PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

LEONARDO ROSA AMADO

COMBINING LEARNING AND SYMBOLIC PLANNING FOR ROBUST GOAL AND PLAN RECOGNITION

Porto Alegre
2022

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

COMBINING LEARNING AND SYMBOLIC PLANNING FOR ROBUST GOAL AND PLAN RECOGNITION

LEONARDO ROSA AMADO

Doctoral Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science.

Advisor: Prof. Dr. Felipe Meneguzzi

Porto Alegre
2022

Ficha Catalográfica

A481c Amado, Leonardo Rosa

Combining Learning and Symbolic Planning for Robust Goal and Plan Recognition / Leonardo Rosa Amado. – 2021.
102.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Felipe Rech Meneguzzi.

1. Reconhecimento de objetivos. 2. Reconhecimento de planos. 3. Planejamento Clássico. 4. Auto-encoders. 5. Aprendizado de máquina. I. Meneguzzi, Felipe Rech. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Loiva Duarte Novak CRB-10/2079

LEONARDO ROSA AMADO

COMBINING LEARNING AND SYMBOLIC PLANNING FOR ROBUST GOAL AND PLAN RECOGNITION

This Doctoral Thesis has been submitted in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science, of the Computer Science Graduate Program, School of Technology of the Pontifical Catholic University of Rio Grande do Sul

Sanctioned on Mar 31, 2021.

COMMITTEE MEMBERS:

Prof. Dr. André Grahl Pereira (PPGC/UFRGS)

Prof. Dr. Lavindra De Silva (DIAL/Cambridge)

Prof. Dr. Duncan Dubugras Alcoba Ruiz (PPGCC/PUCRS)

Prof. Dr. Felipe Meneguzzi (PPGCC/PUCRS - Advisor)

COMBINING LEARNING AND SYMBOLIC PLANNING FOR ROBUST GOAL AND PLAN RECOGNITION

RESUMO

Abordagens de reconhecimento de planos e objetivos têm relaxado progressivamente os requerimentos sobre a quantidade de conhecimento de domínio e observações necessárias para o funcionamento destas abordagens, criando algoritmos precisos e eficientes. Porém, essas abordagens se baseiam em duas premissas chaves sobre as informações disponíveis para o processo de reconhecimento. Primeiro, é necessário um especialista de domínio capaz de construir conhecimento de domínio de maneira completa e correta para reconhecer o objetivo sendo buscado por um agente. Segundo, mesmo com um domínio correto e completo, a maioria das abordagens de reconhecimento de planos e objetivos são diretamente afetadas pela qualidade das observações analisadas. Enquanto abordagens clássicas de planejamento podem prover soluções para estes problemas, abordagens de aprendizado de máquina são proficientes em lidar com erros e incompletude nos dados fornecidos. Nesta tese, nós introduzimos três abordagens capazes de melhorar o desempenho de técnicas de reconhecimento de planos e objetivos. Primeiro, utilizamos aprendizado não supervisionado profundo para construir conhecimento de domínio a partir de imagens, utilizando o domínio computado para reconhecer objetivos em problemas baseados em imagens. Segundo,desenvolvemos uma abordagem para prever observações faltando em problemas de reconhecimento de objetivos, aumentando a qualidade das observações destes problemas. Terceiro, combinamos técnicas de aprendizado de máquina e planejamento clássico para construir um novo algoritmo para reconhecimento de planos e objetivos. Nesta tese, mostramos a eficácia de cada técnica desenvolvida em um conjunto diverso de domínios de planejamento, incluindo domínios baseados em imagens e domínios clássicos.

Palavras Chave: Reconhecimento de objetivos, Reconhecimento de planos, Planejamento Clássico, Auto-encoders, Aprendizado de máquina.

