicability of this framework in the 2D hierarchical environment CRAFT, where the framework is used to systematically segment various streams of information, update concepts with ease, and form sub-task dependency graphs that help in hierarchical planning

Concepts are a simple way to represent information about the environment across various levels of hierarchy. These are interpretable, easy to update, and easy to transfer to related tasks and domains. The framework, on the other hand, provides an intuitive way to continually gather concepts from various streams of experience. The framework divides the problem of planning in a way that that eases the functioning of its individual components to tackle complex hierarchical tasks. Each component can be individually updated to reflect the current state of the art and can be easily added-on to different systems of hierarchical planning without incurring much additional cost.

Chapitre 2

Related Work

We divide the literature survey as follows: We first discuss the two major streams of experience generally available to an agent: self-exploration and demonstrations. Then we delve into the literature associated with each of the three sub-systems individually. For System 1, we mention existing works that learn to segment from demonstrations, along with systems that attempt to learn the most effort-consuming prior – an exhaustive set of skill. For system 2, we discuss relevant literature in model based RL, causal RL, and lifelong RL with a specific emphasis on knowledge representation. Finally, for System 3, we talk about different methods that can be used for hierarchical planning with information represented in terms of concepts.

2.1. Di□erent streams of experience

A stream of experience, as we discussed in chapter 1, refers to a sequence of states or observation that the agent encounters. This can be obtained via expert (or sub-optimal) demonstrations of tasks, or via self-exploration.

Exploration is an important aspect of an intelligent system. Proper exploration leads to discovering of crucial information, and helps the agent effectively navigate the environment [**124**]. Exploration alone, however, may not be fully sufficient to uncover all the core concepts of the environment. Even if it is sufficient, it may take a lot of time and effort. Demonstrations, therefore, can be effectively leveraged for this purpose. The topic of learning from

demonstrations has been a focused topic of research in this community for several decades [**8**] [**91**]. Since demonstrations, both optimal and sub-optimal, are readily available for various environments in most cases, or the cost of obtaining one is relatively negligible, this significantly increases the information available for use. In this section, we discuss different methods present in the literature.

2.1.1. Learning via Exploration

The goal of exploration is to learn about the crucial aspects of the environment as quickly and effectively as possible. Preliminary methods include ϵ-greedy exploration or adding Gaussian noise to the controls. More recently, there are works that aim to systematically encourage exploration by providing reward bonuses. Houthooft et. al (2016) [**89**] encourages exploration by having the agent maximize information gain about its belief about the environment dynamics. Recent works have used the notion of curiosity to formulate an exploration signal (We direct the reader to Burda et. al (2018) [**25**] for a systematic review of different methods using curiosity). Sahni et al. (2017) [**146**] tackles the problem of exploration using curiosity along with dynamics model building in a self-supervised manner. In spirit, this is similar to the direction we are proposing and can be thought of as a combination of System 1 and System 2. In Burda et. al (2018) [**26**], the exploration bonus can be given in terms of the error of a neural network predicting features of the observations given by a fixed randomly initialized neural network. This work manages to achieve the state-of-the art performance on Montezuma's revenge, circa 2018.

Count-based exploration methods form a significant component of recent research. Bellemare et. al (2016) [**19**] proposed an algorithm to derive pseudo-count from an arbitrary density model, which is used as an exploration bonus. Ostrovski et. al. (2017) [**145**] extended this method to a more practical and general algorithm by using Pixel CNNs. Tang et. al. (2016) [**179**] suggests a method to generalise classical count-based methods to high-dimensional state spaces, by mapping states to hash-codes. Fu et. al. (2017) [**59**] describes how discriminative modelling using exemplar models corresponds to implicit density estimation, which can then be combined with count-based exploration.

One can simultaneously improve other aspects of the system during exploration, like building reliable world models (see section 2.3.1), or learning a set of skills or options [**146, 49, 71, 2**] (see section 2.2.1). We can see that each objective would support the fulfilment of the other. Learning a good set of skills or model improves the effectiveness of the exploration, and effective exploration improves the probability of finding useful skills and the quality of the model learnt.

2.1.2. Learning from Demonstrations

Demonstrations can be an effective means to gather information about the environment. The demonstrations in most cases are readily available and are relatively cheap to get access to. Even in the case of demonstrations, there can be different levels of supervision.

In “supervised learning" this is achieved by providing a sufficiently large labeled dataset that the agent can learn the mapping from the input space (in this case the state of the environment or an image) to an output space (the actions required to achieve a goal). This approach in the context of agent learning is often referred to as “behavioural cloning" [**126, 136**]. While appealing in its simplicity, it has the drawback of poor generalization in the case where the agent is presented with new tasks or when it deviates from the optimal action even slightly. This is a result of the fact that sequential decision making process violates the “i.i.d" (independent and identically distributed data) assumption of the training and testing data that is required for supervised learning.

A better approach is to learn a policy that is robust to deviations by being able to converge back to the optimal plan. In “imitation learning", or direct policy derivation, the demonstrator and the learner execution is interleaved so that the learner may make mistakes and then be corrected by the expert [**151, 153**]. However, this still requires significant time for the expert who is now explicitly inside the learning loop. Another alternative approach is inverse reinforcement learning, or indirect policy derivation, where the agent learns a reward signal from the demonstrations and then learns a policy using reinforcement learning [**9**]. Unfortunately, this tends to be even more inefficient in terms of expert demonstrations since learning the rewards signal is actually ill-posed (the reward is under-specified).

In our approach, we propose to increase the efficiency of learning from expert demonstrations through the use of a hierarchical decomposition that roughly corresponds to learning the underlying causal mechanisms of the world. This approach removes the expert from the learning loop as it is in the case of imitation learning, and is drastically more efficient than inverse reinforcement learning.

2.2. System 1: Automatic Skill Segmentation

When we consider the problem of learning concepts specifically, it doesn't matter where the experience came from, as long as it can be converted into a format easy to understand by the agent. As described in Chapter 1, all the crucial changes in the environment happens after the execution of a skill. Therefore, the ability of an agent to segment a stream of experience into a sequence of skills becomes very important. This ability allows us to individually examine each segment and its effect on the environment, thus making the problem of credit assignment a lot easier. Several works combine demonstration segmentation with skill learning [**98, 166, 139, 104, 55, 101**], each requiring different forms of prior knowledge.

Kipf et. al. (2018) [**98**] introduce ComPILE, which decomposes the demonstration and encodes each segment with a latent representation using a VAE-like encoder. The corresponding latent conditioned decoder is then used as skill. Shiarlis et. al. (2018) [**166**] also produces a similar output, with the difference being this also requires ground truth label of sub-task sequences as input.

Skill segmentation has also been explored with real world robots [**139, 104**]. Although the data used by the models are low-dimensional and require either task specific priors or hand-specified features, they provide a theoretically complete framework for learning from decomposable demonstrations. Both use variants of auto-regressive HMMs to segment the demonstration and form sub-task dependency graphs. In both cases, the task graph allows for self regulation during testing. Niekum et. al. (2013) [**139**] further allows for fine-tuning through real-time user interactions, while Kroemer et. al. (2015) [**104**] requires only two demonstration to form a task graph.

Fox et. al (2017) [**55**] works on discovering options at multiple levels of hierarchy from demonstrations. It recursively computes the hierarchy of options, and learns the meta-controller at each level, thereby reducing the complexity of long horizon tasks significantly. This is followed up with Krishnan et. al. (2017) [**101**], which removes the need for users to specify the options, and extends the method to continuous control control spaces. Although these methods can deal with high dimensional inputs like images, the tasks dealt with do not have hierarchical dependencies. The transferability of the learned options and meta-policies also has not been explored.

Xu et. al (2017) [**194**] propose Neural Task Programming (NTP), which forms a bridge between few-shot learning from demonstration and neural program induction. NTP takes as input a task specification (e.g., video demonstration of a task) and recursively decomposes it into finer sub-task specifications. These specifications are fed to a hierarchical neural program, where bottom-level programs are callable subroutines that interact with the environment. NTP learns to generalize well towards unseen tasks with increasing lengths, variable topologies, and changing objectives. On a similar stride, Huang et. al. (2018) [**90**] takes in a demonstration and initialises a hierarchical policy. Their method involves creating neural task graphs, which serves as an intermediate representation before initialising the hierarchical policy. These methods have been shown to work well in real world tasks with minimal hierarchical dependencies.

All the works cited in this section consider full state information and action sequences as demonstrations. Our work, along with Sun et. al. (2019) [**174**] is able to deal with demonstrations comprising of sequence of observation (partial state information). We note that several domain adaptation techniques can be applied to utilise demonstrations from different points of view or operating under different environment configurations [**195, 92**].

Option learning from demonstrations

Options [**178**] is a well-established framework for long-horizon tasks and hierarchical environments. There is a one to one correlation between options available and the set of skills, concepts, and events in our framework. The conditions in the each concept is related to the initiation set of an option, the skills necessary to accomplish an event can represent the

intra-option policy and the final transition, when achieved would satisfy the termination condition. Decomposing the problem in terms of the initial set of skills, and providing a framework for the lifelong learning of concepts is equivalent to providing a complete set of intra-options skills, and learning concepts for creating new options given an environment. In other words, we are replacing the option learning with concept learning. We argue that is a much simpler problem as events and concepts can be much easily related to the set of skills that the agent possesses. If we were to look at System 3 in terms of options, it uses the gathered knowledge about the environment to identify the set of options that are possible and plan. Thus, we can replace a sub-task dependency graph based planner with an options-based policy by representing the learnt information differently.

There are several methods, both supervised [**66, 48, 100, 166, 173**] and unsupervised [**79, 56, 55, 101, 165**], that learn options directly from demonstrations. Specifically in unsupervised learning approaches, "Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets" [**79**] uses a single optimisation objective to segment demonstrations and learns skills simultaneously. However, in this work there are no hierarchical dependencies between the sub-tasks and the problem of scaling to higher levels of hierarchies is not considered. Parameterised hierarchical procedures for neural programming [**56**], considers the problem of learning multi-level hierarchical neural programs. This paper is an extension to [**55**] and [**101**]. However, they experiment in algorithmic domains, and robotics/RL tasks haven't been considered. Directed info-GAIL [**165**] proposed a principled way using information theory to segment demonstrations. The tasks considered, however, have comparatively shorter horizons, and are less compositional in nature.

2.2.1. On relaxing the initial skill set assumption

In our work, we assume access a set of initial skills that is used by System 1 to segment demonstration, and by System 3 for the purpose of planning. The exhaustiveness of the skill set determines the effectiveness of the overall framework. In this work, we assume that this initial set of skills is exhaustive enough and do not update this with experience. We note that learning and updating the set of skills along with the set of concepts would be the ideal

way to approach this problem. In this section, we explore different methods in the existing literature that can be used to relax this assumption.

There are works that attempt to learn such a set of skills from scratch. Eysenbach (2018) [**49**] specifies an exploration objective based on information theory that forces the skills learnt to be as diverse from each other as possible. Florensa et. al (2017) [**53**] learns skills using a stochastic neural network which allows for flexible weight sharing among different policies. The diversity of skills is encouraged using an information theoretic objective, similar to Eysenbach et. al (2018) [**49**]. Other work on transfer in RL has focused on learning reusable skills e.g. in the form of embedding spaces [**80, 53, 180**].

A suitable set of skills can also be learnt either using demonstrations [**5**] or while optimising for performance in a multi-task setting [**58, 6, 143**]. Andersen et. al (2018) [**5**] proposes to learn a suitable set of skills from demonstration, using latent space factorization and transition state clustering [**102**]. However, it does require access to the environment and reward signals.

Co-Reyes et. al (2018) [**37**] propose SeCTAR, which learns a latent-conditioned policy and a latent-conditioned model, that are consistent with each other. Therefore, one can use the latent-conditioned model to predict the sequence of latent codes that can achieve the required outcome for a given task. These latent codes are subsequently used to initialise policies that interact with the environment to fulfil the required objective. This work presents another method to simultaneously learn a set of policies along with the model of the environment. Although latent variables are relatively more transferable and interpretable, it non-trivial to continually improve the overall model with experience. Compared to our work, where we use unlabelled streams of experience to facilitate learning, they assume access to final task objectives.

Option Critic [**11**] explores the aspect of using deep neural networks as a basis for learning options from scratch. However, options learnt may not always be useful. In this work, they show instances where multiple options do the same thing, or where a single option attempts to learn the entire task without any hierarchical decomposition. Although such failure cases

occur frequently, this is a significant step in learning to uncover hierarchy in the environment from scratch.

Learning a default behaviour, or base skill, (Def. 7, see section 4.1) in an environment can be of tremendous use. Several behaviours that an agent typically learns in the environment may be represented as a slight deviation from the default behaviour. In dynamic environments like Freeway or Seaquest [**18**], the agent is always executing a skill just to survive the episode. In such cases, a significant portion of the demonstration can be represented as segments of the default behaviour. Therefore, learning default behaviours and learning skill discriminators to identify such behaviour becomes critical. Several works in the recent literature aim to learn default behaviours from scratch [**61, 180, 65**]. Goyal et al. (2019) [**65**] uses an information bottleneck objective to learn a base policy that is generally optimal for all possible goals. This can be useful for distinguishing unique behaviours the agent may be undertaking, and thus, for segmenting the demonstration.

2.3. System 2: Continual concept learning

In this section, we first explore model based RL methods and the types of concepts they are able to extract from the environment. Then we move on to meta learning and lifelong Learning (LL) methods, where the concepts are learned with an emphasis on re-usability. We also examine the system architectures that enable lifelong learning. Then, we look into causality based methods in RL, where there is a formal attempt to extract the causal relations in the environment. In principle, understanding the causal structure of the environment is the most compact and complete way to model the environment.

2.3.1. Model Based Methods

Model based reinforcement learning (MBRL) methods work towards learning the model of the environment and then use this to either plan directly [**13**], or work in conjunction with model free methods to improve their sample efficiency [**50, 74**]. The model of the environment may be learnt in terms of future prediction models [**72, 141, 33**], expected value functions [**50, 176**], or they can be a little more explicit in nature like latent space

approximation functions [**74, 142, 57, 77, 62, 70, 187, 76**]. In several works, the model learned is implicitly embedded in the weights of an RNN [**169, 190, 189**]. Ha et. al. 2018 [**74**] give a nice background of model based methods evolving from basic neural networks [**191, 133, 152, 192, 137, 158, 159, 160**] and probabilistic methods [**41, 122**] to their more recent counterparts involving deep neural networks [**67, 73, 47, 161, 189, 190, 169, 42, 123**]

Most of the work in MBRL can be categorised as either improving the accuracy and usefulness of the model, or improving the interpretability and transferability of the learned concepts. The latter aspects are important when it comes to working with humans and equipping the agent with continual learning capabilities. Depending on how the model is learnt, certain aspects of the information may be transferable. In multi-task settings, for example, training the model against a variety of tasks can make the model more robust for transfer [**110**]. One can also specifically extract transferable features from the environment, in terms of successor representations [**15**], skills [**80, 53, 180**], or priors [**61**].

A common way to tackle high dimensional state spaces is to extract the crucial aspects from the environment by learning a latent space representation [**74, 142, 57, 77, 62, 70, 187, 76**]. DeepMDP [**62**] can be viewed as a formalization of such methods and also provides theoretical guarantees for the quality of the learned latent space representation of the state space and the environment model. There is a possibility that features irrelevant to the tasks are learned and the relevant ones are missed. This can be alleviated in Universal Planning Networks [**172**], which learns to extract task specific features along with the model of the world in the inner loop while optimising to imitate the expert in the outer loop.

Recent works have made progress in specifically modelling individual entities and their local interactions as opposed to a global model of the world. This is under the hypothesis that this would lead to more sample efficient way to learn about the world and increase the generalisation capability to novel scenarios. In a recent example, Object-centric perception, prediction, and planning (OP3) [**183**] works with an interactive inference algorithm to bind information about object properties to the entity variables. This work shows that individual entity abstraction can help the system generalise to novel environment configurations.

This system also achieves two to three times better accuracy than a state-of-the-art video prediction model that does not exhibit entity abstraction.

Recent methods have combined model free RL techniques to increase the final asymptotic performance of the overall system while retaining the sample efficiency of model based methods [**135, 13, 88, 149, 107, 186, 50**]. And there are several threads of work aiming to reduce model bias and distribution mismatch [**197, 200, 193, 24, 95**]. These techniques hold promise to improve the current capability of model based methods to learn crucial concepts. More than just improving upon the sample inefficiency of model free methods, there is an expectation that the learnt model can be used for different tasks in the same environment and may even be extrapolated to different environments. Most methods, however, learn concepts implicitly, which is great for general applicability of the method, but affects the ability to reuse learnt concepts. We explore this further in the sub-section on Meta and Lifelong learning (section 2.3.2).

2.3.2. Meta and Lifelong Learning in RL

Meta-learning generally refers to learning an effective set of prior knowledge that makes the task of future learning and adapting to situations easier. This can be considered as a sub-set of lifelong learning, where the goal is to continually refine an agent's understanding about the environment and improve its capabilities. Learning a good set of concepts effectively initialises the framework for single shot imitation of the demonstration and planning in different environment instances. So in a way System 2 can be seen as performing a meta-learning update. If one were to simultaneously update the set of the skills either by adding new skills or composing old ones, that could also be considered as a meta-learning update. Learning to continually make these updates such that the task of learning new concepts and refining skills becomes easier with experience, upgrades the problem from a meta-learning setting to a lifelong learning setting. Works in these domains can be classified in terms of knowledge representation, knowledge extraction techniques, and the amount prior engineering it takes for the system to start the process of continual learning. We direct the reader to Vanschoren (2018) [**182**] for an in-depth survey.

In several works, information is extracted and stored implicitly. RNNs are widely used for the purpose of meta learning [**86, 87, 7, 30**], where the learning happens at two stages – rapid adaptation within a task, and gradually learning across a series of tasks. Works are built on top of this concept that consider neural networks with explicit memory architectures like (NTMs) [**68**] and memory networks [**38**], under the premise that this would help in scenarios where there is a sudden influx of new information [**156, 157**]. Meta Networks is a recent addition to this class of methods, which introduce a new meta-learning algorithm that significantly improves upon the sample efficiency in comparison [**132**].

The field of meta learning has gained tremendous attention with the advent of MAML [**51**]. This popularised the idea of learning an initialisation of model parameters such that the model is able to adapt to the new task with just a few examples. This was an improvement from the previous state-of-the-art in few-shot learning, where the LSTM[**86**]-based meta-learner aimed at learning an update rule training a neural network learner. Nichol et. al (2018) [**138**] obtain an algorithm similar to the first order approximation of MAML, by applying the Shortest Decent algorithm [**84**] in a meta learning setting. This method improves the scalability of meta learning algorithms by reducing the computation and memory usage. But these methods do not consider the problem of refining and utilising knowledge in a lifelong learning setting. Mishra et. al (2017) [**127**] propose a class of meta-learning architectures that use temporal convolutions and soft attention to aggregate information from past experience and pinpoint specific pieces of information. The applicability of this framework has been demonstrated in both supervised and reinforcement learning domains.

Meta-learning from the perspective of Reinforcement Learning has also been explored through the use of Guided Policy Search [**115**] to meta-learn an optimisation algorithm [**116, 117**]. Dual et. al (2016) [**46**] represent the reinforcement learning algorithm using an RNN, utilising its property of slowly incorporating task related meta information with learning of weights while quickly adapting to the given situation by storing information in the activations of the RNNs. The method has been shown to work in both small scale settings like the multi-armed bandits and finite MDPs, along with tasks involving high-dimensional information like vision based navigation. A similar method has been proposed in parallel by Wang et. al (2016) [**185**] that focuses on structured tasks like dependent bandits. Although

the process of adapting to new tasks is highly sample efficient, the meta-training process is highly data consuming. To resolve this problem, Menconda et. al. (2019) [**125**] proposed "guided meta policy search", which leverages expert demonstrations in a nested optimisation loop to reduce the sample complexity of the meta learning algorithm. Another approach to increasing sample efficiency is to disentangle the problem of task inference and control [**150**]. Through the use of online probabilistic filtering of task variables, one can infer how to solve a new task from limited data and integrated with off-policy RL algorithms leading to efficient meta-training and adaptation.

Meta learning has also been explored in the context of imitation learning. Since it requires an impractical amount of data to learn complex vision-based tasks from demonstrations alone, meta-imitation learning aims to reduce the need for data by leveraging experience from previous tasks [**45, 52, 94**]. It is interesting to note that the prior knowledge learned from previous tasks is represented as skills [**52**]. James et. al (2018) [**94**] learn a task embedding from demonstration that helps it to learn a new task from a single demonstration. With the help of domain randomisation techniques, this has shown promising results in sim-to-real transfer. Although these methods demonstrate significant reduction in sample complexity, the training data required to adequately train these models are very specific and in some cases, tedious to obtain. Zhou et. al. [**199**] propose a method that combines the benefits of the trial-and-error method of learning in RL to scale to a broader distribution of tasks, and improve the accuracy and sample efficiency of meta-imitation learning methods. On a similar thread, Yu et. al (2018) [**195**] incorporate demonstrations that are not directly available for use by incorporating domain invariance during meta-training. This allows them to adapt to human demonstrations from different point of views and environment configurations in a one shot fashion. These methods take a significant step in learning adequate priors that increases their effectiveness on tackling new tasks. Although these methods can successfully handle high dimensional state spaces, their effectiveness in domains with more complex causal mechanisms are yet to be tested.