COMBINING LEARNING AND SYMBOLIC PLANNING FOR ROBUST GOAL AND PLAN RECOGNITION

ABSTRACT

Recent approaches to goal and plan recognition have progressively relaxed the requirements about the amount of domain knowledge and available observations, yielding accurate and efficient algorithms. These approaches, however, make two key assumptions about the information available to the recognizer. First, they assume that there is a domain expert capable of building complete and correct domain knowledge to successfully recognize an agent's goal. Second, even with a complete and correct domain knowledge, most plan recognition approaches are directly affected by the quality of such observations. Such shortcomings can limit the application of such techniques in real-world applications. While symbolic approaches can provide provable solutions to such problems, learning approaches are adept at dealing with incomplete and noisy data. In this thesis, we introduce three approaches that improve the performance of goal and plan recognition by combining learning and symbolic planning techniques. First, we use deep unsupervised learning to generate domain theories from data streams (images) and use the resulting domain theories to deal recognize goals in image-based problems. Second, we develop an approach leveraging attention networks to enhance the observation traces of goal recognition problems by predicting missing observations. Third, we combine learning and symbolic planning techniques to compensate for noise and missing observations into new and efficient goal and plan recognition techniques. We show the effectiveness of each technique in a number of domains, ranging from classical domains from planning competitions to image-based domains.

Keywords: Goal recognition, Plan Recognition, Classical planning, Auto-encoders, Machine learning.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AAAI – Association for the Advancement of Artificial Intelligence

IJCAI – International Joint Conference on Artificial Intelligence

IJCNN – International Joint Conference on Neural Networks

AAMAS – International Conference on Autonomous Agents and Multiagent Systems

ICML – International Conference on Machine Learning

KER – The Knowledge Engineering Review

FLAIRS – Florida Artificial Intelligence Research Society

STRIPS – Stanford Research Institute Problem Solver

PDDL – Planning Domain Definition Language

PRAP – Plan recognition as planning.

ANN – Artificial Neural Network

DNN – Deep Neural Network

AE – Auto-encoder

PCA – Principal Component Analysis

SAE – State Auto-encoder

VAE – Variational Auto-encoder

LSTM – Long Short-Term Memory

RNN – Recurrent Neural Network

NMT – Neural Machine Translation

LATPLAN – Latent Space Planner

POM – Goal recognition approach developed by Pereira, Oren, and Meneguzzi

RG, RG2009, RG2010 – Plan and goal recognition approach developed by Ramírez and Geffner.

PPR – Predictive Plan Recognition

H-PPR – Heuristic Predictive Plan Recognition

LIST OF ALGORITHMS

LIST OF SYMBOLS

CONTENTS

1. INTRODUCTION

Goal recognition and plan recognition refer to the tasks of identifying, respectively, the desired goal an observed agent intends to achieve and the specific plan the agent has committed to executing to achieve said goal, given a sequence of observations as evidence of their behavior in an environment, and a model of the agent behavior which describes how the observed agents may act in such environment to generate the observed evidence [78]. Solving the problem of goal and plan recognition is important for several real-world applications, usually when anticipating an agent's behavior is needed, such as risk management [82], monitoring activities in elder-care [25], traffic monitoring [70], crime and detection prevention, and much more [26, 32, 55, 56]. These problems arise in a multitude of different areas, including natural language processing [28], elder-care [25], multi-agent systems [81], collaborative problem-solving, epistemic problems [79], adversarial planning [69], and more [32, 80].

Approaches to solve goal and plan recognition problems can vary on multiple factors, such as the type of domain model used to describe the observed agent, the mechanism used to compute a plan, and the level of observability and noise in the observations trace used as evidence for recognizing goals and plans. [86, Chapter 1]. There are two main types of domain models for goal and plan recognition, *plan–libraries* and *planning domain theories*.While plan–library approaches to goal and plan recognition have shown to be very fast and accurate in many domains with varying degrees of observability [12, 13, 57, 58], formalizing plan–libraries for each domain can be a laborious and time-consuming task, which requires a substantial amount of domain knowledge. Although the first approaches to plan recognition based on planning theories require a substantial amount of domain knowledge [42], subsequent approaches gradually relax such requirements either by using more expressive planning and plan-library based formalisms [12, 20, 28, 52] as well as allowing for different levels of accuracy and amount of information available in observations required to recognize goals [49, 63, 65, 83].

Regardless of using planning domain theories or plan libraries to recognize the goals and plans of agents acting in an environment, all such approaches assume that a human domain engineer can provide an accurate and complete domain model capable of representing the environment for the recognition algorithm. Such dependence on a human domain engineer severely limits modern plan and goal recognition algorithms' applicability to abstracted domains rather than real-world ones. Recent work on goal and plan recognition uses machine learning to infer the domain knowledge [10, 85, 93], or obviate the need of one [54, 96]. Even given enough domain knowledge to build a domain model, real-world plan recognition problems impose limitations on the quality and quantity of the observations from an agent's interactions in the environment, resulting in observations missing parts of the underlying plan or including spurious observations from silent errors in the sensors [53, 67]. While recent approaches to goal and plan recognition have substantially improved performance under partial observability and noisy conditions [53, 65, 67, 84, 96], dealing with these problems remains a challenge. Machine

learning techniques are adept at dealing with noisy data [44,77], creating robust models capable of accurate predictions with missing or noisy data.

Inspired by these developments, in this thesis, we develop three distinct approaches that leverage machine learning techniques and classical planning approaches to improve the accuracy of goal and plan recognition algorithms across image-based domains and standard classical planning domains. The contributions of this thesis are threefold. First, we develop an approach for recognizing goals in image-based domains. We compute domain knowledge relying only on images and solve goal recognition problems where images represent every component of the goal recognition problem. Second, we introduce a novel method for enhancing the observations in goal recognition problems using a machine learning model to predict missing states between the observations provided in the goal recognition problem. Finally, building from the second contribution, we develop a novel approach for goal and plan recognition that combines planning and machine learning techniques to mitigate low and faulty observability in goal and plan recognition problems, computing complete plans. To achieve such contributions, we have developed three distinct approaches that combine machine learning techniques with classical planning techniques:

Recognizing goals in latent space: Here, we overcome the dependence on human domain engineers for goal recognition by automatically building planning domain knowledge from raw data and using the resulting model in an algorithm capable of recognizing an agent's goal from the same type of raw data. To automatically generate such domain knowledge, we employ unsupervised learning techniques to map from raw data (in this thesis, images) into a latent space representing logical fluents, and, using such fluents, we derive a PDDL [23] action library over which we can reason using planning techniques [10]. Specifically, we extend landmark-based goal recognition techniques [65] to infer goals from the encoded raw data and use the decoder part of an auto-encoder to visualize the plan steps expected of the observed agent. Thus, our main contribution in this approach is a novel goal recognition mechanism that combines deep-learning and heuristic planning techniques to obviate the need for accurate domain engineered planning domains. Our approach allows modern goal recognition algorithms to work directly on real-world data without the need for a domain engineer to bridge real-world data to a symbolic representation.

Enhancing observation for goal recognition problems: Here, we develop a learning model based on LSTMs, leveraging attention mechanisms, to enhance observed traces by predicting missing observations in goal recognition problems. This approach improves the quality of observation traces in goal and plan recognition problems by predicting missing observations so that we can apply off-the-shelf recognition approaches to solve these problems with higher recognition accuracy. We train a model for each of the domains we experiment with and apply them to a dataset of goal recognition problems, using state-of-the-art goal recognition approaches to measure our approach's performance.

Combining Learning and Symbolic Planning for Robust Plan Recognition: Finally, we develop a novel mechanism to explicitly reason about every actual or presumed missing observation by predicting the states induced by a planning model towards each goal hypothesis. This results in two distinct approaches for goal and plan recognition: a *statistical* approach and a *symbolic* approach. Our statistical approach combines machine learning techniques and landmark-based planning heuristics to address the two most common problems in observations: missing observations and noisy observations. We introduce a learning model that allows us to fill in missing observations and rebuild the sequence of states of a complete plan from an initial state to a goal state. We detect faulty (noisy) observations as we rebuild observations and generate state sequences that do not necessarily comply with all the observations if some are not consistent with the planning model. Our symbolic approach relies on planning heuristics to predict missing states, computing plans obviating a learning mechanism.