Under the context of zero-shot learning, there have been related streams of research that work towards learning a fundamental aspect about the tasks in the training data, such that they are able to function in test tasks without any additional training. Juhnyuk et. al

(2017) [**143**] train a system to function in a multi-task setting by learning sub-policies to generalise to natural language instructions. The system is able to function in tasks with longer and never-seen-before instructions, while accounting to interruptions, simply because it learnt to relate the instructions to the sub-policies it possesses. DARLA [**83**] proposes a multi-step training procedure where the agent first learns to disentangle different aspects of the environment, then learns to act using the disentangled features. This allows the agent to adapt different configurations of the domain without additional training. These works are related to research previously described in the context of meta-learning, except that these method rely on either a certain amount of prior knowledge to extract the information crucial for zero-shot adaptation or the problem setting is specifically designed for such algorithms to thrive.

Meta learning has been explored in the context of model based reinforcement learning (MBRL). One argument for model-based RL is the potential of transfer to tasks in the same environment. Sutton et. al (1990) [**175**] discuss early examples of this type of transfer on simple problems. Recent works have demonstrated that meta-learning a predictive dynamics model can help the agent quickly adapt to related but novel task configurations [**35, 134, 27**]. This has been demonstrated in the context of vision based robot manipulation with novel reward structures and visual distractors [**27**], continuous online adaptation in continuous control tasks on both simulated and real-world agents [**35, 134**]. As described in the previous section on MBRL methods (Section 2.3.1), learning re-usable skills can be considered as a meta-learning step. The re-usable skills can be learnt in the form of embedding spaces [**80, 53, 180**], successor representations [**15**], meta-learned priors [**61**] or meta-policies [**51, 36**]. Meta-policies can either learn to opt for different models based on the environment requirements [**36**] or quickly adapt its parameters to the new environment setting [**51**]. As with previous approaches in meta-learning, the ability to continually update the learned information is lacking. These methods would suffer from the usual problems of catastrophic forgetting (phenomenon of overwriting previously learned information while learning new information) and capacity saturation (running out of memory to store newly learned information, this is also a frequent cause of catastrophic forgetting) if applied in the context of lifelong learning.

Rusu et. al (2016) [**154**] tackle the problem of continually learning task related information by introducing the architecture of progressive nets. The network makes use of lateral connections to incorporate new information as an add-on to previously learned information. By progressively adding new layers while having the previous layers frozen, the architecture manages to avoid the problem of catastrophic forgetting. This also makes incorporating new information less time consuming and adapting to new situations more sample efficient. However, a downside is that the parameters of the neural network grows exponentially with experience. As we discuss in the section of Causal Reinforcement Learning (Section 2.3.3), in order to have general lifelong learning capabilities without exponential increase in storage capacity, it is important to represent knowledge in terms of learning the core causal mechanisms of the environment.

The ideas presented in this section help an agent improve its generalisation capabilities and efficiency by re-using learnt information. The majority of the research work in this area is focused towards reducing the need for labelled data, or domain specific knowledge. This is usually compensated with an increase in the cost of training, or the requirement of carefully crafted training data. Our method deviates from this approach by assuming access to domain specific knowledge like the initial skill set and state embedding functions. Instead we focus on the quality and the re-usability of the information extracted. The concepts learnt using our framework are much more fundamental to the environment. Therefore, the information retained consumes much less storage space, is transferable across a wider variety of situations, and can be easily updated as the agent gets access to more data. Our framework can also be easily adapted to work with different streams of experience with varying levels of information.

2.3.3. Causal Reinforcement Learning

Understanding the underlying causal structure of the environment can lead to a dramatic increase in the agent's performance in terms of generalization. It can reduce the search space during execution, increase the efficiency of the way information is stored, ease of understanding the objectives of demonstrations, among many other things. Everything the agent can do would generally be enhanced with the understanding of the underlying causal mechanisms.

The existing literature on learning core causal mechanisms can be roughly divided into two categories. One line of research follows the ideas mentioned in Lake et. al (2017) [**109**]. This work encourages the development of causal models as a means to gather knowledge about the environment and emphasise the need for prior knowledge that is grounded in intuitive physics and intuitive psychology. The majority of the recent works that are built from this foundation focus on improving object representation techniques (as a component of intuitive physics) as a means to bridge the gap between human and machine reasoning. For example, Davidson et. al (2020) [**40**] found that providing the Rainbow model [**82**] with simple, feature-engineered object representations substantially boosts its performance on the Frostbite game from Atari 2600. The other line of research works towards bridging the gap between causal inference [**147**] and reinforcement learning.

The importance of object representation has long been recognised in the RL literature. Agnew et. al (2020) provides a nice overview about the growth of this line of research [**3**]. Diuk et al. (2008) [**44**] proposed OO-MDP (Object oriented MDPs) as a means to represent the environment, which makes interactions related to different objects the main focus of the environment. This framework was extended in Scholz et al. (2014) [**162**] to include physics models of object dynamics as priors, and Li et al. (2017) [**118**] investigate integrating object information into modern deep learning approaches. Kansky et. al (2017) [**96**] introduce Schema networks, which demonstrate the ability of learning models of object interactions based on intuitive physics to generalise to drastically different configurations of the Breakout [**18**] without additional training. This work is an improvement from Interactive Networks [**16**] and the Neural Physics Engine [**28**] as latent physical properties and relations need not be hard-coded, and planning can make use of backward search, since the model can distinguish different causes. However, all of these techniques require environment specific object labels.

Methods have been proposed to learn the model of the environment with a focus on object representations and their interactions in an unsupervised way [**63, 201, 69, 188, 183, 4, 106, 120**]. Watters et. al (2019) [**188**] follow a four module decomposition – the vision module, transition model, exploration policy, and the reward predictor. The first three modules are similar to the three components of our framework. These modules are trained

during an unsupervised exploration phase without rewards. The fourth module, however, is trained during a subsequent task phase. In our specifications, the function of the reward predictor is performed by the third system itself, which has access to the concept base from the second sub-system. Veerapaneni et. al (2019). [**183**] tackle the key technical challenge of grounding entity representations to actual objects in the environment by framing this variable binding problem as an inference problem. Interactive inference algorithm that the system uses works on the basis of temporal continuity and interactive feedback to bind information about object properties to the entity variables.

Overall, these methods improve the sample efficiency of existing RL algorithms by orders of magnitude and show promise for a reliable way to build scalable models of the environment. In our problem setting, since we consider low dimensional state spaces, the problem of object representation is trivialised. The concepts we build are more related to object utilisation and conversion into newer entities. The problem setting we consider is more suited towards the challenges posed by tasks with long range hierarchical dependencies.

Other than these works, there are several works that consider the problem of concept learning. However, the terminology of concept is used differently in each case. In Hay et. al (2018) [**81**], the concepts are represented in terms of different interdependent behaviours, while Lake et. al (2015) [**108**] and Lazaro-Gredilla et. al (2019) [**112**] represent concepts as programs. Lake et. al (2015) [**108**] consider simple programs that represent strokes of an alphabet and can be learnt from very small number of examples. On the other hand, problem setting considered by Lazaro-Gredilla et. al (2019) [**112**] is very relevant to ours. The concepts are represented as programs with complex architecture, which can be decomposed into visual perception system, working memory, and action controller. This is very similar our sub-system decomposition, however, our method of representing concepts is a part of System 2 architecture, and our sub-system decomposition allows each sub-system to function independently of the other. Instead of demonstrations or self-exploration to learn about the concepts, the concepts are extracted from image pairs. The applicability of this framework is demonstrated in both simulated and real-world manipulation environments.

On the other hand, there are works that explore the applicability of causal inference in the context of reinforcement learning. Zhu et. al (2019) [**203**] proposes discovering the causal

dependency graph with the best scoring using reinforcement learning methods. They produce notable results in non-linear datasets with quadratic functions and sociology datasets, although they do not work with any reinforcement learning environments. There is a rich literature as a result of a systematic study of causal inference in the multi-armed bandit setting [**21, 14, 111, 196, 54, 164, 113**]. The need for causal inference arises the moment the assumption of independence between multiple arms is removed. [**113**] introduced structural causal bandit problem, SCM-MAB, where an SCM [**147**] is used to capture the relationship between the underlying causal mechanisms. [**114**] extends it to the case where not all variables are manipulable.

Roughly, our learning process can be described as a learning the $do(x)$ (*do-operator* over the variable x) through a combination of online learning from observations without any interference [**54**] and learning through the process of interference [**196**]. One of the assumptions that we make that renders the problem significantly easier is the absence of confounder variables in the environment. We assume all the variables required for the process of causal inference are visible. Overall, these methods provide a systematic and theoretically rigorous way to reason about different causal mechanisms in the environment. These methods, however, are yet to be extended to more complex problem settings.

2.4. System 3: Sub-task graph generation and planning

System 3 is responsible for the overall planning aspect of the system. In the context of learning to replicate demonstrations, System 3 can be used to infer the objective of the given demonstration in terms of a generalisable reward vector. It works by constructing a sub-task dependency graph that is specific to the current environment situation, and uses that to plan across different levels of abstraction. The effectiveness of this component is primarily dependent on the quality of the concept base learnt using System 1 & System 2.

With the absence of a concept base, inferring the demonstration objective and replicating it in different environments can also be done using classical imitation learning methods like behaviour cloning [**126, 136**], Dagger [**151, 153**]. Inverse reinforcement learning methods [**9**] tend to be more generalisable with respect to some aspects, although highly data inefficient.

These methods of replicating demonstration aren't always robust to changes in environment configuration and sub-task dependencies. It is not trivial to reuse the information learned in different context.

In terms of planning, if we assume access to the environment dynamics and the availability of hierarchical abstraction, then we can employ classical planning methods like depth-first search [**64, 140**], A-star [**78**] or Dijkstra [**43**]. However, the cost of these methods grows exponentially with the number of interactive objects present in the environment. The heuristics that would be needed to use approximate versions of these methods would need to have some resemblance to the learnt concepts.

Classical RL based policy and value iteration methods [**176**] can also be used at the abstraction level to learn generalisable policies. However, this is highly cost ineffective as it is non-trivial for the basic versions of the algorithms to adapt to new reward functions and it takes a lot of data to learn a decently accurate policy. It is also highly improbable that the learnt policy would extract the core causal mechanisms from the environment, and therefore would fail in environments with different sub-task dependencies. However, methods described in section 2.3.3 that explicitly model objects and their interactions can be much more generalisable in these settings.

If concepts are available during planning, along with the exhaustive set of skills, we can create an exhaustive set of options, with the initiation and the termination condition defined using the set of concepts. This would enable all the option-based policy and value iteration methods to come into play[**178**], making the planning a lot more efficient. We can also convert the environment into the graph format proposed in [**170**], enabling the use of a pre-trained zero shot sub-task graph planner.

Morere et. al. (2019) [**131**] propose a systems approach to learning and planning in hierarchical with unknown dynamics. Their work is similar in spirit to ours and deals with environments that have sub-task dependencies. Their method of learning concepts, or gathering knowledge about the environment is implicit in their skills. As their agent matures with experience, the learned skills are decomposed into fundamental skills and also whenever beneficial, composed into complex skills. Benefits gained by using composed skills are

similar to the benefits gained by composing concepts. However we note that the benefit of composing skills is lost when the sub-task dependencies themselves are changed. This is not true for the case of concepts, as they are based on core causal mechanisms of the environment. For example, a complex skill of "obtaining gold" as a combination of "removing stone using an axe" and "picking up gold" is valid only when the object gold is blocked by stone. However the concept of "stone" getting removed from the environment with the use of an "axe" is always valid. Therefore, it is much more simpler and practical to plan in terms of concepts instead of skills. Other than the fact that this method is not applicable when there is a change in the inherent sub-task dependencies, this method is the closely related to ours. Both methods deal with the aspect of gradually growing the knowledge base from experience and having a framework based approach towards hierarchical domains.

Chapitre 3

Method Overview

Our framework has three components:

- System 1: Experience segmentation
- System 2: Lifelong concept learning
- System 3: Graph based planning

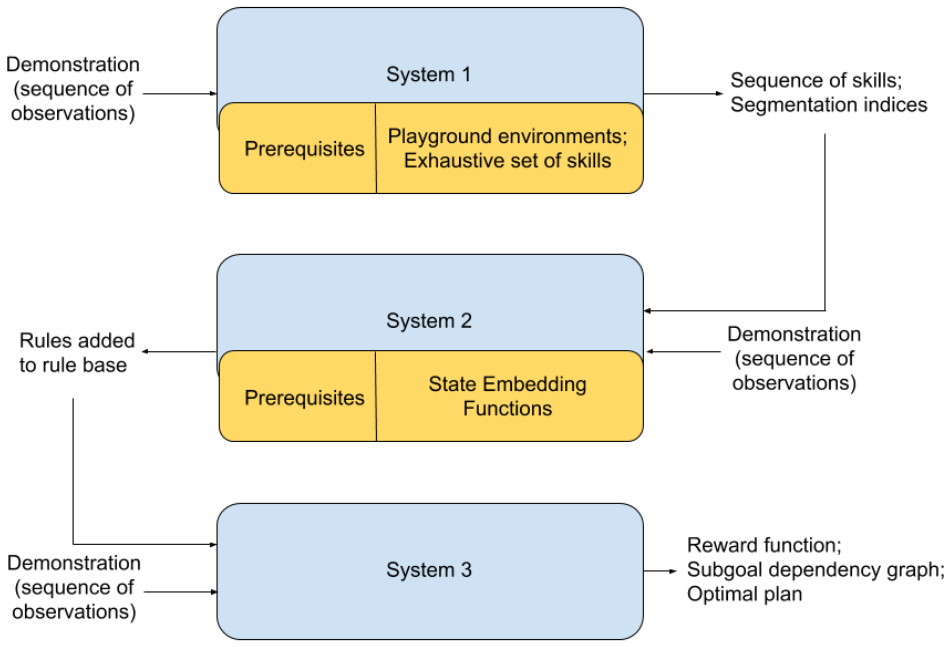


Fig. 3.1. Overall system diagram with demonstrations as the input stream of experience

This framework is designed to ensure continual learning and planning across different levels of abstractions in complex hierarchical environments. System 1 performs the preliminary step of preparing a given stream of experience such that relevant information can be easily extracted. In this work, by assuming access to an initial set of skills, a stream of experience is segment into a sequence of skills. System 2 analyses these segments and assimilates the extracted pieces of information into concepts. These concepts are repeatedly refined and generalised as the agent gets access to more information. Finally, System 3 uses the initial set of skills and the learnt concepts for the purpose of planning. It can infer objectives from a demonstration in terms of concepts, which is a robust way to replicate the demonstration across a variety of environment configurations.

3.1. Background and Definitions

The environment E has a state space S, action space A and the transition function P such that $P(s_{t+1}|s_t, a)$ denotes the probability that the environment reaches the next state s_{t+1} by taking the action a at state s_t, where $s_t, s_{t+1} \in S; a \in A$. The environment is partially observable, such that O is the observable part of S.

We also define the initial set of states, $I \subseteq S$ as the set of states the agent can find itself in on resetting the environment or at the beginning of an episode.

Definition 2. *The state value function represents a mapping from the state to the average expected result the agent can get from the state.*

Definition 3. *A policy is a mapping from state to action. This mapping can either be deterministic or stochastic*

$$\text{Deterministic} : \pi_i : S \rightarrow A \tag{3.1.1}$$

$$\text{Stochastic} : \pi_i : S \times A \rightarrow [0,1] \tag{3.1.2}$$

A Skill is a special case of a policy. Skills are parameterised and also have an additional action, END. The parameterisation makes the skill re-usable in different contexts. For

example, "cutting carrots" and "cutting onions" can be defined using the a single skill "cutting" that can incorporate parameters "carrots" and "onion" to fine-tune its behaviour in the environment. The END action ensures that once the required objective is reached, the skill stops executing and returns control to the framework. This aspect of returning control is important in planning across different levels of abstractions.

Definition 4. *A skill is a sequence of actions, specific to the current state and given parameters, that brings a distinct change in the environment, if such a change is possible.*

$$\text{Deterministic} : \pi_i : \Omega \times S \rightarrow A \cup \{END\} \tag{3.1.3}$$

$$\text{Stochastic} : \pi_i : \Omega \times S \times (A \cup \{END\}) \rightarrow [0, 1] \tag{3.1.4}$$

where $\pi_i \in \Pi$ and $\Omega \in \mathbb{R}^{p_i}$ is a tuple of parameters, specific to π_i, and $p_i \in \mathbb{W}$ is the number of parameters. In our work, once a set of parameters is chosen, they are typically kept constant until the "END" action is chosen and the skill execution is deterministic in nature. Therefore, instead of sampling actions one at a time, we can repeatedly sample actions till the "END" action is chosen. So, specific to this work, the skill transition can be described as:

$$\pi_i : \Omega \times S \rightarrow S \times A^K \tag{3.1.5}$$

where $K \in \mathbb{W}$ represents the number of low-level actions that the skill executed till the "END" action was taken.

Definition 5. *A set of skills is termed as exhaustive if any sequence of observations can be described as a sequence of the skills contained in the set.*

The most trivial set of exhaustive skills in an environment with discrete actions is the action space itself. The total number of exhaustive skills sets are infinite, because one can keep composing existing skills in the set to form new skills. This is similar to the options framework [**178**], where the potential set of options are infinite, and the utility is optimal only at certain number, beyond which, it actually hurts the performance.

Definition 6. *A minimally exhaustive set of skills contains the minimum number of skills that can explain any sequence of observations as a sequence of skills contained in the set*

One can note that in order for a set of skills to be minimally exhaustive, the skills must have an intricate relationship with the underlying causal mechanisms of the environment. Given an environment, humans can be remarkably accurate as to what the minimally exhaustive set of skills could be. Search for how to do this automatically has been an area of research for the past few decades. In our work, however, we assume that a minimally exhaustive set of skills is available at the start.

Definition 7. *Depending on the nature of the environment (if the environment is dynamic and stochastic, or if the nature of the tasks involve a consistent type of behaviour), there can be a skill that is always being executed unless something else is chosen. We call such a skill as the default or base skill.*

For example, in the case of CRAFT, the base skill is navigation. In Freeway, the base skill is dodging oncoming cars. In Seaquest, the base skill is staying afloat above water. In Montezuma's revenge, the base skill is staying still, not doing anything. This can be learned from scratch given sufficient interaction with the environment (see section 2.2.1).

Definition 8. *At any level of hierarchy, execution of skills leads the agent from one state to another. There are particular states that cause a change in either the structure of the environment, or the internal state of the agent. Such states are termed as events.*

Note that not all structural changes in the environment need not correspond to skill execution, some of the changes can simply be due to the dynamic nature of the environment. Using this definition of events, let us recall the definition of concepts (Def. 1.1) as the 2-tuple that consists of an abstract state transition and conditions required for such a transition to take place. In this work, all the state transitions that are consolidated into concepts are, in fact, events.

Definition 9. *Some of the environments available for use for the agent at all times are called practice environments. Here, the transitions are fully observable, and are typically used to refine and develop new concepts*

3.2. The CRAFT Environment

Craft is a 2D grid based environment, developed by Jacob Andreas at al. 2006, where every tile of the grid is populated by one of the following (11 objects or agent):

- Wood
- Iron
- Grass
- Workshop (w0)
- Tool shed (w1)
- Workbench (w2)
- Stone
- Water
- Gem
- Gold
- Boundary
- Agent
- Empty space

Fig 3.2: State space of CRAFT (visual)

The state space of the agent S, has an observable part O, and a non-observable part I which is the internal state of the agent, also called as the inventory. Both these parts are represented using a matrix that is populated with -1, 0, and 1. The observation O is given in the form of a grid, which has a dimension of $(W \times H \times 12)$, where 12 represents the different types of entities available in the environment state (List 3.2). The 12th layer is related to the information about the agent where 1 indicates the position of the agent, -1 represents the direction towards which the agent is facing. For example, if the agent is at the position (4,5) facing north, then (4,5,11) = 1, (4,6,11) = -1, and rest of the elements in (4,5,*) are 0. The inventory has a dimension of (21), where each dimension is used to represent the

quantity of a particular object in the inventory. The observable part can also be represented visually, as shown in Fig 3.2

There are 5 actions an agent can take, each represented by an integer. 0 stands for *down*, 1 stands for *up*, 2 stands for *left*, 3 stands for *right*, 4 stands for *USE*. The action *USE* is used to abstract environment interactions. For example, if you perform the action *USE* at workshop # 0 with the object “wood" in the inventory, you get a “plank". If you perform the same action in front of “grass", you remove the “grass" from the environment and add it to your inventory.

The tasks in this environment have an inherent hierarchical structure. Getting the gem, for example, requires you to cross the water. To do that you need to build a bridge by collecting iron and wood, and assembling it at the workshop. We describe all such tasks in the subsection 3.2.1.

3.2.1. Recipes

There are 10 possible tasks in the environment, each of which can be completed following a particular order of sub-tasks (assuming empty inventory):

- Make[plank] — [”get_wood”, ”use_workshop0”]
- Make[stick] — [”get_wood”, ”use_workshop1”]
- Make[cloth] — [”get_grass”, ”use_workshop2”]
- Make[rope] — [”get_grass”, ”use_workshop0”]
- Make[bridge] — [”get_iron”, ”get_wood”, ”use_workshop2”]
- Make[bed] — [”get_wood”, ”use_workshop0”, ”get_grass”, ”use_workshop1”]
- Make[axe] — [”get_wood”, ”use_workshop1”, ”get_iron”, ”use_workshop0”]
- Make[shears] — [”get_wood”, ”use_workshop1”, ”get_iron”, ”use_workshop1”]
- Get[gold] — [”get_iron”, ”get_wood”, ”use_workshop2”, ”get_gold”]
- Get[gem] — [”get_wood”, ”use_workshop1”, ”get_iron”, ”use_workshop0”, ”get_gem”]

3.2.2. Possible events

We have a set of events which can be categorized as follows:

- **Primary events** {get_wood, get_iron, get_grass} involve collecting the raw materials that are readily available in the environment. These events are usually successfully executed.
- **Workshop events** {use_workshop0, use_workshop1, use_workshop2}, depending on the state of the inventory during execution, give out different complex objects, which can be used to pass through barriers in the environment or make more complex objects. Success usually involves a change in the inventory, from simpler to more complicated objects
- **Complex events** {get_gem, get_gold} usually require having a particular complex object that is used to pass the barrier surrounding the final object to be successful. This is usually harder to successfully complete.

3.2.3. Skill Set

We have two parameterised skills in CRAFT, assuming only a single level of hierarchy:

(1) Go_to_(x,y): This uses dijkstra shortest path algorithm to move to the position (x,y) from the current position.
(2) Use_object_at_(x,y): This uses dijkstra shortest path algorithm to move to a position adjacent to (x,y) from the current position, orients itself towards (x,y), and performs the action USE.

Given the way CRAFT is constructed, if we use expert demonstrations, only the second skill is sufficient to explain the demonstration. And every single skill execution usually corresponds to an event. Recall that an event corresponds to either a change in the internal state of the agent, or a change in the structure of the environment. The first skill in this case is the default skill. This skill ensures the completeness of the set of skills for this level

of hierarchy, i.e., these two skills are sufficient to ensure that any demonstration at a state-action level can be uniquely converted to a sequence of sub-tasks and skills at this level of hierarchy.

Chapitre 4

System 1: Automatic skill segmentation

In this chapter, we consider the general case of segmenting a demonstration. We assume that the demonstration is comprised of observations at different time instants, i.e. we do not have access to the action sequence or to the full state. We assume access to several fully observable practice environments, and an exhaustive set of skills such that any demonstration can be decomposed as a sequence of these skills. Given this setup, our objective is to recover the sequence of skills being performed in the demonstration.

4.1. Formal Problem Definition

Input: Single demonstration τ,

where, $\tau = \{o_1, o_2, o_3, \dots o_N\}$, $o_i \in O$, $\forall i \in \{1,2, \dots N\}$ and all the observations are from consecutive states, i.e. if $s_i \in S$ is the state corresponding to observation o_i, then $\exists a \in A$ such that $\mathrm{P}(s_{i+1}|s_i, a) > 0$ $\forall i \in \{1,2, \dots N-1\}$.

Given:

(1) Practice environments 9), $\mathrm{D} : \{e_1, e_2, \dots e_L\}$, $e_i \in E$, $\forall i \in \{1,2, \dots L\}$
where the agent can act out random or specific policies and fully observe the transitions. Each e_i has a distinct initial condition and configuration.

(2) Exhaustive set of skills (Def. 5), $\Pi : \{\pi_1, \pi_2, \dots \pi_K\}$

Output: Sequence of skills (and respective parameters) executed in the demonstration

Output: $(\pi_{k_1}, \omega_{k_1}, \pi_{k_2}, \omega_{k_2}, \ldots \pi_{k_T}, \omega_{k_T}), \forall k_i \in \mathbb{Z}^+$ such that

If,

$$\pi_{k_i}(s_{\{k_i,1\}}, \omega_{k_i}) = (a_{\{k_i,1\}}, a_{\{k_i,2\}} \ldots a_{\{k_i,L_i\}}),$$
where $s_{\{k_i,(j+1)\}} = transition(s_{\{k_i,j\}}, a_{\{k_i,j\}})$; $s_{\{k_i,(L_i+1)\}} = s_{\{k_{i+1},1\}}$; and
$o_{\{k_i,j\}} = observation(s_{\{k_i,j\}})$

Then,

$$(o_{\{k_1,1\}}, o_{\{k_1,2\}} \ldots o_{\{k_1,L_1\}}, o_{\{k_2,1\}} \ldots o_{\{k_T,L_T\}}) = (o_1, o_2 \ldots o_N)$$

Here, note that $\pi_{ki} \in \Pi$ is a skill (Def. 4), and $\omega_{ki} \in \Omega$ represents the parameters for that skill. In simple terms, if we execute the output sequence of skills in the initial state of the demonstration, we would recover the sequence of observations that make up the demonstration.

Assumptions:

- Demonstrations can be decomposed into a sequence of skills, and this set of skills can be learned by interacting in the practice environment.
- The agent is the only dynamic part of the environment, although this method can be extended to dynamic environments as well.
- The practice, demonstration and test environments come from the same distribution of environments.

Key contributions: We introduce a method to segment demonstrations, which can be comprised of only partially observable state sequences, and may not have an associated action sequence. Since we learn to use the existing priors well, we can apply this method even when a single demonstration is available. The ability to convert a monolithic demonstration into a sequence of skills makes the problem of credit assignment much simpler. As a by-product, inferring the intention of a demonstration, and replicating the demonstration in various instances becomes much simpler. This ability to convert a sequence of low level observations

into a sequence of skills at a higher abstraction level is crucial for long range planning in complex hierarchical environments.

4.2. Skill discriminators

For the purposes of segmenting demonstrations into sequence of skills, we build skill discriminators for each individual skill. We pass the observations to skill discriminators, and each skill discriminator individually outputs either 0 (no), 0.5 (possible), 1 (yes), which we can use to identify which skill is being executed.

The skill discriminator of the base skill, if exists, mostly outputs the value 1, unless it sees a sub-optimal behaviour, when it outputs 0. The sub-optimal behaviour usually corresponds to some other skill being executed. For example, the base skill in Seaquest [**18**] is to stay afloat and do nothing. From the perspective of the base skill discriminator, if the agent is doing nothing, it is displaying the optimal base skill behaviour. If the agent decides to go underwater to "shoot sharks" or "rescue divers", although it is performing a useful sub-task in terms of the overall episode, it is a sub-optimal behaviour when looking through the lens of the base skill discriminator. Similarly, in our example, CRAFT, the base skill is "navigation". As long as the agent is optimally moving from point A to point B, the base skill discriminator outputs the value 1. The moment it takes a detour or starts interacting with an object, the base skill discriminator outputs 0.

4.2.1. A note on defining skill discriminators

The environment we consider in this work is deterministic in nature. Since we assume access to an initial set of skills, defining skill discriminators becomes relatively easy. For a given segment of experience, we simply check if the agent can reach the final state from the initial state by executing a particular skill. If the skill execution is equally efficient as the segment of experience and they achieves the same environment transition, then we can confidently predict that the skill was executed.

In the case of a stochastic environment, we can approach this in a more principled way by learning configuration-invariant skill embeddings for every skill. This can be done by sampling skills in random order in different practice environments (Def. 9) and utilising the sampled segments to form a supervised learning dataset[1]. This dataset can be used to train an RNN that can predict both the skill and the value of skill discriminator given a segment of experience. In the case of an RNN the skill embeddings are learned implicitly, however, there are architectures in recent work where skill embeddings are learnt explicitly [**37**].

4.3. Example: System 1 in CRAFT

Skill Discriminators: As mentioned in the previous section, since CRAFT involves deterministic transitions, it is relatively straightforward to define skill discriminators when given access to the skills. In CRAFT, we use two skills:

(1) Go_to skill: This skill is defined using the Dijkstra algorithm which the goal state as input and outputs the optimal trajectory that takes the agent to the goal state from the current state

(2) Use_object skill: Given the position of an object as the input, this skill computes the optimal set of actions that takes the agent from the current state to a state that is adjacent to the object. The skill outputs this set of actions along with the action "USE" appended at the end. The optimal set of actions is calculated using the Dijkstra algorithm, similar to the "go_to" skill

The skill discriminators are then defined using the skill definitions as follows:

(1) Go_to_discriminator: Given a sequence of observation, this discriminator uses the Go_to skill to compute the optimal trajectory from the initial state to the current state. If the given sequence of states is longer than the computed optimal path, then the discriminator outputs 0, otherwise it outputs 1. Note that go_to is a base skill, and therefore go_to_discriminator mostly outputs the value 1.

(2) Use_object_discriminator: The output of this discriminator depends on the last action executed and length of the path from the initial position to the final position. If

[1]Since we know which skill was executed and if it was successful, we automatically create the labels required.

the path followed is optimal and the last action executed is USE, then the discriminator outputs 1. If the path followed is optimal, and the last action executed is not USE, then the discriminator outputs 0.5. This is because we cannot be sure if the segment represents the use_object skill. In all other cases, the discriminator outputs 0.

Predictor Function: This is responsible for assimilating information from all the skill discriminators and making the final decision of where to segment a given stream of experience.

To understand how the function works, let us understand the behaviour of different skill discriminators. Recall that a skill discriminator can output the values $\{0, 0.5, 1\}$ on receiving a segment of experience. Here, '1' indicates that the given segment is a consequence of executing the given skill, '0' indicates that it is not, and '0.5' indicates that it could be and that it is waiting for future observations to make a final decision.

The base skill discriminators output the value '1' for most of the observation, whereas other skill discriminators output '0', or '0.5' depending on the way the observation sequence is proceeding. As we discussed in section 1.1, an event occurs after the successful execution of a skill. Therefore, at every time-step when an event has occurred, the corresponding skill discriminator outputs '1', whereas every other discriminator (including the base skill discriminator) outputs '0'.

Therefore, at any point of time, there is at most one skill discriminator that outputs the value '1', however there may be several skill discriminators that haven't made a decision '0.5'. The prediction functions waits until the end of observation sequence or until the point when all the skill discriminator outputs '0'. From this point, it searches backward for the first time instance where a skill discriminator has output the value '1'. This is the place where the stream of experience is segmented and remaining segmented is passed again to the predictor function. This process happens recursively until the stream of experience is represented in terms of a sequence of skills. Since the predictor function considers one observation at a time, it can function in online settings without any changes in the design.

4.3.1. Assumptions

Here we summarize some of the key assumptions inherent in our system:

(1) The demonstrations only consist of actions that either make some observable change in the demonstration, or is the action "USE": This means that the demonstration does not have situations where the agent is trying to walk into the obstacles or objects. This assumption ensures that we have a single correct prediction for the sequence of skills. This can be removed easily, without affecting the generality of the solution

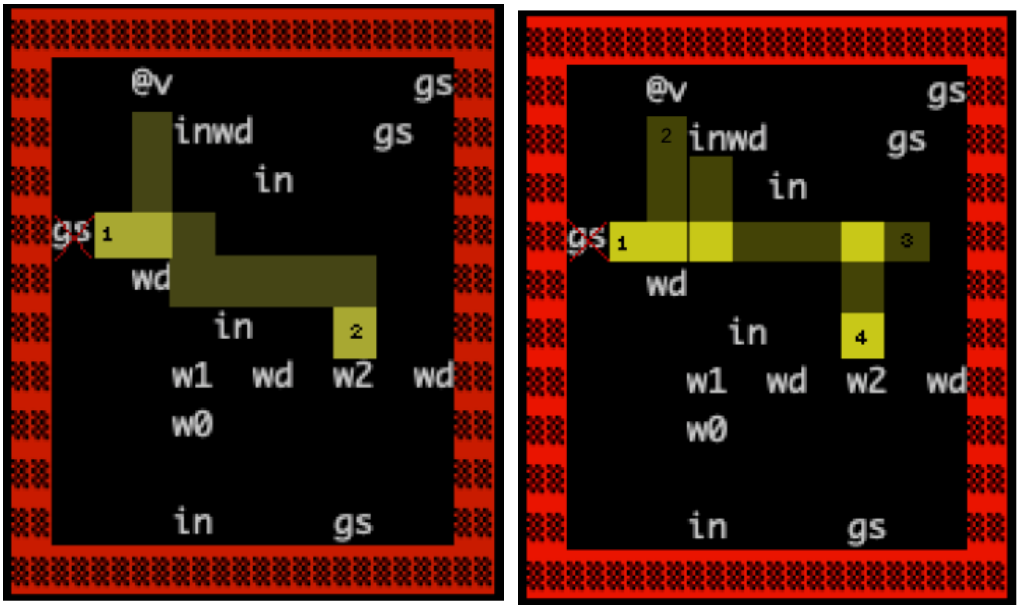
(2) We have access to fully observable and customizable reference environments, i.e. we can spawn multiple environments with a specific inventory and a specific set of objects initialized at random positions. This reduces the time and effort taken to reproduce the fully observable version of the events without affecting the generality of the baseline.

4.3.2. Example of segmentation, using predict function

To show the behaviour of System 1, we consider the task of making "cloth". To make a cloth in the CRAFT environment, we first obtain "grass", and then USE workshop no.2 (Figure 4.1). As we see in Figure 4.2-4.3, we segment the demonstration sequence, when we are sure about the skill that the segment belongs to. We can also see that the default skill (navigation), usually predicts 1 (green), as long as the agent is navigating in the environment optimally. The moment it behaves sub-optimally (Figure 4.3), our framework assumes that the behaviour was intentional and classifies it as a skill.

4.4. Summary

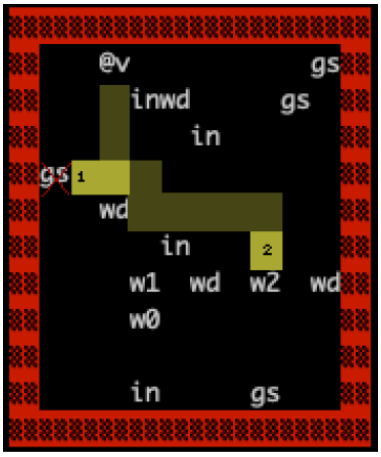
In this chapter, we describe the functioning of System 1 and the design choices we make that are specific to the CRAFT environment. We show how the System 1 uses the predictor function, along with the skill discriminators to decide where to segment a given stream of experience. The effectiveness of the system depends on the exhaustiveness of the skills it



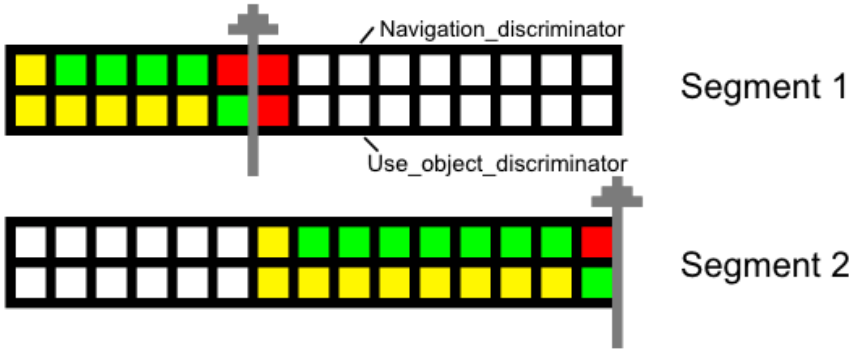
(a) Optimal demo (b) Sub-optimal demo

Fig. 4.1. Behaviour of System 1 with optimal (*Left*) and sub-optimal (*Right*) demonstrations of making stick. "@v" represents the agent's starting position. In the optimal demo, the agent first moves to the object grass denoted using "gs", and then uses workshop #2 denotes using "w2". In the sub-optimal demo, the agent makes a few detours to positions '2', and '3'

possesses. This reliance, however, can be relaxed by learning generalisable skill discriminators as described in section4.2.1 or like the one in Kipf et. al (2018)[**98**]. However, the principle behind System 1 remains the same. In Chapter 5, we discuss how these segments can be analysed to extract concepts.

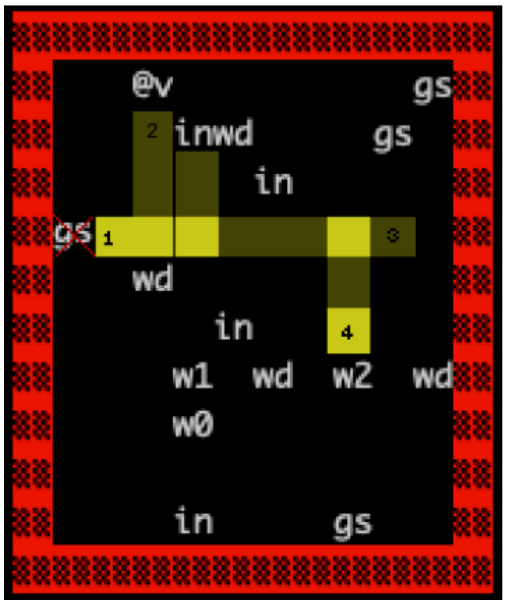


(a) Optimal demo

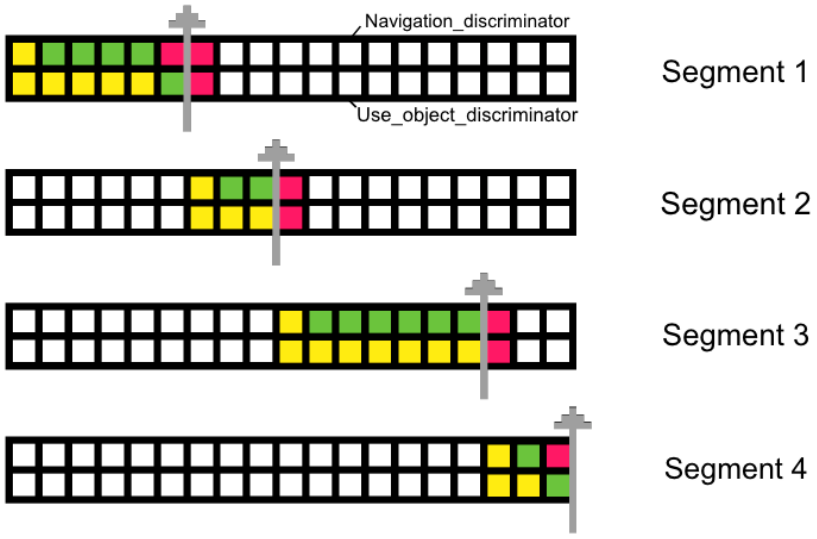


(b) Predictor function

Fig. 4.2. Skill segmentation prediction for the optimal demo. Yellow represents the value '0.5', which indicates that the skill discriminator's decision is pending. Green represents the value '1', which indicates that the skill has successfully finished executing. Red represents the value '0', which indicates that the given segment does not correspond to the particular skill. We can see that predictor function waits till the time instance when all the skill discriminators outputs '0'. Then, it segments at the latest point of time when at least one skill discriminator has an output of '1' (gray arrow). Then the sequence after the segmentation is again passed through the skill discriminators.



(a) Sub-optimal demo



(b) Predictor function

Fig. 4.3. Skill segmentation prediction for the sub-optimal demo. System 1 successfully decomposes the sequence of observations into 4 segments. As we describe in Figure 4.2, the system declares a segmentation (gray arrow) at the position where one only skill discriminator confidently predicts the end of skill execution.

Chapitre 5

System 2: Lifelong Concept Learning

The System 2 works in conjunction with System 1 to identify the characteristic concepts of the environment. This is a lifelong learning system that keeps on improving with experience, both from input demonstrations and self-exploration in practice environments. System 2 consists of a concept database that tracks all of the concepts about the characteristic events that the agent has experienced so far. It also has access to practice environment and skills due to System 1, with which it can reproduce the partially observed events in the fully observable practice environments whenever necessary.

System 1 can be used to segment demonstrations and identify concepts on the fly. If the agent encounters a new event, i.e. it does not find an appropriate concept in the System 2 database to explain the partially observed sequence of states of the event, then the agent initiates System 2 to reproduce the event in fully observable practice environments to identify the transition concepts and pre-conditions.

There is an element of planning required to reproduce the partially observed events in fully observable environments. Most often, if the environment conditions allow it, the event is reproduced by following some or all of the previous skills executed up until that point. Specifically for the sections of the state space that are ill-defined, we experiment with various pre-conditions, until we have sufficient information to disentangle out the most elementary set of concepts for that event.

Using these concepts, the agent can adapt to partial demonstrations and different initial conditions. When the agent cannot complete the demonstration, we understand the reason for not doing so easily by analysing the conditions that were not met. More importantly, in conjunction with System 3, this becomes a powerful way to plan ahead in complex hierarchical environments, and can attempt to fulfill the final objective in a different way.

One can do this in a more principled way by associating this with the way humans learn based on (Un)conscious (In)competence [**22**]:

(1) Unconscious Incompetence: An agent is usually unaware of the concepts or transitions that it hasn't encountered before. The moment it encounters something it cannot explain with the existing set of concepts, it moves on to the next state.
(2) Conscious Incompetence: This is the phase when we initiate system 2 to reproduce the event with different full state configurations, collecting data of various instances when the partially observable part of the event is reproduced, along with the corresponding full state configurations.
(3) Conscious Competence: This phase corresponds to the process where we disentangle out the core concepts of the environments from the data collected in the previous phase. Once we form competence about the transitions, we add the formulated concepts to the concept database and move onto the next phase. Prior to forming the concepts, we use a lot of memory to store the collected samples and we have a sub-optimal explanation of the transition that we have witnessed. This excess use of memory corresponds to the word "conscious", i.e. the transition that we are trying to explain hasn't yet been boiled down to it's core essence.
(4) Unconscious Competence: Once we have discarded the excess information, we have distilled down the collected transitions to the simplest possible concepts. We free up the memory of the agent, while being competent in being able to explain the observed event, and use this information in planning.

5.1. Example: System 2 in CRAFT

The motivation of the agent is to discover and store the most basic set of concepts, with which any event in the environment can be described. Events are a subset of sub-tasks, which happen after the end of skill execution. In CRAFT, due to assumption 2 in section 4.3.1, every skill execution corresponds to a possible event. Depending on the current inventory and the object being interacted with, different changes may happen in the inventory (the unobservable part of the state space) and in the structure of the environment. Our goal with the concept database of System 2 is to predict the unobservable changes in the state of the environment.