These contributions can aid the applicability of goal recognition and plan recognition approaches in real-world scenarios using data, improving the performance of such recognition approaches, helping towards solving the plan recognition task. By employing unstructured data, our contributions apply to different applications, bringing symbolic goal and plan recognition closer to the real world. Our thesis shows that machine learning can aid classical planning approaches for plan and goal recognition without relying exclusively on machine learning or classical planning approaches to solve such tasks.

1.1 Publications

During the four-years of our Ph.D. program, we published the following works that are connected to any of the three main contributions of this thesis:

- A full paper in Proceedings of the 31rd International Joint Conference on Neural Networks (IJCNN) titled: "Goal Recognition in Latent Space", 2018 [6];
- A full paper in AAAI workshop on Plan, Activity, and Intent Recognition titled: "An LSTM-Based Approach for Goal Recognition in Latent Space" [8];
- A full paper in ICML/IJCAI/AAMAS 2018 Workshop on Planning and Learning (PAL-18) titled: "LSTM-Based Goal Recognition in Latent Space" [7];
- A student poster int 34th AAAI Conference on Artificial Intelligence (AAAI) titled: "LatRec: Recognizing Goals in Latent Space (Student Poster)", 2020 [5];
- A demo in the AAAI 2020 Workshop on Plan, Activity, and Intent Recognition (PAIR) titled: "LatRec+: Learning-based Goal Recognition in Latent Space (Demo)", 2020 [3] ;

- A demo in Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS) titled: "LatRec: Recognizing Goals in Latent Space (Demo)", 2019 [4];
- A full paper in Proceedings of the 33rd International Joint Conference on Neural Networks (IJCNN), titled: "Goal Recognition in Latent Space", 2020 [6];
- A extended abstract in Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), titled: "Combining LSTMs and Symbolic Approaches for Robust Plan Recognition", 2021.

Moreover, here the work that is not directly connected to our thesis and was published during our Ph.D. program:

- A journal article in The Knowledge Engineering Review, vol 33, e22 (KER) titled: "Q-Table compression for reinforcement learning", 2018 [6];
- A poster in Proceedings of the 30th Florida Artificial Intelligence Research Society Conference (FLAIRS) titled: "Q-Table compression for reinforcement learning" [8], 2017.

1.2 Thesis Outline

This thesis is organized as follows. In Chapter 2, we provide background on automated planning, goal and plan recognition problems, landmarks, planning in latent space, machine learning and long short-term memory, and self-attention neural networks. In Chapter 3 we detail our first contribution, an approach to solve the problem of goal recognition in latent space, where we solve goal recognition problems that use real-world data. We discuss results using off-the-shelf goal recognition techniques and detail a machine learning approach and its shortcomings. In Chapter 4, we develop our second contribution, an approach using machine learning techniques to improve the accuracy of goal recognition approaches by using data. We analyze how this approach improves the accuracy of off-the-shelf goal recognition techniques in standard planning domains and latent space domains across four distinct domains. In Chapter 5, we develop our final approach, which combines machine learning techniques and planning techniques to solve the problem of plan and goal recognition. We compare our approach against state-of-the-art approaches for plan recognition and develop an approach without machine learning to provide an ablation study of the impact of our techniques. In Chapter 6, we discuss related work for all our three approaches, contextualizing where our contributions are situated when compared to the state-of-the-art of plan and goal recognition techniques that leverage machine learning algorithms. Finally, in Chapter 7, we conclude this thesis by discussing our main contributions, open issues, and limitations of our developed approaches, as well as future directions regarding the three proposed approaches in this thesis.