5.1.1. State Embedding Functions

State Embedding Functions are built to extract crucial information from the segments that we get from System 1. Only this information from the State Embedding Functions is used in System 2 to build the concept base.

The State Embedding Functions, in our case study, take in the two observations for an event – the observation just before the end of segment and the one just after, and the output from State Embedding Functions is recorded as the event. We use the following State Embedding Functions:

- Trigger: Outputs the index of the skill that caused the event[1].
- Object_in_front_before: Outputs the index of the object in front of the agent in the observation pre-skill execution.
- Object_in_front_after: Outputs the index of the object in front of the agent in the observation post-skill execution.
- Event_location: Store the coordinate at which the agent performed the event.
- New_Reachable_Objects: Outputs the new objects that became reachable via go_to_(x,y) skill. This usually happens if objects were blocked by other consumable objects that later got consumed. This function is very useful for System 3.

[1] In our case, since all the events are caused after the completion of "use_object" skill. The index is always equal to 1

Format of the stored concepts

The concepts of the agent comprise five characteristics:

(1) Object of interaction: Represented using the object index.
(2) Skill: Represented using skill index. In our situation, it is always 1.
(3) Transition: Represented using an integer vector of size 22. 21 elements represent the transition of the internal inventory of the agent, and the final element indicates if the object of interaction was consumed or not (indicated via -1 or 0).
(4) Pre-condition: Represented using a integer vector of size 21, describes the minimal conditions that need to be satisfied in the inventory of the agent for the event execution to take place successfully.

5.1.2. Reproducing Events

In CRAFT, the inventory is the only unobservable part of the demonstration. Therefore, in order to get the full picture of the event, we reproduce the same observations with various initial inventories. Specifically, we spawn an environment with the detected object of interaction, and different initial inventory and execute use_object_at_(x,y) skill. We experiment with the following initial inventory configurations:

- One object only. We consider cases where all the objects are present once.
- Two non-identical objects only. We consider all the possible pairs.
- Randomly initialised inventory. All the objects have an equal probability of being 0 to 3 in number. We consider 100 such initializations.

The first two types of initializations ensure that we encounter all the basic concepts that can exist. With this initialisation and the object of interaction, we execute the skill and record the transition (in terms of the structure of the environment and the inventory) along with the initial condition. Let us call this dataset D.

5.1.3. Learning to Disentangle concepts

Most of the concepts in CRAFT are easy to infer. A minor complication arises with stone and water, but the workshop interactions are the only major source of complication. We use the following method to disentangle all the possible concepts:

(1) Unique transitions: From set of recorded transitions D, we gather all the unique transitions and rank them in ascending order such that the simplest transitions occur first [2].

(2) Linearly independent transitions: We collect transitions, starting from the simplest ones, such that the new transition is linearly independent to all the collected ones. Let us call this collection of linearly independent transitions as S. These steps ensure that we collect the simplest possible set of concepts that can explain all the phenomena[3].

(3) Pre-condition concept: For every transition T in the set of linearly independent transitions S, we check the entire repository of recorded transitions D and collect the various initial conditions where T occurred. Of all these initial conditions, we select the simplest one as the pre-condition P[4], for the transition T.

5.1.4. Assumptions

(1) We have access to fully observable and customizable reference environments, i.e. we can spawn multiple environments with a specific inventory and a specific set of objects initialized at random positions. This reduces the time and effort taken to reproduce the fully observable version of the events without affecting the generality of the baseline.

(2) The concepts can be disentangled via simple linear decomposition of all the observed transitions.

(3) Every event has only one possible initial condition. There is no "OR" in the pre-conditions.

[2] Any complex object being made or used up is weighted more than simpler ones

[3] There may be cases where we don't end up learning the simplest set of concepts, there may provably be a better method to do so

[4] We take the least common number of times each object in the inventory occurs in the initial condition vectors as the number of that object in P

(4) Every complex object can be made using only one method.

5.1.5. Learning concepts via exploration

In this section, we experiment with agent executing skills at random and discovering concepts by itself. Each environment is constructed with 3 primitives of each kind, one instance of every other object placed randomly in a 15x15 grid. We experiment with 3 such constructed environments, as shown in Figure 5.1-5.3.

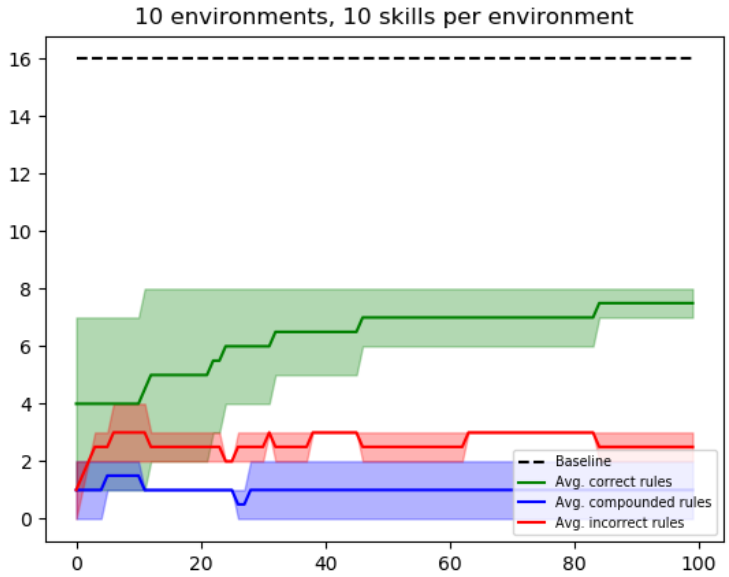


Fig. 5.1. Number of correct, compounded and incorrect concepts learnt when utilising 10 practice environments, using 10 randomly ordered skills in each. (Shaded colored region represents average deviation of the result). Note that the total number of independent core concepts that can be gathered from the environment is 16.

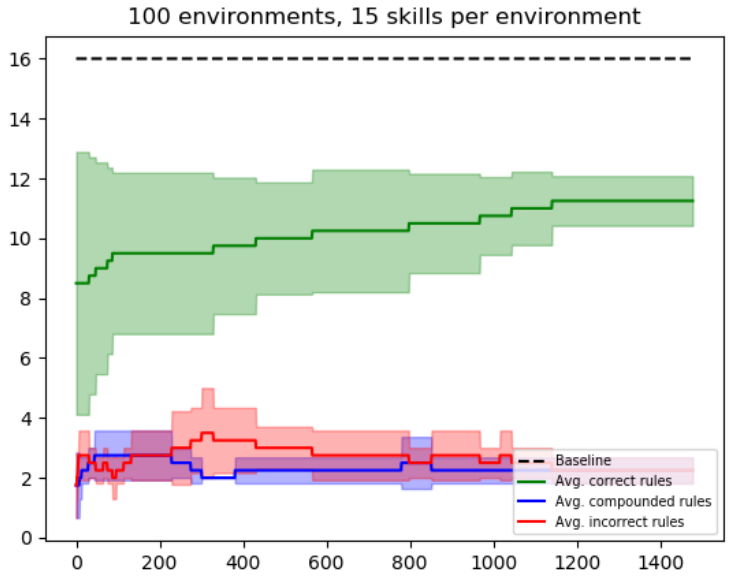


Fig. 5.2. Number of correct, compounded and incorrect concepts learnt when utilising 100 practice environments, using 15 randomly ordered skills in each.

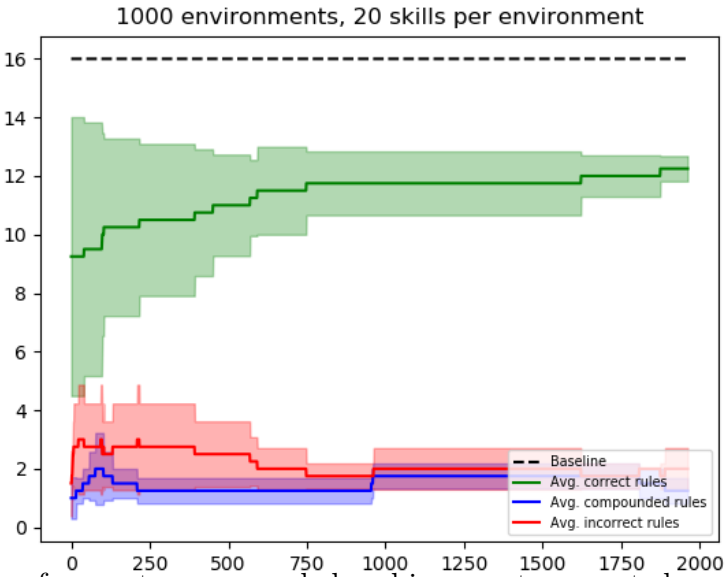


Fig. 5.3. Number of correct, compounded and incorrect concepts learnt when utilising 100 practice environments, using 15 randomly ordered skills in each.

Most of the incorrect concepts we find are a result of not being able to identify the conditions necessary. Because of limited interactions witnessed in the environment, the agent rarely sees the minimal inventory version of the same transitions. Let us consider the example of making "stick" from "wood". Ideally, the concept requires only 1 piece of "wood" as the pre-requisite.

But since the system sees limited examples of the concept, it is possible that in all the examples of the transition, the agent has at least 1 piece of "wood" and 1 piece of "cloth" as its initial inventory. In such a case, the agent learns that to make "stick", we must have at least 1 piece of "wood" and 1 piece of "cloth". It is easy for even human beings to make such mistakes when one has no prior on how the individual entities are typically used.

5.1.6. Learning concepts using demonstration

Let us see an example of how the concept base grows with different demonstrations. Recall that concepts are defined as a 2-tuple consisting of an abstract state transition and conditions required for such a transition to take place (Def. 1.1). Here, the CRAFT environment's internal state of the agent (inventory) is a decent abstraction of the current environment situation. Although, this abstraction does not take the reachability of objects and environment structure into account, this is sufficient to represent the concepts necessary for our purpose. The state transition is represented as a 22-dimension vector where the first 21 elements represent the inventory changes that takes place, and the final element indicates if the object being interacted with was consumed or not (-1 indicates that it was consumed, 0 indicates that it was not). The pre-requisite or conditions vector is a 21-dimension vector that indicates the minimal objects that are needed in the inventory for the transition to take place successfully. Specifically, each index of the inventory represents a particular object in the environment (7:iron, 8:grass, 9:wood, 10:gold, 11:gem, 12:plank, 13: stick, 14:axe, 15:rope, 16:bed, 17:shears, 18:cloth, 19:bridge, 20:ladder). The first seven indices usually assume the value 0 as they represent environment entities that cannot be taken into the inventory. (0:empty_space, 1:boundary, 2:workshop0, 3:workshop1, 4:workshop2, 5:water, 6:stone) [5].

Let us now examine how the concepts are added to the inventory. In Table 5.1, we show that the concepts that are gathered by the agent as it interacts with different workshops with the object "grass" already present in the inventory. In workshop #0, it learns the concept of making a "rope". In workshop #2, it becomes familiar with the object "cloth". When

[5]Note that the we use the internal state of the agent provided by CRAFT as is, and the indices used to represent di□erent entities are consistent throughout the environment.

interacting with workshop #1, it has an initial inventory of "grass" and "plank", and it learns the concept of constructing a "bed". Finally, in the last diagram, we are shown a full snapshot of the concept base. Here, we notice that although it is not acclimatised with obtaining the object "plank", it knows how to use it in the context of making "bed". Therefore, if it had to make a "bed" with no "plank" in the inventory, it would not know how to do so. However, it can add the missing concepts to its repertoire whenever the opportunity is presented. The advantage of such a method of learning and representing concepts explicitly is that they can still function without having the complete information and it is very intuitive for users to interact with such systems. For example, in this case, the user can simply notice that the agent is missing a trivial concept, and manually add it or guide the agent to learn the required concept.

Concepts added while getting grass

```
Object: grass
Transition: [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
 0  0  0  0  0  0 -1]
Pre-requisite: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Concepts added while making cloth

```
Object: workshop2
Transition: [ 0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0
 0  0  0  1  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
```

Concepts added while making rope

```
Object: workshop0
Transition: [ 0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0
 1  0  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
```

Concepts added while making bed

```
Object: workshop1
Transition: [ 0  0  0  0  0  0  0  0 -1  0  0  0 -1  0  0
 0  1  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0]
```

Final snapshot

```
Object: grass
Transition: [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
 0  0  0  0  0  0 -1]
Pre-requisite: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]


Object: workshop2
Transition: [ 0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0
 0  0  0  1  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]


Object: workshop0
Transition: [ 0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0
 1  0  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]


Object: workshop1
Transition: [ 0  0  0  0  0  0  0  0 -1  0  0  0 -1  0  0
 0  1  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0]
```

Tab. 5.1. Different concepts the agent learns with respect to the object "grass"

In Table 5.2, we showcase the ability to break down a compounded concept into basic elemental concepts as the agent gains experience. As discussed before, it is possible for an agent to register redundant information about the environment into its conceptualisation of

a transition. This may simply be because the agent didn't see enough examples of the transitions to disentangle accidental correlations. Similarly, it may store two or more separate elemental concepts as a single concept. For example, in Table 5.2, when the agent interacts with "workshop #0" with both "grass" and "wood" in its inventory, it learns the concept of making "plank & rope" as a single concept. Only later when it is shown examples of making "plank" and "rope" separately, it eliminates the compound concept of making "plank & rope" from its repertoire and retains the concept of making them individually.

Using workshop0 with grass & wood

```
Object: workshop0
Transition: [ 0  0  0  0  0  0  0  0 -1 -1  0  0  1  0  0  1
 0  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0]
 Used grass, wood to make plank, rope at workshop0
```

Using workshop0 with wood

```
Object: workshop0
Transition: [ 0  0  0  0  0  0  0  0  0 -1  0  0  1  0  0  0
 0  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 Used wood to make plank at workshop0

Transition: [ 0  0  0  0  0  0  0  0 -1 -1  0  0  1  0  0  1
 0  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0]
 Used grass, wood to make plank, rope at workshop0
```

Using workshop0 with grass

```
Object: workshop0
Transition: [ 0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  1
 0  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 Used grass to make rope at workshop0

Transition: [ 0  0  0  0  0  0  0  0  0 -1  0  0  1  0  0  0
 0  0  0  0  0  0]
Pre-requisite: [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 Used wood to make plank at workshop0
```

Tab. 5.2. Example of a compounded concept broken down after witnessing more information

We notice that this way of learning concepts is similar to the way we humans disentangle factors of variation and understand the core causal mechanisms of the environment. This is a simple way for a lifelong learning agent to learn different concepts about the environment from limited data. These concepts can be refined as the agent gets access to more experience. The learnt concepts are intricately linked to segments supplied by System 1. If one has access to skills at multiple levels of hierarchy that are sufficiently exhaustive (Def. 5), then one can learn concepts at multiple levels of hierarchy. This would enable an agent to plan across different time scales and help the agent effectively navigate across complex environments.

5.2. Summary

In this chapter we show how the System 2 functions in the environment CRAFT. For the abstracted state representation, we used the agent's internal state that is pre-built in the CRAFT environment. In a general scenario, we'd have to learn this representation from scratch or use expert knowledge about the domain to design a representation. The purpose of the representation is to abstract away all the unnecessary details and ensure that the concepts learnt can be re-used in a wide variety of situations. Using this abstracted representation, we demonstrated how concepts can be learnt in a lifelong manner from different streams of

experience like partially observable demonstrations and from self-exploration where internal state of the agent can be seen. When the system is unable to disentangle factors of variation, the concepts learnt may be incorrect. The system may learn the wrong pre-requisites, or combine multiple concepts and store it as a single concept. Some of these mistakes, however, can be corrected with more data, as demonstrated in Table 5.2. This ability to refine and update concepts with ease is an important aspect of a lifelong learning agent. In Chapter 6, we explore how the accuracy and completeness of the learnt concepts determines the final performance of the agent.

Chapitre 6

System 3: Graph Based Planning

System 3 utilises the concepts learned by the agent using System 1 and System 2 for planning. These concepts make the agent robust to drastic changes in the environment configurations and vastly reduces the search space during operation. System 3 can be employed in both the imitation and goal-specific planning scenario. In imitation learning, System 3 utilises its understanding about the event dependencies to understand the demonstrator's intentions. This intention is converted into a reward function which is later used in planning.

6.1. Inferring objective

In a given demonstration, there is at least one core objective that the agent is trying to achieve. But if one were to observe a demonstration of an agent operating in a complex environment, it becomes hard to identify the core objective from other side-objectives that were also achieved. In the example of making a salad, if the demonstration also consisted of converting "vegetable waste" into "compost" which was later utilised elsewhere, it would be hard for an agent to comprehend if making "compost" was the core objective or making "salad". Similarly, in the domain of CRAFT, if the core objective is to get "gem" but the "gem" is surrounded by "stone", it becomes necessary for the agent to make an "axe" for the complete sequence of events). In such demonstrations, it becomes ill-posed for an agent to identify if core objective included making "axe". In this work, we follow certain guidelines to decide which events should be a part of the core objective. These objectives must be achieved in the test environment for a replication to be considered successful.

Let us examine the example of obtaining "gem" that is surrounded by "stone". As mentioned previously, this involves making an "axe" and "removing stone". To make an "axe" there is a pre-requisite of having "stick" and "iron" already present in the inventory. These items are taken to "workshop #0", where the "axe" is made. Similarly, to make "stick", one must first obtain "wood" and build the "stick" at "workshop #1". As we can see, there are several dependency relationships that exist between the events (See figure 6.1b). Here, getting "gem" is the only event in the sequence that is not utilised elsewhere. Such events are termed as *independent events*. There are other events that either make other objects in the environment reachable or lead to building of objects that can be utilised later. Such events are termed as *reusable events*. In our work, we treat all the independent events as the core objective and reusable events as the bonus objective[1]. . It is not necessary for an agent to ensure that the reusable events are replicated in test environments. In this example, since "axe" can reused again for the purpose of removing other "stones", making an "axe" becomes a part of the bonus objective.

System uses this principle to identify independent and reusable events from a given sequence of events. Figure 6.1c shows the final output of System 3 for the above example. We outline the general algorithm in Algorithm 1. In this algorithm, we analyse events one at a time and keep track of what each event achieved in the environment. Specifically, we make note of all the objects in the environment that become reachable to the agent and the event that was previously executed to ensure that the current event becomes reachable. This information is stored using a "reachability dependency" table. Similarly, we have an "inventory dependency" table that contains the information related to events that created the objects being used in this event and the new objects that are being created in this event. We use these dependency tables to determine which events were executed for their own sake (independent events) and which events left certain objects in the inventory that can be reused later (reusable events).

[1]The numerical value of the bonus objective is a hyper-parameter

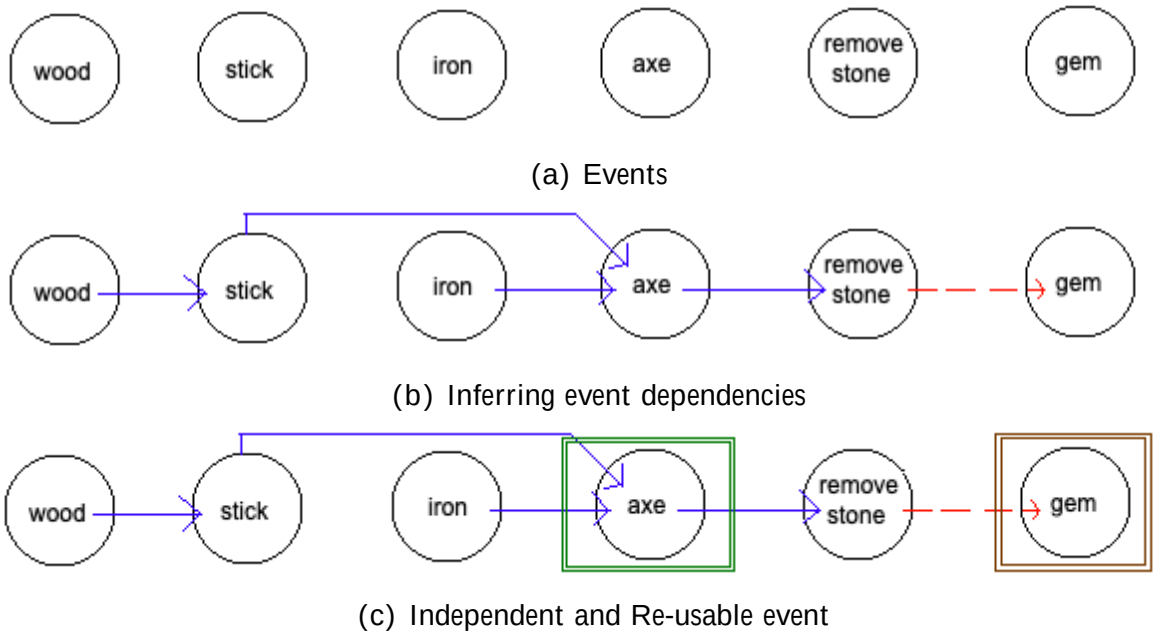


Fig. 6.1. Inferring objectives from demonstrations. Some events are inventory dependent (*blue line*), and some are reachability dependent (*dashed red line*).

Algorithm 1 Inferring the objectives given event sequence

```
1: procedure InferObjective(Event_Sequence, Initial_set_of_reachable_objects)
2:     Inventory_buffer ← [ ]
3:     Reachability_buffer ← [ ]
4:     Independent_events ← Event_Sequence
5:     Reusable_events ← [ ]
6:     Parent_Edges ← [ ]
7:     for all Event ∈ Event_Sequence do
8:         Event_pre-requisite ← Check_prerequisite(Event)
9:         New_reachable_positions ← Get_new_reachable_position(Event)
10:        New_objects_created ← Get_newly_created_objects(Event)
11:        Event_position ← Check_event_position(Event)
```