2. BACKGROUND

In this chapter, we introduce the fundamental background to the main topics discussed in this thesis. First, we provide a quick overview of the basics of automated planning. Second, we explain the formal definition of the problem of goal recognition and plan recognition, which is a super-set of the goal recognition problem. Third, we explain the concept of landmarks in automated planning. Fourth, we briefly discuss the area of machine learning and its basic applications. Fifth, we provide a brief overview of neural networks and auto-encoders. Sixth, we detail a more complex type of neural network, the long short-term memory networks, and self-attention mechanisms. Finally, we discuss the current state of the art on planning on latent space, where it is possible to compute plans in image-based problems, by combining auto-encoders and planning techniques.

2.1 Automated Planning

Planning is the task of finding a sequence of actions (*i.e.*, plan) able to transition an agent from a given initial state to a particular goal state [75]. The planning problem can be described as a graph search problem, whose nodes are states, edges are transitions between states that are caused by applying an action at a state, and the solution of planning problem is the sequence of actions (edges) between two nodes, which forms a path to traverse the planning graph. In this thesis, as most of the planning literature, we follow the planning terminology from Ghallab *et al.* [29] to represent states and actions in planning domain problems.

The most fundamental part of a planning task is the planning domain. A planning domain is a formal description of the dynamics of an environment in which an agent acts. Formally, we define the planning domain in Definition 1.

Definition 1 (Planning Domain) *A planning domain Ξ is represented by a pair $\langle \mathcal{F}, \mathcal{A} \rangle$, which specifies the knowledge of the domain, and consists of:*

- *A finite set of facts $\mathcal{F}$, i.e., a set of ground instantiated predicates, defining the environment state properties; and*
- *A finite set of actions $\mathcal{A}$, which is technically a set of ground instantiated operators, representing the actions that can be performed in the environment.*

States $s \subseteq \mathcal{F}$ are composed of facts from a planning domain indicating properties that are true at any moment in time and follow the closed world assumption so that any fact not included in a state is assumed to be false. Conditions or formulas in our formalism comprise positive and negative facts (f, $\neg$ f) representing an implicit conjunctive formula indicating what must be true (alternatively, false) in a state. The positive part of a condition c (pos(c)) comprises

the positive facts in a condition, and the negative part of a condition $\text{neg}(c)$ comprises the negative facts in a condition. We say a state s supports a condition c, $s \models c$ (alternatively, c is valid in s), iff all positive facts are present in s, and all negative facts are absent in s, i.e. $s \models c$ iff $(s \cup \text{pos}(c) = s) \wedge (s \cap \text{neg}(c) = \emptyset)$. An action $\text{a} \in \mathcal{A}$ is represented by a tuple $o = \langle \text{pre(a)}, \text{eff(a)}, \text{cost(a)} \rangle$ containing the preconditions pre(a), the effects eff(a), and a non-negative cost cost(a), which indicating possible transitions between states. The transition of a state s into a new state s' using an action a is represented as $s' = \gamma(s, \text{a})$. The transition is valid iff $s \models \text{pre(a)}$, and $s' = (s \cup \text{pos}(\text{eff(a)})) - \text{neg}(\text{eff(a)})$.

A planning task (or planning instance) is the formal description of a task to be solved in a given planning domain. Thus, a planning task is composed of a planning domain Ξ and planning problem, which describes the finite set of objects of the environment, the initial state from which the planning problem starts, the goal state which an agent desires to achieve.

Definition 2 (Planning Instance) *A planning task is a tuple $\Pi = \langle \Xi, \mathcal{I}, G \rangle$, in which $\mathcal{I}$ is an initial state, G is a goal condition (a conjunctive formula), and Ξ is a planning domain.*

The solution for a planning task is called a plan, which is a sequence of actions that an agent can perform in a planning domain to achieve the goal state G from the initial state $\mathcal{I}$. Formally, we define a plan in Definition 3.