```
        if Event_position ∉ Initial_set_of_reachable_objects then
            Parent_event ← Get_parent(Event_position, Reachability_buffer)
            Parent_edges ← Add_parent_child(Parent_event, Event)
            Independent_events ← Independent_events − {Event}
        end if
        Inventory_buffer ← Add_to_inventory_buffer(New_objects_created, Event)
        Reachability_buffer ← Add_to_reachability_buffer(New_reachable_positions,
                                                         Event)
        if Event_pre-requisite ∈ Inventory_buffer then
            Parent_events, Reachability_buffer ← Update_reachability_buffer(
                                                    Event_pre-requisite,
                                                    Reachability_buffer)
            if Parent_events then
                Independent_events ← Independent_events − {Event}
                for Parent_event ∈ Parent_events do
                    Parent_edges ← Add_parent_child(Parent_event, Event)
                end for
            end if
        end if
    end for
    Reusable_events ← Get_events_from_buffer(Inventory_buffer)
    return: Independent_events, Reusable_events
end procedure
```

6.2. Planning using a Graph Guide

System 3 utilises the information it has gathered using prior systems to construct a graph guide:

Definition 10. *A graph guide is graphical representation of the environment state that is constructed using the concepts present in the concept base. It has a core structure that represents dependencies between events that do not change with environment structure and an auxiliary structure that needs to be regularly updated to reflect the current environment structure.*

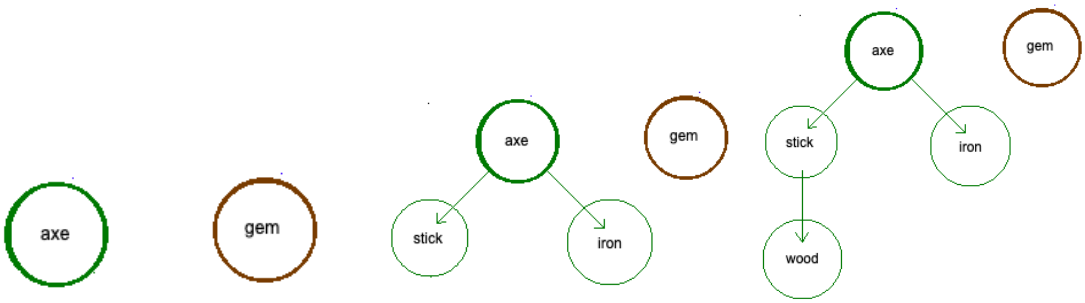
System 3 relies on this graph guide to provide dependency information about different events. This information is used to formulate plans that help the agent accomplish a given objective. The graph guide is similar to the subtask graph introduced in Sohn et. al (2018) [**170**]. In their work, the RL problem is completely restructured to represent dependency information

in terms of graph. The optimal solver proposed in their work uses only this graph to formulate solutions. Our work, up to this point, can be considered as a bridge between classical RL and the subtask graph RL. The key difference in the subtask graph introduced in Sohn et. al (2018) [**170**] and graph guide presented in our work is the presence of an auxiliary structure in the graph guide. This auxiliary structure captures environment details that could alter task dependencies. For example, in the case of CRAFT, the auxiliary graph contains the position information about different objects in the environment. Since using up certain objects make can other objects directly reachable by the agent, this can affect the overall optimal plan of the agent.

6.2.1. Construction of the graph guide in CRAFT

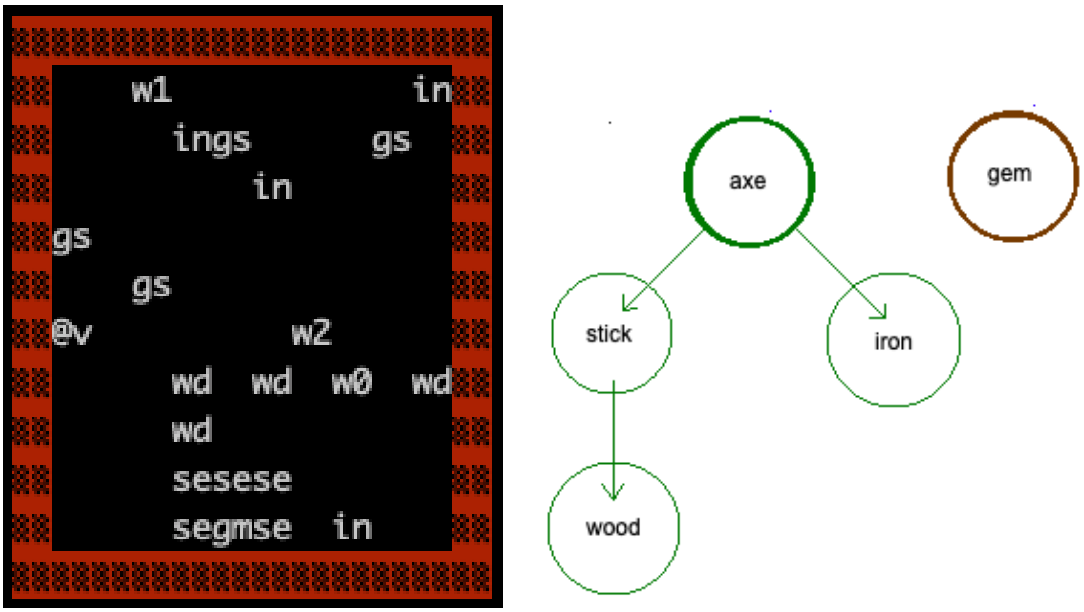
The purpose of a graph guide is to provide all the information necessary for the purpose of planning at different levels of abstraction. To execute an event successfully, the graph guide must be able to represent all the pre-requisites that need to be achieved and different ways of achieving them. This ensures that the agent can completely rely on the graph guide during the course of planning and the optimisation procedures reduces to choosing between different pathways to achieve the same final objective.

As we discussed before, there are two types of dependency relationships between events in the CRAFT environment – inventory dependency and reachability dependency. Inventory dependencies are trivial to identify using the learnt concepts and these dependencies do not change with environment structure. These dependencies are represented by the core structure of a graph guide. For example, irrespective of the environment configuration, one must have a "stick" and "iron" in the inventory, and utilise these objects in "workshop #0" to build an "axe" (See Fig. 6.2). The reachability dependency, however, depends on the current environment configuration. This dependency is used to represent objects that block a particular object from being readily reachable to the agent. For example, in Fig. 6.3, "gem" is reachability dependent on removing "stone", whereas in Fig. 6.4, it is reachability dependent on removing "grass".



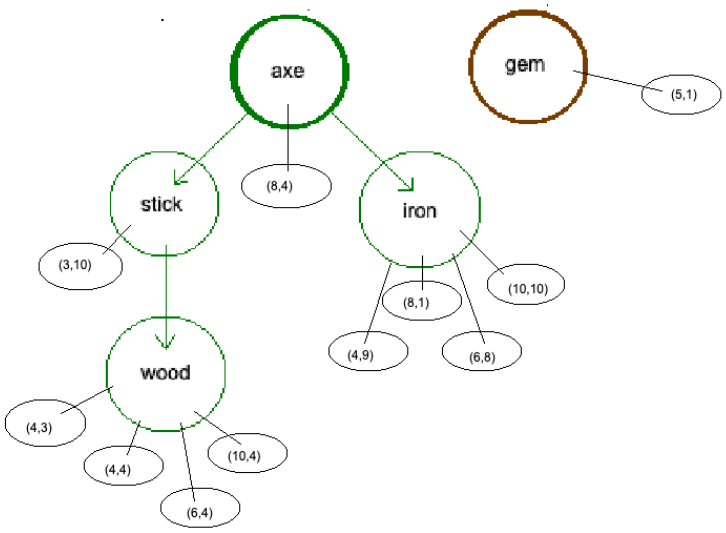
(a) Stage 0 (b) Stage 1 (c) Stage 2

Fig. 6.2. Constructing the core skeleton of the graph

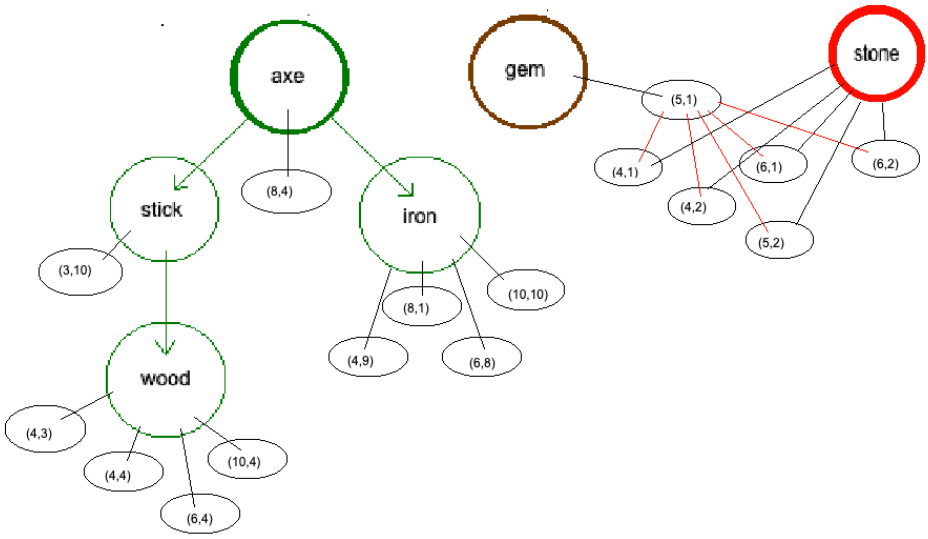


(a) Environment #1 (b) Core Skeleton

Fig. 6.3. Adapting the core skeleton to Environment #1 *(cont.)*

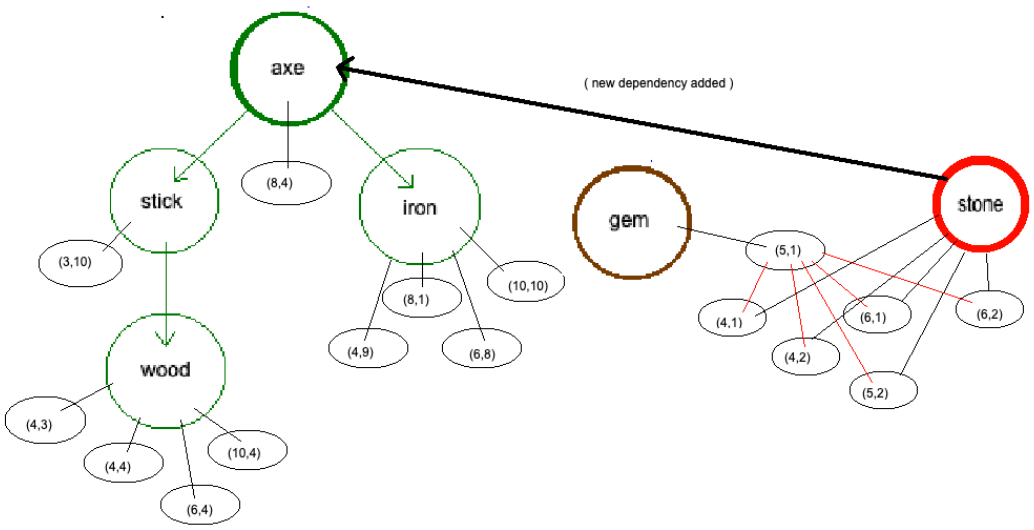


(c) Step 1: Node positions



(d) Step 2: Updating reachability conditions

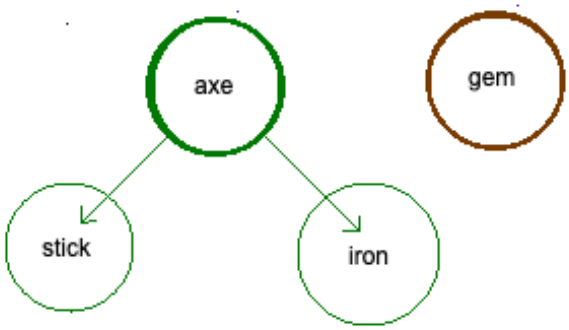
Fig. 6.3. Adapting the core skeleton to Environment #1 *(cont.)*



(e) Re-configuring core skeleton

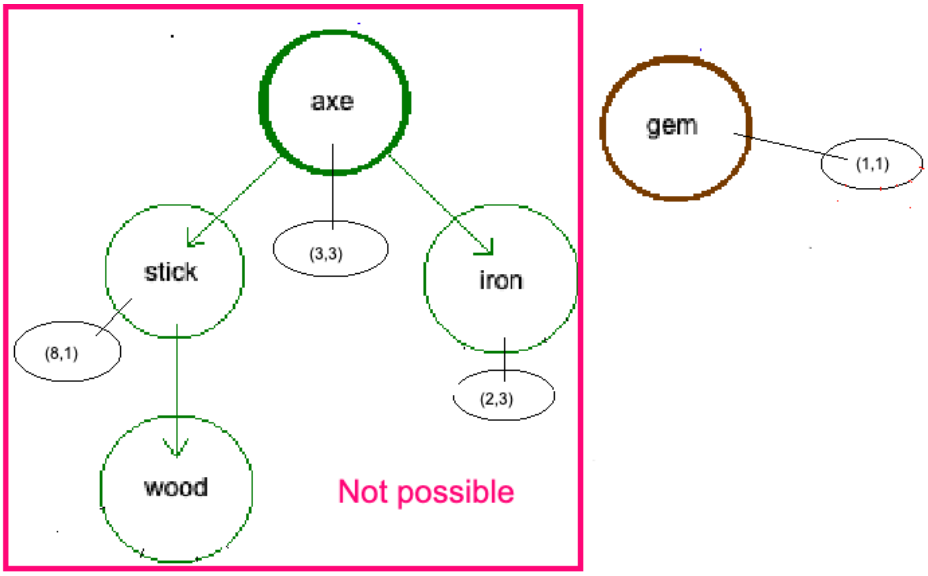
Fig. 6.3. Adapting the core skeleton to Environment #1

(a) Environment #2

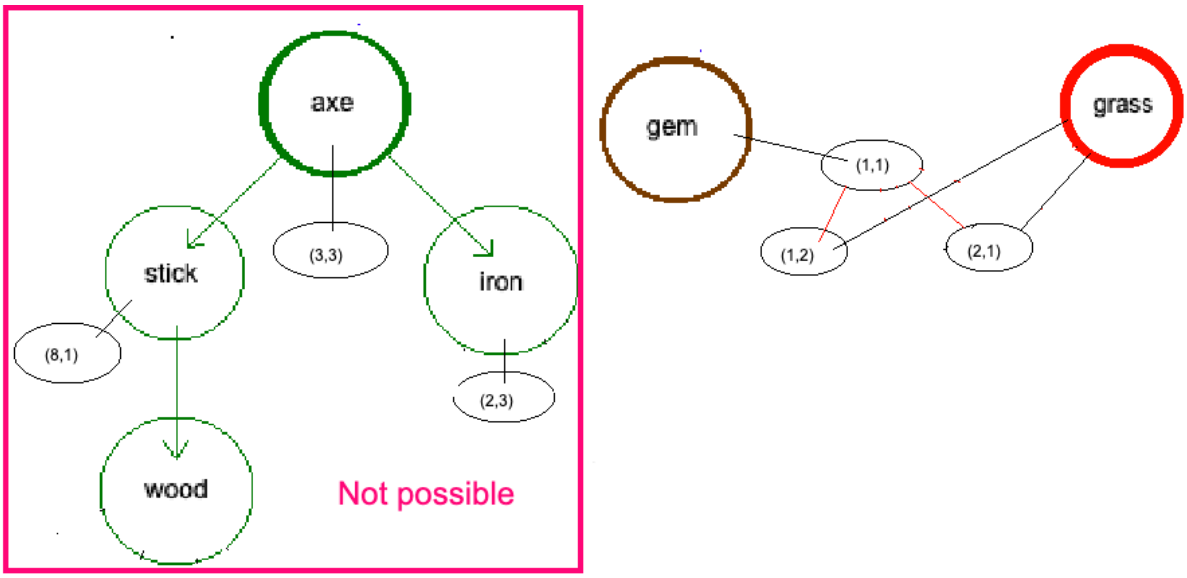


(b) Core Skeleton

Fig. 6.4. Adapting the core skeleton to Environment #2. *(cont.)*



(c) Step 1: Node positions (no "wood" available in the environment)



(d) Step 2: Updating reachability conditions

Fig. 6.4. Adapting the core skeleton to Environment #2.

The graph guide is constructed by iteratively updating the core skeleton graph and the auxiliary skeleton graph of the environment until all the dependencies have been represented in the graph guide. First, considering the agent's objectives, the core skeleton graph is constructed using the concepts acquired in the concept base (Figure 6.2). Then the we identify different positions at which each event in the core skeleton can take place. Then we check if all the positions are reachable by our agent. In the case that they are not, we

check the concept base to identify events that can make those positions reachable. These events are then added onto the core skeleton graph, and the entire process is repeated. The detailed version of this algorithm is outlined in Algorithm 2.

Let us consider the case when getting a "gem" is the core objective, and making an "axe" is the bonus objective. We observe the functioning of this algorithm in two environments with different reachability dependencies (Figure 6.3 & 6.4). In Figure 6.3, the core skeleton is re-configured such that making "axe" is now a pre-requisite for getting "gem", whereas in Figure 6.4, the agent has to remove "grass" in order to obtain "gem". Note, that the agent is able to realise beforehand that "axe" cannot be made with the available objects in the second environment (Figure 6.4c). As we can see, that such deductions can ease planning considerably. However, as we mentioned before, this vastly depends on the quality of the concepts that the agent has gathered. In case, the concept base is incomplete, we may end up with sub-optimal solutions or even miss solutions.

Algorithm 2 Constructing Graph Guide

```
1: procedure ConstructGraphGuide(Current_environment, Independent_events, Reusable_events)
2:     All_events ← Independent_events ∪ Reusable_events
3:     New_events ← Independent_events ∪ Reusable_events
4:     Core_Skeleton = []
5:     Node_locations = []
6:     Auxiliary_skeleton = []
7:     while len(New_events) > 0 do
8:         Event ← Pop_event_from_queue(New_events)
9:         Child_event ← Get_event_prerequisite(Event)
10:        Add_to_core_skeleton(Core_Skeleton, Event, Child_event)
11:        if Child_event ∉ All_events then
12:            New_events ← New_events ∪ Child_event
13:            All_events ← All_events ∪ Child_event
14:        end if
```

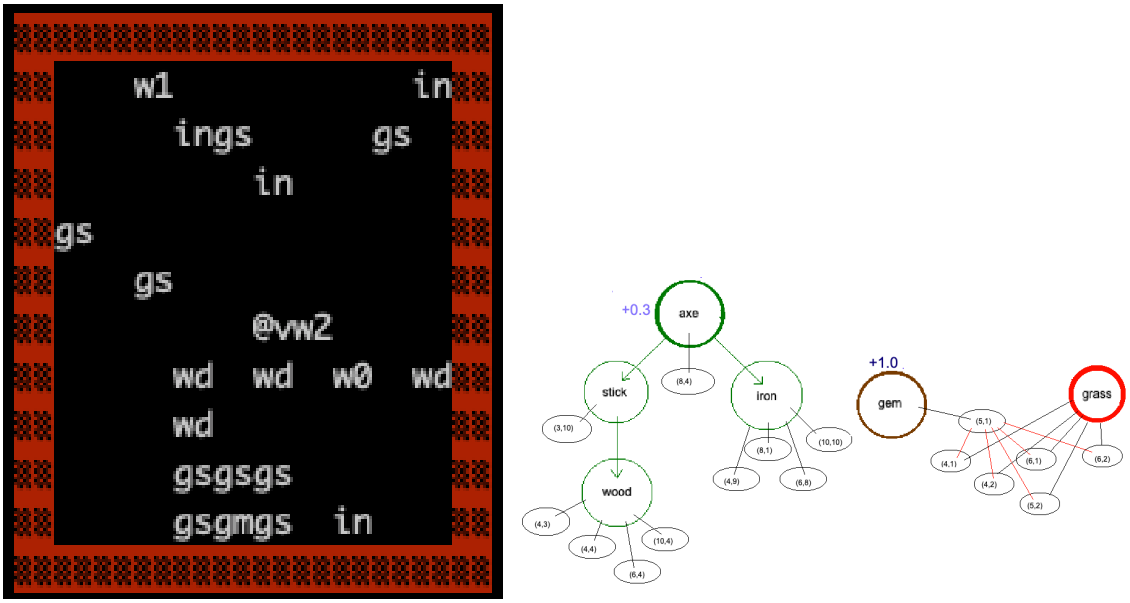
```
        Node_locations ← Get_location_in_env(Current_environment, Event)
        for node_location ∈ Node_locations do
            nodes_blocking ← get_blocking_nodes(node_location, Current_environment)
            for node in nodes_blocking do
                All_possible_events ← possible_events_to_unblock(node)
                Unexplored_events ← All_possible_events − All_events
                if len(Unexplored_events) > 0 then
                    Add_to_auxiliary_skeleton(node_location, node)
                    New_Events ← New_Events ∪ Unexplored_events
                    All_Events ← All_Events ∪ Unexplored_events
                end if
            end for
        end for
    end while
    return: Core_Skeleton, Node_locations, Auxiliary_skeleton
end procedure
```

6.2.2. Using the graph guide to plan

As we mentioned before, the events that constitute core objective are the ones that need to occur for the planning to be considered successful, whereas the events that constitute the bonus objective need not necessarily occur. When using the graph guide to plan, the agent has to decide between different events that can be executed. To help the agent weigh between different choices, we assign rewards to both types of events. This is a hyperparameter that would determine the sequence of events in the solution outputted by System 3. At any point of time, the agent considers the trade-off between the cost to reach a particular position and the reward it would get on completing an event, and uses this information to make a decision. This roughly corresponds to an agent considering different options in the "Options" framework [**178**] by considering the expected value of each option.

In our experiments, the events constituting the core objective have a reward of $+1$, and the bonus events have a reward of $+0.3$. These rewards are propagated to the lower-most leaves of in the graph guide. For example, if making an axe has a reward of $+1$, but the agent does not have the necessary pre-requisites to do this, then this reward of $+1$ is propagated to getting wood and getting iron. Then interacting with any positions that contain either an iron ore or a tree would provide the agent with a reward of $+1$. (Figure 6.5). At any

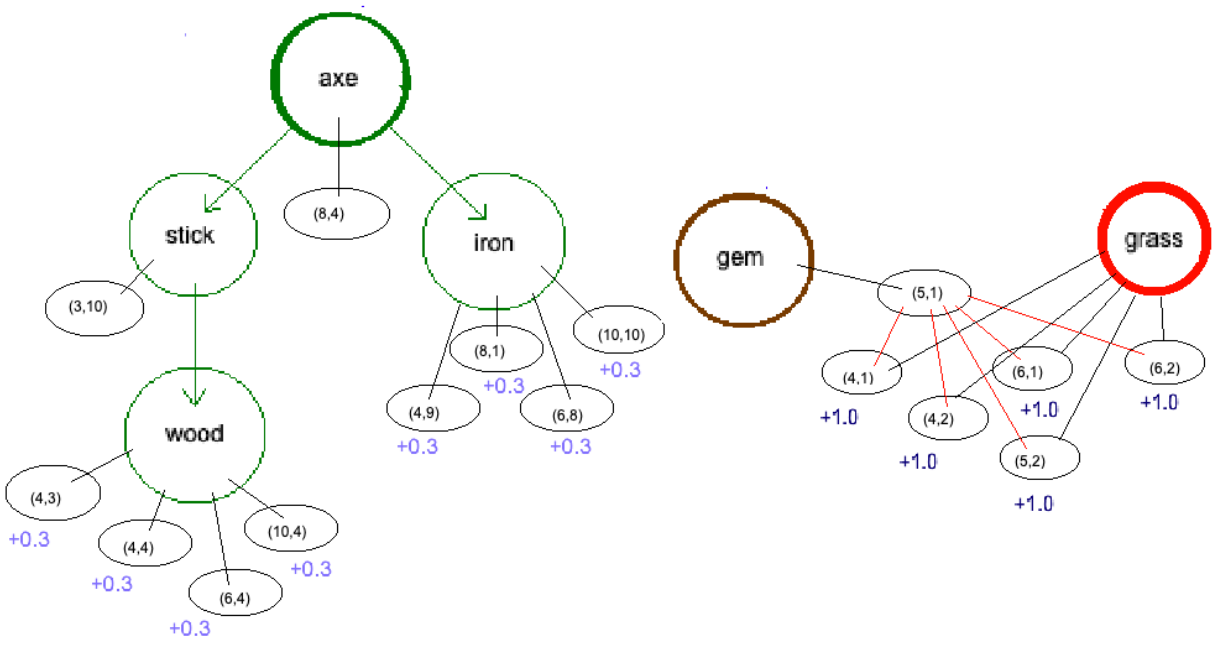
point of time during planning, the agent would choose between different nodes based on the estimated reward of executing a given node [2]. After each execution, we update the guiding graph with new costs and available nodes (Figure 6.5 – 6.5d). This is described in Algorithm 3 & Algorithm 4



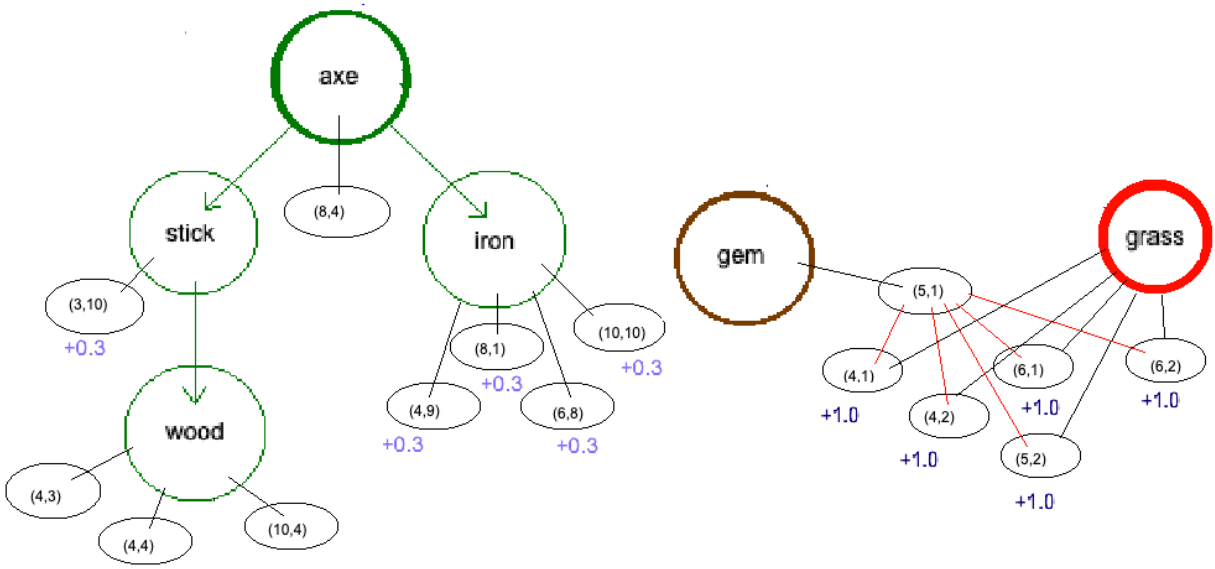
(a) Environment #3 (b) Graph guide

Fig. 6.5. In these figures we show how the assigned event cost is propagated through the graph, *(cont.)*

[2] We also use a discount factor of 0.9 while propagating the cost down the graph



(c) Graph guide with cost



(d) Graph guide with updated cost

Fig. 6.5. The assigned event cost is updated as the agent interacts with the environment

Algorithm 3 Assigning cost to nodes and gathering available options using graph guide

```
1: procedure GetOptions(Core_skeleton, Auxiliary_skeleton, Node_locations,
       Independent_events, Reusable_events, Current_environment)
2:     α ← 0.4 # Ratio of Bonus event rewards to Must execute event reward ratio
3:     γ ← 0.9 # Reward dilution ratio
4:     Final_event_reward_list ← [ ]
5:     Event_reward_list ← Initialise_events_rewards_list(Independent_events,
                                                  Reusable_events, α)
6:     while len(Event_reward_list) > 0 do
7:         Event, reward ← Pop_first_entry(Event_reward_list)
8:         event_reachable, inventory_satisfied ← check_event_conditions(Event,
                                        Auxiliary_skeleton, Current_environment)
9:         if event_reachable & inventory_satisfied then
10:            nodes_available ← Get_available_positions(Auxiliary_skeleton, Event)
11:            Add_nodes_to_list(Final_event_reward_list, nodes_available, reward)
12:        else
13:            if not inventory_satisfied then
14:                Prerequisite_event ← Get_prerequisite_event(Event, Core_skeleton)
15:                New_reward ← reward ×γ
16:                Add_nodes_to_list(Event_reward_list, Prerequisite_event,
                                        New_reward)
17:            end if
18:            if not event_reachable then
19:                New_events ← Get_events_to_unblock_location(Event,
                                        Auxiliary_skeleton)
20:                New_reward ← reward ×γ
21:                Add_nodes_to_list(Event_reward_list, New_events, New_reward)
22:            end if
23:        end if
24:    end while
25:    result: Final_event_reward_list
26: end procedure
```

Algorithm 4 Overall planning algorithm with graph guide

```
1: procedure ConvertToPlanNode(Environment, Cost_so_far, Reward_so_far)
2:     return: Make_node(Environment, Cost_so_far, Reward_so_far)
3: end procedure
4: procedure PlanWithGuide(Initial_environment, Event_sequence)
5:     Solutions ← [ ]
6:     To_search ← [ ]
7:     Initial_node ← ConvertToPlanNode(Initial_environment, 0, 0)
8:     To_search ← To_search ∪ Initial_node
9:     Initial_reachable_objects ← Get_initially_reachable_objects(Initial_environment)
10:    Independent_events, Reusable_events ← InferObjective(Initial_reachable_objects,
       Event_sequence)
11:    while len(To_search) > 0 do
12:        Current_node ← Pop_first_entry(To_search)
13:        Graph_guide ← ConstructGraphGuide(Current_node,
             Independent_events, Reusable_events)
14:        Options ← GetOptions(Graph_guide, Reusable_events,
             Independent_events, Current_environment)
15:        for option ∈ Options do
16:            New_node ← execute_option(Current_node, option)
17:            bool_value ← Node_is_costlier_than_existing_solutions(New_node,
                                                    Solutions)
18:            if bool_value then
19:                break
20:            end if
21:            To_search ← To_search ∪ option
22:            if Objective_satisfied(New_node, Independent_Events) then
23:                Solutions ← Solutions ∪ New_node
24:            end if
25:        end for
26:    end while
27:    result: Solutions
28: end procedure
```

We must note that the quality of the solution depends heavily on the completeness of the concept-base. As we can see in Table 6.1, initially the agent is not able to figure out a solution. Once it understands that the wood can be taken out of the environment, it proposes a sub-optimal solution. Finally, when it understands that grass is also a removable object, it gives the best solution.

Current conceptbase:

```
Object: gem
Transition: [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0 -1]
Pre-requisite: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 Got: gem. Used up: None
```

Proposed solution:

No solution

Current conceptbase:

```
Object: gem
Transition: [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0 -1]
Pre-requisite: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 Got: gem. Used up: None

Object: grass
Transition: [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
 0  0  0  0  0 -1]
Pre-requisite: [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 Got: grass. Used up: None
```

Proposed solution:

{get_grass, get_grass, get_gem}

Current conceptbase:

```
Object: gem
Transition: [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0 -1]
Pre-requisite: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 Got: gem. Used up: None

Object: grass
Transition: [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
 0  0  0  0  0 -1]
Pre-requisite: [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 Got: grass. Used up: None