Definition 3 (Plan) *A plan π for a planning task Π is a sequence of actions $\pi = \langle \text{a}_1, \ldots, \text{a}_n \rangle$ that induces a sequence of states $\langle s_0, s_1, \ldots, s_n \rangle$ such that $\mathcal{I} = s_0 \models \text{pre}(\text{a}_1)$, $s_n \models G$ and that every state $s_i \in \pi$ is such that $s_{i-1} \models \text{pre}(\text{a}_i)$ and $s_i = \gamma(s_{i-1}, a_i)$. The cost of a plan is the sum of of the cost all of its actions such that $\text{cost}(\pi) = \sum_{i=1}^{n} \text{cost}(\text{a}_i)$. An optimal plan π^* has the minimum possible cost for achieving a state s_G such that $s_G \models G$ from an initial state I.*

For simplicity, we assume that every action in $\mathcal{A}$ has cost 1, hence, the optimal plan is the plan with the smallest number of actions. Modern planners use the Planning Domain Definition Language (PDDL) (a STRIPS-style [21] domain encoding which is more expressive) as a standardized domain and problem representation medium [23], which encodes the formalism described thus far.

2.2 Goal and Plan Recognition as Planning

Goal recognition is the task of recognizing the goal being pursued by a rational (software or human) agent from observations of its actions in a defined environment.Plan recognition is a related task to goal recognition, but whose object is recognizing the agent's goal being observed and inferring the upcoming actions the agent will take towards such goal [61, 62, 86]. The observations collected from the environment can be either a sequence of actions performed by

the agent or the consequences thereof — such sequences can be either seen in full or a partial subsequence of the agent's actions.

Goal and plan recognition in real-world data assumes an underlying processing step that translates raw sensor data into some symbolic representation [86], as well as a model of the observed agent's behavior generation mechanism. Most goal and plan recognition approaches [12, 27, 58] employ plan libraries to represent agent behavior (*i.e.*, a library that describes all plans for achieving goals).

Recent work uses classical planning domain definitions to represent potential agent behavior bringing goal and plan recognition closer to automated planning [43, 49, 63, 65, 71, 72, 83]. These approaches — which do not use plan libraries — show that automated planning techniques can efficiently recognize goals and plans. Plan libraries may be unavailable in many domains where goal and plan recognition are important (*e.g.*, smart environments, user monitoring, and crime detection), making this second class of approaches critical.

Approaches that use STRIPS-style [21] domain encodings are often known as *plan recognition as planning* (PRAP) because they often use planning domains to generate hypotheses of possible plans consistent with observations [71]. While most approaches for this kind of plan recognition have serious scalability issues, recent work on plan recognition as planning has solved this issue [65] by using landmark-based heuristics [39] to efficiently process observations without the need to call a planner multiple times. Planning landmarks are necessary facts or actions in plans that achieve a particular goal from an initial state and can discriminate observations from plans towards different goals. Recent work develops heuristics and efficient algorithms to use landmark information to rank goal hypotheses in time linear with the number of observations [53, 65].

Most goal and plan recognition approaches use a very specific notion of observation consisting of a sequence of identifiers of the actions executed by an agent, as follows.

Definition 4 (Action observations) *Let $\pi = \langle \mathrm{a}_1, \ldots, \mathrm{a}_n \rangle$ be a plan for a planning task Π. Then a sequence of observations Ω_π is a sequence of actions from π possibly missing actions, but maintaining the same order. Action observations may be noisy if they contain at least one observation that was not included in the plan from which they originate (i.e. if they contain any action $\mathrm{a} \in (\mathcal{A} - \pi)$). We denote the individual observation corresponding to action a_i as $\vec{\mathrm{a}_i}$.*

Example 2.1 *Consider the initial state and possible goal hypothesis illustrated in Figure 2.1. Consider that we can observe an agent trying to achieve the third goal hypothesis, in which the blocks spell the word "ACE". The optimal plan π^* (which follows Definition 3 for this goal is given by the following actions:* $\pi^* =$ [`(unstack D C)`, `(putdown D)`, `(pickup C)`, `(stack C E)`, `(pickup A, B)`, `(stack A C)`]. *An example of action observations for this plan would be* $\Omega_\pi =$ [`(unstack D C)`, `(pickup C)`, `(pickup A, B)`].