Object: wood
Transition: [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
 0  0  0  0  0 -1]
Pre-requisite: [0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0]
 Got: wood. Used up: None
```

Proposed solution:

{get_wood, get_gem}

Tab. 6.1. We see how the quality of the concept base affects the optimality of the solution. If "get_wood" was present in the concept base, it comes up with a more optimal solution than if only "get_grass" is present in the concept base. If neither of the concepts are present, the agent doesn't understand how to complete the task

6.3. Comparison with other planning methods

	Planning efficiency	**Adaptability**	**Training cost**	**Pre-requisite**

Imitation Learning	Negligible cost, but high probability of failure	Barely adaptable	Multiple demonstrations	Low (network hyperparameters)
Inverse RL	Low	More adaptable than imitation learning, but no guarantee	Significantly more number of demonstration	Low (network hyperparameters)
RL	Low	Fully adaptable	Lots of episodes on practice environments	Low (network hyperparameters)
Classic dijkstra planning	Very poor, exponential with the number of choices	Fully adaptable to new situations	None	None
Our Method	Highly efficient	Highly adaptable, although dependent on the quality of the concept base	Low training data – either in terms of demonstrations or practice environment runs	State embedding functions, exhaustive set of skills, concept format

Tab. 6.2. Comparison with different methods

In this chapter, we explored how System 3 uses the learnt concepts to infer objectives from a demonstration and formulates an optimal plan when given the objective. The System 3 constructs a graph guide that represents the inventory and reachability dependencies of

different events. Having access to this significantly simplifies the process of planning in environments with hierarchical dependencies, and ensures that the agent can easily adapt to a vast range of environment configurations.

Chapitre 7

Experiments and Results

In this chapter we discuss the two kinds of problem settings where our framework provides the maximum utility. First, we consider the case of replicating demonstrations across different environment configurations. In this case, our framework is used to infer the objective of the demonstration in terms of events. These events are then replicated by the agent in the test environment configurations. In the second problem setting, we consider the case of planning when the objective is already mentioned in terms of a reward function. In this case, our framework computes the best possible way to achieve this objective using the concepts it has learned about the environment. In a hierarchical environment like CRAFT, we show how the ability to derive and utilise concepts is crucial for success in planning and replication. Using the concepts, our system is able to succeed in instances where most other approaches fail.

7.1. Learning from Demonstrations

In this section, we focus on learning the intention of the demonstrator and replicating that intention in different test environments. We assume infinite access to practice environments for any algorithm that can make use of it.

7.1.1. Single Demonstration

:

In this experiment, we have a single demonstration of an agent picking up gold after dismantling a stone using an axe. We apply our algorithm to infer the objective of the demonstration and attempt to replicate it across test environments having different sub-task dependencies. We perform an ablation study for our method to understand the effect of different aspects of our framework on the performance of the agent. We use the following algorithms:

(1) Our method (perfect concept base): We assume full access to the set of concepts that govern the environment dynamics.
(2) Our method (concept base learned from demo only): We assume that the concept base has been assembled only using the given demonstration.
(3) Our method (concept base learned from demo + random exploration): The concept base has been assembled using the demonstration and random exploration within the practice environment [1],
(4) Repeat skill sequence + Meta controller to choose the skill (Behaviour cloning at the skill level): We use our system to figure out the set of skills that were executed in the demonstrations, but instead of using System 3 to plan, we employ a neural network to learn the mapping from (state, inventory) to action
(5) Behaviour cloning at the action level: Just plain behaviour cloning with no extra modifications. There is an aspect of System 1 being used to decipher the sequence of actions.

First we explore how additional exploration in the practice environments can enhance the set of concepts learned when compared to learning from demonstration alone. These methods are compared to the baseline where the framework has access to the perfect concept base. Then we explore the contribution of the System 3's planning algorithm by comparing it with a meta controller that learns to pick skills based on the current environment state. Then we consider behaviour cloning at the level of primitive actions (UP, DOWN, RIGHT, LEFT, USE) which shows the contribution of planning at the level of skills compared to planing at the level of primitive actions. The results are displayed in Table 7.3.

[1] 3 distinct environment, 100 skills per environment

	% Environments Solved	Extra steps taken	Training time	Average deployment time
Our method (perfect concept base)	100%	0	n/a	2.58 secs
Our method (concept base learned from demo only)	18.18%	0	0.147 secs	0.047 secs
Our method (concept base learned from demo and random exploration)	51.51%	13.58	9.37 secs	43.25 secs
Meta controller to choose skills	15.15%	17.8	n/a	0.01 secs
Behaviour cloning at action level	0%	n/a	n/a	n/a

Tab. 7.1. Comparison with different methods

When the concept base is learned from a single demonstration, the agent has a very limited understanding of the environment. This limits the generalisation capability of the agent, but since the number of concepts learned is smaller, it doesn't have to spend much time finding the optimal path. This is seen in "Our method (concept base learned from demo only)" which takes only 0.047 secs to deploy but achieves an accuracy of 18.18%. As the concept base increases in size, so does its ability to solve a wider range of environment. "Our method (concept base learned form demo and random exploration)" solves 51.51% of the environments it is deployed in. However, time taken to deploy is very high – 43.25 secs on average. We hypothesise that due to missing concepts, it sometimes may take much longer to

arrive at a given state, and due to compounded concepts, it may spend unnecessary amount of time making objects it does not need.

Since we have a single demonstration, behaviour cloning at skill level and at the action level suffers from a data deficiency. In this case, we therefore simply replicate the skill sequence and the action sequence. The skill level replication works when all the relevant sub-task dependencies are the same, whereas it is highly improbable for direct action replication to work. We can see this in the table above. We can see that the best case scenario is planning at the level of skills while having the most accurate possible set of concepts.

7.1.2. Multiple demonstrations

The experimental setup is similar to the one in section 7.1.1, except that we have access to multiple demonstrations. In this experimental setup, we ensure that each demonstration has a different sub-task dependency structure. That is, to accomplish the task of getting gold, it has to ensure that a different set of events occur in the environment. We consider a similar ablation study to understand how planning at the level of skills, planning using System 3's algorithm, and planning with concept bases with different levels of accuracy individually affect the performance of the agent.

Algorithms being used :

(1) Our method (perfect concept base),
(2) Our method (concept base learned from demos only),
(3) Our method (concept base learned from demo + random exploration),
(4) Our method (concept base learned from demo + directed exploration),
(5) Repeat skill sequence + Meta controller to choose the skill (Behaviour cloning at the skill level)
(6) Behaviour cloning at the action level

Results:

	% Environments Solved	Extra steps taken	Training time	Average deployment time
Our method (perfect concept base)	100%	0	n/a	1.78 secs
Our method (concept base learned from demo only)	90.90%	0	1.79 secs	1.385 secs
Our method (concept base learned from demo and random exploration)	90.90%	0	10.59 secs	1.245 secs
Meta controller to choose skills	24.24%	8.0	23.17 secs	0.016 secs
Behaviour cloning at action level	3.03%	0	47.58 secs	0.007secs

Tab. 7.2. Comparison with different methods

Statistics for the framework with the perfect concept base remain the same as there is no additional training that takes place. In the case where the concept base is only updated using the demonstrations, since there are more demonstrations, it is able to show performance similar to the case when further exploration is allowed. This result has high variance and is susceptible to the variance and quality of the demonstrations. In both these cases, imperfect concept bases can lead to unnecessary optimisation in certain scenarios that can lead to agent getting stuck or performing unnecessary actions. Similar to the single demonstration replication setting, behaviour cloning at skill level would work with high probability if the sub-task dependencies remains the same. Behaviour cloning at the action level is highly sensitive to small changes in the environment and the performance is very unreliable. This

is obvious due to the fact that this environment is specifically designed to fail for algorithms that rely on rote memorisation. Generalisation capabilities of the behaviour cloning are also limited as the problem setting considers a very low data regime.

7.1.3. A note on baselines

In this work, we consider a scenario where we have very limited expert demonstrations that were gathered in environments which we can no longer access. This assumption itself disqualifies a significant number of methods from contention. This includes methods related to Inverse Reinforcement Learning [**9**], GAIL and its variants [**85, 165**], and even simpler methods in direct policy derivation [**151, 153**]. These methods require frequent interaction with the environment during their training procedure. Although we assume unlimited access to practice environments, we do not have access to the exact environments which were used to generate these demonstrations.

TACO [**166**] seems to be a contender given the similarity of our problem setup, but it actually represents the inverse of System 1. System 1 uses an exhaustive set of skills as a pre-requisite to observe a stream of experience and output the sequence of skills that were executed. TACO on the other hand, uses the sequence of skills as a pre-requisite (or sketches, as they call it) to output a set of sub-policies, or skills. Once we have the sequence of executed skills and the set of skills, the difference between our methods reduce to using System 3 versus using a meta controller to execute the sequence of skills. We have already included the latter as a baseline.

Methods like behaviour cloning, on the other hand, are destined to perform poorly in our problem setup. Not only do we have very limited data to train these algorithms, these algorithms are also required to handle long range sub-tasks dependencies. When it comes to learning efficient priors to operate in low data regimes, it is natural to look at work in meta learning and related fields. It is important to note that most of the methods that we mention in the related work (section 2.3), along with methods in the domain of OO-MDP (section 2.3.3) work towards improving sample efficiency in simple tasks in high dimensional input

regime. These methods are not necessarily equipped to tackle long range sub-task dependencies. Added to this, they do not explicitly focus on inferring objective from demonstrations. The objectives that the expert demonstrations aim to achieve are usually very straightforward. Therefore, it puts them at a disadvantage in our problem setup where there may be multiple core and auxiliary objectives.

Due to these considerations, we believe that behaviour cloning at action level and meta-controller operating at the skill level are sufficient baselines to provide a proof of concept for this framework. To fully explore the scope of this framework, we must examine how this framework would fare in a high dimensional state space setting. This would involving comparing various methods that replace the current version of System 1, System 2, and System 3. These settings may prove challenging as uncover underlying concepts may not be as trivial. However, this examination is currently out of scope for this publication.

7.2. Planning with a Reward

Planning can be done either at a high abstraction level using the learnt skills and concepts or using primitive environment actions. Planners that opt to act using primitive environment actions face a remarkably challenging problem as the environment configuration itself can change with object interactions. In most cases, unless we use deep neural networks that are allowed to train with large amounts of data, the problem itself is intractable. Even in these cases, we postulate that the network implicitly learns abstractions of the environment that it uses to predict the next action. As we mentioned before, this problem setting is designed specifically to fail for methods that do not work towards learning the core causal mechanisms of the environment. Therefore, when it comes to the question of quickly adapting to new task settings, such methods have a high probability of failure.

For these reasons, we only consider planners that can act at a high level of abstraction. In spite of acting at a higher abstraction level, classical AI methods like Dijkstra, A-Star or depth-first search that cannot work with the concepts learnt from systems 1 & 2 face an extremely challenging situation. This is because the cost of planning in this environment increases exponentially with the number of interactive objects. This is due to the fact that the

environment structure itself can change with the use of certain objects. For example, when a "stone" is removed from an environment configuration, certain objects that were blocked by the "stone" may now become accessible, and the cost of travelling to other objects may also change. This would change the overall setup of the problem and the agent has to include the overall state into its search space instead of just the agent itself.

Reinforcement learning methods at the skill level face a similar challenge. Since the sub-task dependencies for each environment may be different, any superfluous concepts that it may learn by over-fitting to specific environment settings would lead to failure in test cases. In a limited data domain, it is highly improbable for an RL agent to uncover core causal mechanisms, or concepts. Our method, which uses an imperfect concept base itself fails about half of the time.

In the comparison table below, we use a DQN [**130**] to compare against our methods which have access either an imperfect or a perfect concept base. The task involves getting "gold" in the same 33 environments which were used for previous experiments. These environments are designed such that each one of them have a different sub-task dependency graph with varying range of complexity.

Algorithms used:

(1) Our method (perfect concept base)
(2) Our method (concept base with random exploration)
(3) RL at the skill level

Results:

	% Test environments Solved	**Extra steps taken**	**Training time**	**Average deployment time**
Our method (perfect concept base)	100%	0	n/a	2.58 secs

Our method (concept base learned random exploration)	51.51%	13.58	9.37 secs	43.25 secs
RL at skill level	12.12%	17.8	507 secs	0.02 secs

Tab. 7.3. Comparison with different methods

As we can see, our method has very little time to train compared to RL at the skill level. The performance of our method, however, is highly dependent on the correctness of the concept base. Note that the RL agent simply outputs the skill given a state, whereas our method builds a sub-task dependency graph using the learnt concepts at each time-step where there is a change in the environmental structure. This is the primary reason for higher deployment time, but also higher accuracy compared to the RL agent.

In this chapter we saw how our framework performs in different problem settings. In the case of replicating demonstrations, we saw that the accuracy of the concept base can be improved by random exploration in practice environments which in turn improves the overall performance of the agent. In both planning and demonstration replication setting, the framework outperforms methods that do not plan using the understanding of the environment provided by the learnt concepts.

Chapitre 8

Conclusion

In this work we have introduced concepts as a means to accumulate and represent information about the environment. Based on the relationship between concepts, skills, and events, we proposed a framework for an agent to continually improve its capability in a given environment. The framework proposed is comprised of three sub-systems that tackle different aspects of the problem - System 1 works in making information from various modes (including partially observable demonstrations) accessible, System 2 helps in disentangling and consolidating the concepts of the environment, while System 3 makes use of the agent's current understanding of the world to effectively plan in novel environment configurations.

It is important to note that the effectiveness of this framework improves with time and experience. Concepts are an intuitive way to assimilate information as it is lightweight, interpretable, and easily transferable. An agent is able to function with even a partially developed set of concepts that can be improved whenever it is feasible. A neat example of this capability is the ability extract relevant concepts from partially observable demonstrations, combining it with existing information, and being able to generalise to novel scenarios. Since, the planning component of the framework is disentangled from the component that learns the concepts, the agent can learn in an online fashion. It can update its capabilities even as it is being deployed.

Some limitations of the current framework

In this work, we assume access to a initial set of skills that can exhaustively cover all the different behaviours that the agent can exhibit (See Def. **??**). However, there are several real world environments where it would be tedious and impractical to specify such an exhaustive set of skills. If one considers situations like cooking, or operating in a construction site, there are several objects which have to be dealt with in a specific way. For example, cutting a pineapple is specifically different from cutting most fruits and vegetables, because one has to also remove the thorns during the cutting process. If one had to exhaustively specify a set of skills in such environments, cutting for pineapple would have to be an separate skill by itself. In environments like the kitchen and the construction site, the variety of objects that could be potentially encountered are tremendously large. In such environments it would be better to compose skills at a lower level of hierarchy and form object-specific skills at a higher level. The cutting of pineapple, for example, can be described in terms of multiple cuts specifically parameterised according to the shape and texture of the pineapple. One can compose this information in both the concept base and the set of skills simultaneously. We leave out such environments in this work, as that would involve the need to consider switching between multiple levels of hierarchy for an optimal plan.

We also leave out environments where the agent has to deal with interruptions like charging battery and evading enemies. Success in such environments vastly depends on the ability of an agent to learn an effective hierarchical controller. This includes stochastic environments like Freeway, Seaquest, Pacman and real world environments like autonomous driving. In such environments, constructing and planning using a sub-task dependency graph does not completely solve the problem. Such environments may require utilising Linear Temporal Logic (LTL) [**155**] or training adaptive meta controllers [**143**].

For continuous control domains, like the Cheetah and Hopper [**181, 23**], it is theoretically impossible to specify an exhaustive set of skills as the action space itself is infinite. Although via directed exploration we can come up with a broad ranging set of skills [**49**]. We do not explicitly consider such environments, but we note that our proposed method of skill segmentation would still work with a decent margin of error.

We note that the utility of our framework is apparent when we're tackling hierarchical environments. However, one may not find much benefit using this framework for strategy

games like Go, Chess or multi-agent games like DotA and StarCRAFT. Here, it is not intuitive to abstract strategic plays into an exhaustive set of skills, and we are not sure if this is the right way to look at this problem. On the other hand, the core concepts of a game environment are easy to obtain and can be much easily incorporated as priors.

Another limitation in the current version of the framework is having a fixed set of skills that is decided at the start of deployment. The framework could potentially function better if there was a robust way of improving the quality of skills. The trade-off between skill interpretability and exhaustiveness of the skill set is something we have not considered in this work.

8.1. Possible avenues for future Work

Demonstrating the working of this framework in the environment CRAFT involved an assumption that we have access to an exhaustive set of initial skills. The exhaustiveness of the skill set ensures that the segments obtained from System 1 has all the relevant information necessary for the formation of concepts. As discussed in section 2.2.1, there are several ways to relax this assumption. There are works that learn skills during the exploration phase [**49**], or while segmenting the demonstrations [**166**]. An exhaustive set of skills is non-trivial to define and it is especially hard when we're discovering skills or behaviours in an unsupervised way. A successful way to do this is by specifying that the learnt skills should be as diverse from the one another as possible [**49**]. We also assume that the learnt set of skills is fixed and don't work towards refining individual skills, decomposing skills into skills of a lower abstraction level, or composing skills to form more abstract skills. Some works have worked towards this on a preliminary level [**155, 131**], but a generalised approach to this problem is missing.

Refining and updating skills at different levels of abstractions can be done in conjunction with developing concepts. In section 1.1, we talked about the intricate relationship between skills, events, and concepts. This relationship is the basis on which our framework is designed. In an ideal situation, skills, events, and concepts are learned simultaneously without any prior assumptions. But as we discussed, it is highly non-intuitive for even humans to do this

effectively. We as humans are able to utilise prior knowledge to not just execute various tasks effectively, but also to further refine and develop our understanding about the environment. In the future we would like to explore how one could start with a very limited amount of information about the environment. We would like to explore what is the minimum amount of information that is just enough to define the problem well, and how one could use this to gradually develop an expansive set of concepts and skills. As described in the lifelong learning book [**32**], an advanced lifelong learning agent should also be able to discover new problem statements that are worth tackling and refine relevant concepts in a self-supervised way.

In this work, we assumed access to the format of the concepts, and State Embedding Functions that are used to extract information from the segments. We note that it is non-trivial to automate the process of discovering these from scratch. Progress in this aspect of the problem is tied to the progress in unsupervised learning, and knowledge representation in lifelong learning. Improvement in these aspects would lead to a lot of dividends in terms of real world applicability and generalisation capabilities of algorithms. Another aspect we would like explicitly explore is learning concepts and planning across different levels of hierarchy. The options framework [**178**] makes it very flexible for the planner to operate at different levels, however the size of the search space expands exponentially. It takes a lot of effort for an algorithm to learn the value function for different options such that the planner can function effectively. It is also not trivial to update the options when there is an influx of new information. In future work, we hope to experiment with planners that can easily switch between various levels of hierarchy. We postulate that effective representation and usage of concepts can work towards improving the quality and efficiency of existing planners.

8.1.1. Other ways of skill segmentation

In this work, we consider hard-coded skill discriminators. Although, they most use information from the corresponding skills, we must work towards a more generalisable way of skill segmentation. An self-supervised way to do this would be use trajectory embedding techniques [**97**] to map the behaviour of different skill to controllable latent variables. When a

demonstration is encountered by the system, it can observe the behaviour of latent variables to detect segmentation points and classification of the segments.

Developing reliable skill embedding is one way to go about this. Here, we take the actions to the sub-policy space, and work towards a solution in that sub-policy domain. We can also go the other way by taking the sub-policies into the action space and defining relevant loss functions in the discrete action domain. Example of such loss functions can be found in the trajectory comparison literature [**39**], although it is not trivial to ignore to state space artifacts that we consider as irrelevant.

ComPILE [**98**] is an important advancement in an agent's ability to divide demonstrations into segments that can be used to learn skills. The basic information extraction block in ComPILE is a VAE. It iteratively predicts the segmentation and the encodes the segment into a latent variable z, which is then decoded into low level actions. The information extraction mechanism can be replaced with other generative modelling methods like GANs. This avenue is yet to be explored.

One can also explore adding different layers of complexity in a UPN-like-architecture [**172**]. Where instead of relying on learning transition function on a action level, we can learn transition function on a skill level. Instead of having to replicate the demonstration in terms of actions, we can work towards having skills as the basis of creating a demonstration. Since, the UPNs learn task specific features from state, these features may be able to ignore the state specific details that do not matter at higher levels of abstraction.

8.1.2. Lifelong Learning Agents

This work forms a stepping stone towards consolidating different aspects of machine learning into a single problem statement. The core of aspect of intelligent systems is being able to re-use learnt information to continually upgrade one's capability. This work highlights the progress made in various domains to achieve this, and introduces the notion of "concepts" that can potentially simplify this endeavour.

This work can be grouped with other works that tackle hierarchical environments using a framework based approach [**131, 171**]. In this respect, there are several avenues that are yet to be explored such as extending the method to environments that resemble the real world more closely and improving the robustness of information extraction techniques. As mentioned in the Lifelong learning book [**32**], a critical aspect of information extraction is affirming the correctness of extracted knowledge and understanding when a piece of information can be applied. These aspects have been explored in the context of Natural Language Processing [**31, 128**], and are yet to be explored in the RL setting.

We hope that this work encourages researchers to look at hierarchical domains from a framework point of view. The use of concepts specifically demonstrates how prior knowledge about the environment can be efficiently utilised and help the agent to continually improve its understanding about the environment. Specifically, in CRAFT environment, we demonstrate the drastic difference this makes in terms of the agent's generalisation capability and planning efficiency.

Bibliographie

[1] Joshua Achiam : Spinning Up in Deep Reinforcement Learning. 2018.

[2] Joshua Achiam, Harrison Edwards, Dario Amodei et Pieter Abbeel : Variational option discovery algorithms. *CoRR*, abs/1807.10299, 2018.

[3] William S. Agnew et Pedro M. Domingos : Self-supervised object-level deep reinforcement learning. *ArXiv*, abs/2003.01384, 2020.

[4] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté et R. Devon Hjelm : Unsupervised state representation learning in atari. *In NeurIPS*, 2019.

[5] Garrett Andersen, Peter Vrancx et Haitham Bou-Ammar : Learning high-level representations from demonstrations. *CoRR*, abs/1802.06604, 2018.

[6] Jacob Andreas, Dan Klein et Sergey Levine : Modular multitask reinforcement learning with policy sketches. *CoRR*, abs/1611.01796, 2016.

[7] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul et Nando de Freitas : Learning to learn by gradient descent by gradient descent. *ArXiv*, abs/1606.04474, 2016.

[8] Brenna D. Argall, Sonia Chernova, Manuela Veloso et Brett Browning : A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009.

[9] Saurabh Arora et Prashant Doshi : A survey of inverse reinforcement learning: Challenges, methods and progress. *ArXiv*, abs/1806.06877, 2018.

[10] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage et Anil Anthony Bharath : A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017.

[11] Pierre-Luc Bacon, Jean Harb et Doina Precup : The option-critic architecture. *CoRR*, abs/1609.05140, 2016.

[12] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo et Charles Blundell : Agent57: Outperforming the atari human benchmark. 2020.

[13] Somil Bansal, Roberto Calandra, Sergey Levine et Claire Tomlin : MBMF: model-based priors for model-free reinforcement learning. *CoRR*, abs/1709.03153, 2017.

[14] Elias Bareinboim, Andrew Forney et Judea Pearl : Bandits with unobserved confounders: A causal approach. *In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1342–1350, Cambridge, MA, USA, 2015. MIT Press.

[15] André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, David Silver et Hado van Hasselt : Successor features for transfer in reinforcement learning. *In NIPS*, 2016.

[16] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende et Koray Kavukcuoglu : Interaction networks for learning about objects, relations and physics. *ArXiv*, abs/1612.00222, 2016.

[17] Marc G. Bellemare, Will Dabney et Rémi Munos : A distributional perspective on reinforcement learning. *In ICML*, 2017.

[18] Marc G. Bellemare, Yavar Naddaf, Joel Veness et Michael Bowling : The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012.

[19] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton et Rémi Munos : Unifying count-based exploration and intrinsic motivation. *CoRR*, abs/1606.01868, 2016.

[20] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub W. Pachocki, Michael Petrov, Henrique Pond'e de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski et Susan Zhang : Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019.

[21] Léon Bottou, Jonas Peters, Joaquin Quiñonero Candela, Denis X. Charles, D. Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard et Ed Snelson : Counterfactual reasoning and learning systems: The example of computational advertising. *J. Mach. Learn. Res.*, 14(1):3207–3260, janvier 2013.

[22] Martin M. Broadwell : Teaching for learning (xvi). *The Gospel Guardian*, 1969.

[23] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang et Wojciech Zaremba : Openai gym. *ArXiv*, abs/1606.01540, 2016.

[24] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo et Honglak Lee : Sample-e□cient reinforcement learning with stochastic ensemble value expansion. *In NeurIPS*, 2018.

[25] Yuri Burda, Harrison Edwards, Deepak Pathak, Amos J. Storkey, Trevor Darrell et Alexei A. Efros : Large-scale study of curiosity-driven learning. *CoRR*, abs/1808.04355, 2018.

[26] Yuri Burda, Harrison Edwards, Amos J. Storkey et Oleg Klimov : Exploration by random network distillation. *CoRR*, abs/1810.12894, 2018.

[27] Arunkumar Byravan, Jost Tobias Springenberg, Abbas Abdolmaleki, Roland Hafner, Michael Neunert, Thomas Lampe, Noah Siegel, Nicolas Manfred Otto Heess et Martin A. Riedmiller :

Imagined value gradients: Model-based policy optimization with transferable latent dynamics models. *ArXiv*, abs/1910.04142, 2019.

[28] Michael Chang, Tomer Ullman, Antonio Torralba et Joshua B. Tenenbaum : A compositional object-based approach to learning physical dynamics. *ArXiv*, abs/1612.00341, 2017.

[29] Simyung Chang, Young Joon Yoo, Jaeseok Choi et Nojun Kwak : Knowledge sharing for reinforcement learning: Writing a book. *ArXiv*, abs/1709.01308, 2017.

[30] Yutian Chen, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Timothy P. Lillicrap et Nando de Freitas : Learning to learn for global optimization of black box functions. *ArXiv*, abs/1611.03824, 2016.

[31] Zhiyuan Chen et Bing Liu : Mining topics in documents: standing on the shoulders of big data. *In KDD '14*, 2014.

[32] Zhiyuan Chen, Bing Liu, Ronald Brachman, Peter Stone et Francesca Rossi : *Lifelong Machine Learning*. Morgan Claypool Publishers, 2nd édition, 2018.

[33] Silvia Chiappa, Sébastien Racanière, Daan Wierstra et Shakir Mohamed : Recurrent environment simulators. *CoRR*, abs/1704.02254, 2017.

[34] Kamil Ciosek, Quan Van Vuong, Robert Tyler Loftin et Katja Hofmann : Better exploration with optimistic actor-critic. *In NeurIPS*, 2019.

[35] Ignasi Clavera, Anusha Nagabandi, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine et Chelsea Finn : Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *In International Conference on Learning Representations*, 2019.

[36] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour et Pieter Abbeel : Model-based reinforcement learning via meta-policy optimization. *In CoRL*, 2018.

[37] John D. Co-Reyes, Yuxuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel et Sergey Levine : Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *CoRR*, abs/1806.02813, 2018.

[38] Mark Collier et Joeran Beel : Memory-augmented neural networks for machine translation. *In MTSummit*, 2019.

[39] Marco Cuturi et Mathieu Blondel : Soft-dtw: a differentiable loss function for time-series. *In ICML*, 2017.

[40] Guy R. Davidson et Brenden M. Lake : Investigating simple object representations in model-free deep reinforcement learning. *ArXiv*, abs/2002.06703, 2020.

[41] Marc Peter Deisenroth et Carl Edward Rasmussen : Pilco: A model-based and data-efficient approach to policy search. *In Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 465–472, Madison, WI, USA, 2011. Omnipress.

[42] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez et Ste□en Udluft : Learning and policy search in stochastic dynamical systems with bayesian neural networks. *ArXiv*, abs/1605.07127, 2016.

[43] Edsger W Dijkstra : A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[44] Carlos Diuk, Andre Cohen et Michael L. Littman : An object-oriented representation for e□cient reinforcement learning. *In Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 240–247, New York, NY, USA, 2008. Association for Computing Machinery.

[45] Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel et Wojciech Zaremba : One-shot imitation learning. *In NIPS*, 2017.

[46] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever et Pieter Abbeel : Rl^2 : $Fastreinforcementlearningviaslowreinforcementlearning.ArXiv, abs/1611.02779, 2016.$

[47] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee et Sergey Levine : Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *ArXiv*, abs/1812.00568, 2018.

[48] Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles et Bernard Ghanem : Daps: Deep action proposals for action understanding. *In ECCV*, 2016.

[49] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz et Sergey Levine : Diversity is all you need: Learning skills without a reward function. *CoRR*, abs/1802.06070, 2018.

[50] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez et Sergey Levine : Model-based value estimation for e□cient model-free reinforcement learning. *CoRR*, abs/1803.00101, 2018.

[51] Chelsea Finn, Pieter Abbeel et Sergey Levine : Model-agnostic meta-learning for fast adaptation of deep networks. *In ICML*, 2017.

[52] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel et Sergey Levine : One-shot visual imitation learning via meta-learning. *ArXiv*, abs/1709.04905, 2017.

[53] Carlos Florensa, Yan Duan et Pieter Abbeel : Stochastic neural networks for hierarchical reinforcement learning. *ArXiv*, abs/1704.03012, 2017.

[54] Andrew Forney, Judea Pearl et Elias Bareinboim : Counterfactual data-fusion for online reinforcement learners. *In Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1156–1164. JMLR.org, 2017.

[55] Roy Fox, Sanjay Krishnan, Ion Stoica et Ken Goldberg : Multi-level discovery of deep options. *CoRR*, abs/1703.08294, 2017.

[56] Roy Fox, Richard Shin, Sanjay Krishnan, Ken Goldberg, Dawn Song et Ion Stoica : Parametrized hierarchical procedures for neural programming. *In International Conference on Learning Representations*, 2018.

[57] Vincent François-Lavet, Yoshua Bengio, Doina Precup et Joelle Pineau : Combined reinforcement learning via abstract representations. *CoRR*, abs/1809.04506, 2018.

[58] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel et John Schulman : Meta learning shared hierarchies. *ArXiv*, abs/1710.09767, 2018.

[59] Justin Fu, John D. Co-Reyes et Sergey Levine : EX2: exploration with exemplar models for deep reinforcement learning. *CoRR*, abs/1703.01260, 2017.

[60] Scott Fujimoto, Herke van Hoof et David Meger : Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477, 2018.

[61] Alexandre Galashov, Siddhant M. Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech Czarnecki, Yee Whye Teh, Razvan Pascanu et Nicolas Manfred Otto Heess : Information asymmetry in kl-regularized rl. *ArXiv*, abs/1905.01240, 2019.

[62] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum et Marc G. Bellemare : Deepmdp: Learning continuous latent space models for representation learning. *CoRR*, abs/1906.02736, 2019.

[63] Vikash Goel, Jameson Weng et Pascal Poupart : Unsupervised video object segmentation for deep reinforcement learning. *In NeurIPS*, 2018.

[64] Solomon W. Golomb et Leonard D. Baumert : Backtrack programming. *J. ACM*, 12(4):516–524, octobre 1965.

[65] Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Matthew M Botvinick, Hugo Larochelle, Sergey Levine et Yoshua Bengio : Infobot: Transfer and exploration via the information bottleneck. *ArXiv*, abs/1901.10902, 2019.

[66] Alex Graves : *Supervised Sequence Labelling with Recurrent Neural Networks*. 2011.

[67] Alex Graves : Hallucinations with rnns, 2015.

[68] Alex Graves, Greg Wayne et Ivo Danihelka : Neural turing machines. *ArXiv*, abs/1410.5401, 2014.

[69] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loïc Matthey, Matthew M Botvinick et Alexander Lerchner : Multi-object representation learning with iterative variational inference. *In ICML*, 2019.

[70] Karol Gregor, Danilo Jimenez Rezende, Frederic Besse, Yan Wu, Hamza Merzic et Aaron Oord : Shaping belief states with generative environment models for rl. 06 2019.

[71] Karol Gregor, Danilo Jimenez Rezende et Daan Wierstra : Variational intrinsic control. *CoRR*, abs/1611.07507, 2016.

[72] Matthew Guzdial, Boyang Li et Mark O. Riedl : Game engine learning from video. *In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3707–3713, 2017.

[73] Matthew Guzdial, Boyang Li et Mark O. Riedl : Game engine learning from video. *In Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 3707–3713. AAAI Press, 2017.

[74] David Ha et Jürgen Schmidhuber : World models. *CoRR*, abs/1803.10122, 2018.

[75] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel et Sergey Levine : Soft actor-critic: O□-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, abs/1801.01290, 2018.

[76] Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba et Mohammad Norouzi : Dream to control: Learning behaviors by latent imagination. *ArXiv*, abs/1912.01603, 2020.

[77] Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee et James Davidson : Learning latent dynamics for planning from pixels. *CoRR*, abs/1811.04551, 2018.

[78] Peter Hart, Nils Nilsson et Bertram Raphael : A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[79] Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav Sukhatme et Joseph J Lim : Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. *In* I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems 30*, pages 1235–1245. Curran Associates, Inc., 2017.

[80] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess et Martin Riedmiller : Learning an embedding space for transferable robot skills. *In International Conference on Learning Representations*, 2018.

[81] Nicholas Hay, Michael Stark, Alexander Schlegel, Carter Wendelken, Dennis Park, Eric Purdy, Tom Silver, D. Scott Phoenix et Dileep George : Behavior is everything: Towards representing concepts with sensorimotor contingencies. *In AAAI*, 2018.

[82] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar et David Silver : Rainbow: Combining improvements in deep reinforcement learning. *ArXiv*, abs/1710.02298, 2018.

[83] Irina Higgins, Arka Pal, Andrei A. Rusu, Loïc Matthey, Christopher Burgess, Alexander Pritzel, Matthew M Botvinick, Charles Blundell et Alexander Lerchner : Darla: Improving zero-shot transfer in reinforcement learning. *In ICML*, 2017.

[84] Geo□rey E. Hinton : Using fast weights to deblur old memories. 1987.

[85] Jonathan Ho et Stefano Ermon : Generative adversarial imitation learning. *In NIPS*, 2016.

[86] Sepp Hochreiter et Jürgen Schmidhuber : Long short-term memory. *Neural Comput.*, 9(8):1735–1780, novembre 1997.

[87] Sepp Hochreiter, A. Steven Younger et Peter R. Conwell : Learning to learn using gradient descent. *In Proceedings of the International Conference on Artificial Neural Networks*, ICANN '01, page 87–94, Berlin, Heidelberg, 2001. Springer-Verlag.

[88] Zhang-Wei Hong, Joni Pajarinen et Jan Peters : Model-based lookahead reinforcement learning. *ArXiv*, abs/1908.06012, 2019.

[89] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck et Pieter Abbeel : Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *CoRR*, abs/1605.09674, 2016.

[90] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese et Juan Carlos Niebles : Neural task graphs: Generalizing to unseen tasks from a single video demonstration. *CoRR*, abs/1807.03480, 2018.

[91] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan et Chrisina Jayne : Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, avril 2017.

[92] Maximilian Ilse, Jakub M. Tomczak, Christos Louizos et M. Welling : Diva: Domain invariant variational autoencoders. *ArXiv*, abs/1905.10427, 2019.

[93] Alex Irpan : Deep reinforcement learning doesn't work yet. `https://www.alexirpan.com/2018/02/14/rl-hard.html`, 2018.

[94] Stephen James, Michael Bloesch et Andrew J. Davison : Task-embedded control networks for few-shot imitation learning. *In CoRL*, 2018.

[95] Michael Janner, Justin Fu, Marvin Zhang et Sergey Levine : When to trust your model: Model-based policy optimization. *In NeurIPS*, 2019.

[96] Ken Kansky, Tom Silver, David A. Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, D. Scott Phoenix et Dileep George : Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *ArXiv*, abs/1706.04317, 2017.

[97] Diederik P. Kingma et Max Welling : Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[98] Thomas Kipf, Yujia Li, Hanjun Dai, Vinícius Flores Zambaldi, Edward Grefenstette, Pushmeet Kohli et Peter W. Battaglia : Compositional imitation learning: Explaining and executing one task at a time. *ArXiv*, abs/1812.01483, 2018.

[99] Bangalore Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad Al Sallab, Senthil Kumar Yogamani et Patrick Pérez : Deep reinforcement learning for autonomous driving: A survey. *ArXiv*, abs/2002.00444, 2020.

[100] Ranjay Krishna, Kenji Hata, Frederic Ren, Fei-Fei Li et Juan Carlos Niebles : Dense-captioning events in videos. *CoRR*, abs/1705.00754, 2017.

[101] Sanjay Krishnan, Roy Fox, Ion Stoica et Ken Goldberg : DDCO: discovery of deep continuous options forrobot learning from demonstrations. *CoRR*, abs/1710.05421, 2017.

[102] Sanjay Krishnan, Animesh Garg, Sachin Patil, Colin Lea, Gregory Hager, Pieter Abbeel et Ken Goldberg : Transition state clustering. *Int. J. Rob. Res.*, 36(13-14):1595–1618, décembre 2017.

[103] Alex Krizhevsky, Ilya Sutskever et Geo□rey E. Hinton : Imagenet classification with deep convolutional neural networks. *In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[104] O. Kroemer, C. Daniel, G Neumann, H. van Hoof et J. Peters : Towards learning hierarchical skills for multi-phase manipulation tasks. *In IEEE International Conference on Robotics and Automation*, pages 1503 – 1510, 2015.

[105] Oliver Kroemer, Scott Niekum et George Dimitri Konidaris : A review of robot learning for manipulation: Challenges, representations, and algorithms. *CoRR*, abs/1907.03146, 2019.

[106] Tejas Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman et Volodymyr Mnih : Unsupervised learning of object keypoints for perception and control. *ArXiv*, abs/1906.11883, 2019.

[107] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar et Pieter Abbeel : Model-ensemble trust-region policy optimization. *CoRR*, abs/1802.10592, 2018.

[108] Brenden M. Lake, Ruslan Salakhutdinov et Joshua B. Tenenbaum : Human-level concept learning through probabilistic program induction. *Science*, 350:1332–1338, 2015.

[109] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum et Samuel Gershman : Building machines that learn and think like people. *The Behavioral and brain sciences*, 40:e253, 2018.

[110] Nicholas C. Landolfi, Garrett Thomas et Tengyu Ma : A model-based approach for sample-e□cient multi-task reinforcement learning. *ArXiv*, abs/1907.04964, 2019.

[111] Finnian Lattimore, Tor Lattimore et Mark D. Reid : Causal bandits: Learning good interventions via causal inference. *In Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1189–1197, USA, 2016. Curran Associates Inc.

[112] Miguel Lázaro-Gredilla, Dianhuan Lin, J. Swaroop Guntupalli et Dileep George : Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. *Science Robotics*, 4, 2019.

[113] Sanghack Lee et Elias Bareinboim : Structural causal bandits: Where to intervene? *In NeurIPS*, 2018.

[114] Sanghack Lee et Elias Bareinboim : Structural causal bandits with non-manipulable variables. *In The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4164–4172, 2019.

[115] Sergey Levine et Vladlen Koltun : Guided policy search. *In ICML*, 2013.

[116] Ke Li et Jitendra Malik : Learning to optimize. *ArXiv*, abs/1606.01885, 2016.

[117] Ke Li et Jitendra Malik : Learning to optimize neural nets. *ArXiv*, abs/1703.00441, 2017.

[118] Yuezhang Li, Katia P. Sycara et Rahul Iyer : Object-sensitive deep reinforcement learning. *ArXiv*, abs/1809.06064, 2017.

[119] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver et Daan Wierstra : Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[120] Zhixuan Lin, Yi-Fu Wu, Skand Vishwanath Peri, Weihao Sun, Gautam Singh, Fei Deng, Jindong Jiang et Sungjin Ahn : Space: Unsupervised object-oriented scene representation via spatial attention and decomposition. *ArXiv*, abs/2001.02407, 2019.

[121] Sahisnu Mazumder, Nianzu Ma et Bing Liu : Towards a continuous knowledge learning engine for chatbots. *ArXiv*, abs/1802.06024, 2018.

[122] Rowan McAllister et Carl E. Rasmussen : Data-e□cient reinforcement learning in continuous-state pomdps. *ArXiv*, abs/1602.02523, 2016.

[123] Rowan McAllister et Carl Edward Rasmussen : Improving pilco with bayesian neural network dynamics models. 2016.

[124] R. McFarlane : A survey of exploration strategies in reinforcement learning. 2003.

[125] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine et Chelsea Finn : Guided meta-policy search. *CoRR*, abs/1904.00956, 2019.

[126] Donald Michie, Michael Bain et Jean Hayes-Michie : Cognitive models from subcognitive skills. *IEE control engineering series*, 44:71–99, 1990.

[127] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen et Pieter Abbeel : Meta-learning with temporal convolutions. *ArXiv*, abs/1707.03141, 2017.

[128] Tom Michael Mitchell, William W. Cohen, Estevam R. Hruschka, Partha P. Talukdar, Bo Yang, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matt Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil Antonios Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves et Joel Welling : Never-ending learning. *Communications of the ACM*, 61:103 – 115, 2015.

[129] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver et Koray Kavukcuoglu : Asynchronous methods for deep reinforcement learning. *In* Maria Florina Balcan et Kilian Q. Weinberger, éditeurs : *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 de *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[130] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg et Demis Hassabis : Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, février 2015.

[131] Philippe Morere, Lionel Ott et Fabio Ramos : Learning to plan hierarchically from curriculum. *CoRR*, abs/1906.07371, 2019.

[132] Tsendsuren Munkhdalai et Hong Yu : Meta networks. *Proceedings of machine learning research*, 70:2554–2563, 2017.

[133] P. W. Munro : A dual back-propagation scheme for scalar reinforcement learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA*, pages 165–176, 1987.

[134] Anusha Nagabandi, Chelsea Finn et Sergey Levine : Deep online learning via meta-learning: Continual adaptation for model-based rl. *ArXiv*, abs/1812.07671, 2019.

[135] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing et Sergey Levine : Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596, 2017.

[136] Gergely Neu et Csaba Szepesvari : Apprenticeship learning using inverse reinforcement learning and gradient methods. *In UAI*, 2007.

[137] N. Nguyen et B. Widrow : The truck backer-upper: An example of self learning in neural networks. *In Proceedings of the International Joint Conference on Neural Networks*, pages 357–363. IEEE Press, 1989.

[138] Alex Nichol, Joshua Achiam et John Schulman : On first-order meta-learning algorithms. *ArXiv*, abs/1803.02999, 2018.

[139] Scott Niekum, Sachin Chitta, Andrew Barto, Bhaskara Marthi et Sarah Osentoski : Incremental semantically grounded learning from demonstration. 06 2013.

[140] Nils J. Nilsson : *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co., 1971.

[141] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis et Satinder P. Singh : Action-conditional video prediction using deep networks in atari games. *CoRR*, abs/1507.08750, 2015.

[142] Junhyuk Oh, Satinder Singh et Honglak Lee : Value prediction network. *CoRR*, abs/1707.03497, 2017.

[143] Junhyuk Oh, Satinder P. Singh, Honglak Lee et Pushmeet Kohli : Zero-shot task generalization with multi-task deep reinforcement learning. *CoRR*, abs/1706.05064, 2017.

[144] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jadwiga Tworek, Peter Welinder, Lilian Weng, Qi-Ming Yuan, Wojciech Zaremba et Lefei Zhang : Solving rubik's cube with a robot hand. *ArXiv*, abs/1910.07113, 2019.

[145] Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord et Rémi Munos : Count-based exploration with neural density models. *CoRR*, abs/1703.01310, 2017.

[146] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros et Trevor Darrell : Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363, 2017.

[147] Judea Pearl : *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, USA, 2000.

[148] Judea Pearl et Dana Mackenzie : *The Book of Why: The New Science of Cause and E ect*. Basic Books, Inc., USA, 1st édition, 2018.

[149] Vitchyr Pong, Shixiang Gu, Murtaza Dalal et Sergey Levine : Temporal di erence models: Model-free deep RL for model-based control. *CoRR*, abs/1802.09081, 2018.

[150] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn et Sergey Levine : E cient o -policy meta-reinforcement learning via probabilistic context variables. *CoRR*, abs/1903.08254, 2019.

[151] Aaditya Ramdas, Jianbo Chen, Martin J. Wainwright et Michael I. Jordan : Dagger: A sequential algorithm for fdr control on dags. *ArXiv*, abs/1709.10250, 2017.

[152] T. Robinson et F. Fallside : Dynamic reinforcement driven error propagation networks with application to game playing. 1989.

[153] Stéphane Ross et J. Andrew Bagnell : Reinforcement and imitation learning via interactive no-regret learning. *ArXiv*, abs/1406.5979, 2014.

[154] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu et Raia Hadsell : Progressive neural networks. *ArXiv*, abs/1606.04671, 2016.

[155] Himanshu Sahni, Saurabh Kumar, Farhan Tejani et Charles L. Isbell Jr. : Learning to compose skills. *CoRR*, abs/1711.11289, 2017.

[156] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra et Timothy Lillicrap : Meta-learning with memory-augmented neural networks. *In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1842–1850. JMLR.org, 2016.

[157] Adam Santoro, Sergey Bartunov, Matthew M Botvinick, Daan Wierstra et Timothy P. Lillicrap : One-shot learning with memory-augmented neural networks. *ArXiv*, abs/1605.06065, 2016.

[158] J. Schmidhuber : An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. *In Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 2, pages 253–258, 1990.

[159] Jürgen Schmidhuber : Reinforcement learning in markovian and non-markovian environments. *In Proceedings of the 1990 Conference on Advances in Neural Information Processing Systems 3*, NIPS-3, page 500–506, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[160] Jürgen Schmidhuber : A possibility for implementing curiosity and boredom in model-building neural controllers. *In Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, page 222–227, Cambridge, MA, USA, 1991. MIT Press.

[161] Jürgen Schmidhuber : On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models. *CoRR*, abs/1511.09249, 2015.

[162] Jonathan Scholz, Martin Levihn, Charles Lee Isbell et David Wingate : A physics-based model prior for object-oriented mdps. *In ICML*, 2014.

[163] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford et Oleg Klimov : Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[164] Rajat Sen, Karthikeyan Shanmugam, Alexandros G. Dimakis et Sanjay Shakkottai : Identifying best interventions through online importance sampling. *In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3057–3066, 2017.

[165] Arjun Sharma, Mohit Sharma, Nicholas Rhinehart et Kris M. Kitani : Directed-info GAIL: learning hierarchical policies from unsegmented demonstrations using directed information. *CoRR*, abs/1810.01266, 2018.

[166] Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson et Ingmar Posner : TACO: learning task decomposition via temporal alignment for control. *CoRR*, abs/1803.01840, 2018.

[167] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel et Demis Hassabis : Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, janvier 2016.

[168] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan et Demis Hassabis : Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[169] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David P. Reichert, Neil C. Rabinowitz, André Barreto et Thomas Degris : The predictron: End-to-end learning and planning. *CoRR*, abs/1612.08810, 2016.

[170] Sungryull Sohn, Junhyuk Oh et Honglak Lee : Multitask reinforcement learning for zero-shot generalization with subtask dependencies. *CoRR*, abs/1807.07665, 2018.

[171] SungRyull Sohn, Hyunjae Woo, Jong wook Choi et Honglak Lee : Meta reinforcement learning with autonomous inference of subtask dependencies. *ArXiv*, abs/2001.00248, 2020.

[172] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine et Chelsea Finn : Universal planning networks. *CoRR*, abs/1804.00645, 2018.

[173] Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram et Joseph Lim : Neural program synthesis from diverse demonstration videos. *In* Jennifer Dy et Andreas Krause, éditeurs : *Proceedings of the 35th International Conference on Machine Learning*, volume 80 de *Proceedings of Machine Learning Research*, pages 4790–4799, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[174] Wen Sun, Anirudh Vemula, Byron Boots et J. Andrew Bagnell : Provably e□cient imitation learning from observation alone. *CoRR*, abs/1905.10948, 2019.

[175] Richard S. Sutton : Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *In In Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

[176] Richard S. Sutton et Andrew G. Barto : *Reinforcement Learning: An Introduction*. The MIT Press, second édition, 2018.

[177] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White et Doina Precup : Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. *In The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, page 761–768, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[178] Richard S. Sutton, Doina Precup et Satinder Singh : Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, août 1999.

[179] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck et Pieter Abbeel : #exploration: A study of count-based exploration for deep reinforcement learning. *CoRR*, abs/1611.04717, 2016.

[180] Dhruva Tirumala, Hyeonwoo Noh, Alexandre Galashov, Leonard Hasenclever, Arun Ahuja, Greg Wayne, Razvan Pascanu, Yee Whye Teh et Nicolas Manfred Otto Heess : Exploiting hierarchy for learning and transfer in kl-regularized rl. *ArXiv*, abs/1903.07438, 2019.

[181] Emanuel Todorov, Tom Erez et Yuval Tassa : Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[182] Joaquin Vanschoren : Meta-learning: A survey. *ArXiv*, abs/1810.03548, 2018.

[183] Rishi Veerapaneni, John D. Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua B. Tenenbaum et Sergey Levine : Entity abstraction in visual model-based reinforcement learning. *ArXiv*, abs/1910.12827, 2019.

[184] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps et David Silver : Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[185] Jane X. Wang, Zeb Kurth-Nelson, Hubert Soyer, Joel Z. Leibo, Dhruva Tirumala, Rémi Munos, Charles Blundell, Dharshan Kumaran et Matt M. Botvinick : Learning to reinforcement learn. *ArXiv*, abs/1611.05763, 2017.

[186] Tingwu Wang et Jimmy Ba : Exploring model-based planning with policy networks. *CoRR*, abs/1906.08649, 2019.

[187] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker et Martin A. Riedmiller : Embed to control: A locally linear latent dynamics model for control from raw images. *CoRR*, abs/1506.07365, 2015.

[188] Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P. Burgess et Alexander Lerchner : Cobra: Data-e□cient model-based rl through unsupervised object discovery and curiosity-driven exploration. *ArXiv*, abs/1905.09275, 2019.

[189] Nicholas Watters, Andrea Tacchetti, Theophane Weber, Razvan Pascanu, Peter W. Battaglia et Daniel Zoran : Visual interaction networks. *CoRR*, abs/1706.01433, 2017.

[190] Theophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu,

Peter W. Battaglia, David Silver et Daan Wierstra : Imagination-augmented agents for deep reinforcement learning. *CoRR*, abs/1707.06203, 2017.

[191] P. J. Werbos : Learning how the world works: Specifications for predictive networks in robots and brains. *In Proceedings of IEEE International Conference on Systems, Man and Cybernetics, N.Y.*, 1987.

[192] P. J. Werbos : Neural networks for control and system identification. *In Proceedings of IEEE/CDC Tampa, Florida*, 1989.

[193] Chenjun Xiao, Yifan Wu, Chen Ma, Dale Schuurmans et Melanie Mueller : Learning to combat compounding-error in model-based reinforcement learning. *ArXiv*, abs/1912.11206, 2019.

[194] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei et Silvio Savarese : Neural task programming: Learning to generalize across hierarchical tasks. *CoRR*, abs/1710.01813, 2017.

[195] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel et Sergey Levine : One-shot imitation from observing humans via domain-adaptive meta-learning. *ArXiv*, abs/1802.01557, 2018.

[196] Junzhe Zhang et Elias Bareinboim : Transfer learning in multi-armed bandits: A causal approach. *In Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 1340–1346. AAAI Press, 2017.

[197] Shangtong Zhang, Wendelin Boehmer et Shimon Whiteson : Deep residual reinforcement learning. *ArXiv*, abs/1905.01072, 2019.

[198] Tianhao Zhang, Gregory Kahn, Sergey Levine et Pieter Abbeel : Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. *CoRR*, abs/1509.06791, 2015.

[199] Allan Zhou, Eric Jang, Daniel Kappler, Alex Herzog, Mohi Khansari, Paul Wohlhart, Yunfei Bai, Mrinal Kalakrishnan, Sergey Levine et Chelsea Finn : Watch, try, learn: Meta-learning from demonstrations and reward. *ArXiv*, abs/1906.03352, 2019.

[200] Bo Zhou, Hong sheng Zeng, Fan Wang, Yunxiang Li et Hao Tian : E□cient and robust reinforcement learning with uncertainty-based value expansion. *ArXiv*, abs/1912.05328, 2019.

[201] Guangxiang Zhu et Chongjie Zhang : Object-oriented dynamics predictor. *In NeurIPS*, 2018.

[202] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar et Sergey Levine : The ingredients of real world robotic reinforcement learning. *In International Conference on Learning Representations*, 2020.

[203] Shengyu Zhu et Zhitang Chen : Causal discovery with reinforcement learning. *CoRR*, abs/1906.04477, 2019.

Annexe A

Le titre

A.1. Section un de l'Annexe A

...texte...

Annexe B

Les di□érentes parties et leur ordre d'apparition

J'ajoute ici les différentes parties d'un mémoire ou d'une thèse ainsi que leur ordre d'apparition tel que décrit dans le guide de présentation des mémoires et des thèses de la Faculté des études supérieures. Pour plus d'information, consultez le guide sur le site web de la facutlé (www.fes.umontreal.ca).

Ordre des éléments constitutifs du mémoire ou de la thèse		
1.	Les pages de garde	obligatoire*
2.	La page de titre	obligatoire
3.	Le résumé en français et les mots clés français	obligatoires
4.	Le résumé en anglais et les mots clés anglais	obligatoires
5.	Le résumé dans une autre langue que l'anglais ou le français (si le document est écrit dans une autre langue que l'anglais ou le français)	obligatoire
6.	Le résumé de vulgarisation	facultatif
7.	La table des matières, la liste des tableaux, la liste des figures	obligatoires
8.	La liste des sigles, la liste des abréviations	obligatoires
9.	La dédicace	facultative
10.	Les remerciements	facultatifs
11.	L'avant-propos	facultatif
12.	Le corps de l'ouvrage	obligatoire
13.	L'index analytique	facultatif
14.	Les sources documentaires	obligatoires
15.	Les appendices (annexes)	facultatifs
16.	Les documents spéciaux	facultatifs

* Les pages de garde sont obligatoires pour le dépôt initial, qui est normalement fait en format papier. Elles sont cependant à proscrire pour le dépôt final, fait sous forme électronique, dans Papyrus. L'ouverture d'un fichier débutant par une page blanche peut être déconcertante pour le lecteur.