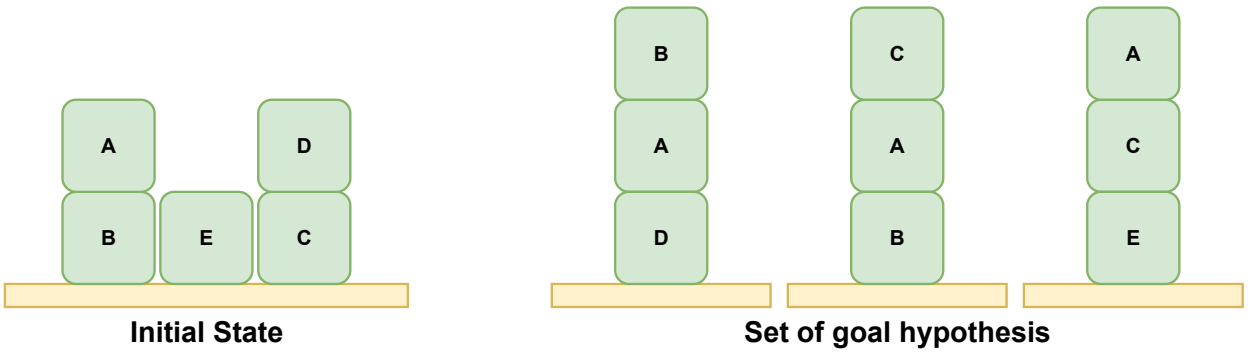


Figure 2.1: Example of a initial state and goal hypotheses for a blocks world recognition problem.

Recent work on goal and plan recognition as planning, such as that of Sohrabi *et al.* [84] also include observations as states, which we formally define as follows.

Definition 5 (State observations) *Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan for a planning task Π, where an observed state is $s_i = \gamma(s_{i-1}, a_i)$, for $0 < i \leq n$, and a sequence of induced states $S_\pi = \langle s_0, \ldots, s_n \rangle$. Then a sequence of state observations Ω_s is a sequence of states from S_π possibly missing states, but maintaining the same order. A missing observation is an entire state missing, not only one or more predicates missing for a given state $s \in \Omega$. Observations may be noisy if they contain at least one observation that was not included in the sequence of induced states from which they originate. We denote the individual observation corresponding to state s_i as $\vec{s_i}$.*

Example 2.2 *Following the Example 2.1, we once again consider an agent that is trying to achieve the third goal hypothesis, in which the blocks spell the word "ACE". The optimal plan π^* for this goal is given by the following actions: $\pi^* =$ [`(unstack D C)`, `(putdown D)`, `(pickup C)`, `(stack C E)`, `(pickup A, B)`, `(stack A C)`]. An example of the induced states S_π and state observations for this plan are illustrated in Figure 2.2. Inside the red box, we have every induced state from following the optimal plan π^*. The red blocks represent when a block is being picked up. In blue, we have an example of a possible set of observations.*

In this thesis, we use Ω to refer to any sequence of observations, such that we can define recognition problems independently of the nature of the observations. Thus, using the planning formalism defined above and the standard definition of plan recognition as planning of Ramirez *et al.* [71, 72], we now formally define a goal and plan recognition problem as follows.

Definition 6 (Plan/Goal recognition problem) *A plan/goal recognition problem Π^Ω_π (alternatively $\Pi^\Omega_\mathcal{G}$) is a tuple $\langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$, where Ξ is a planning domain, $\mathcal{I}$ is an initial state, $\mathcal{G}$ is a set of goal hypotheses, which includes a correct goal G^* (unknown to the observer), and Ω is a sequence of observations (either action observations or state observations).*

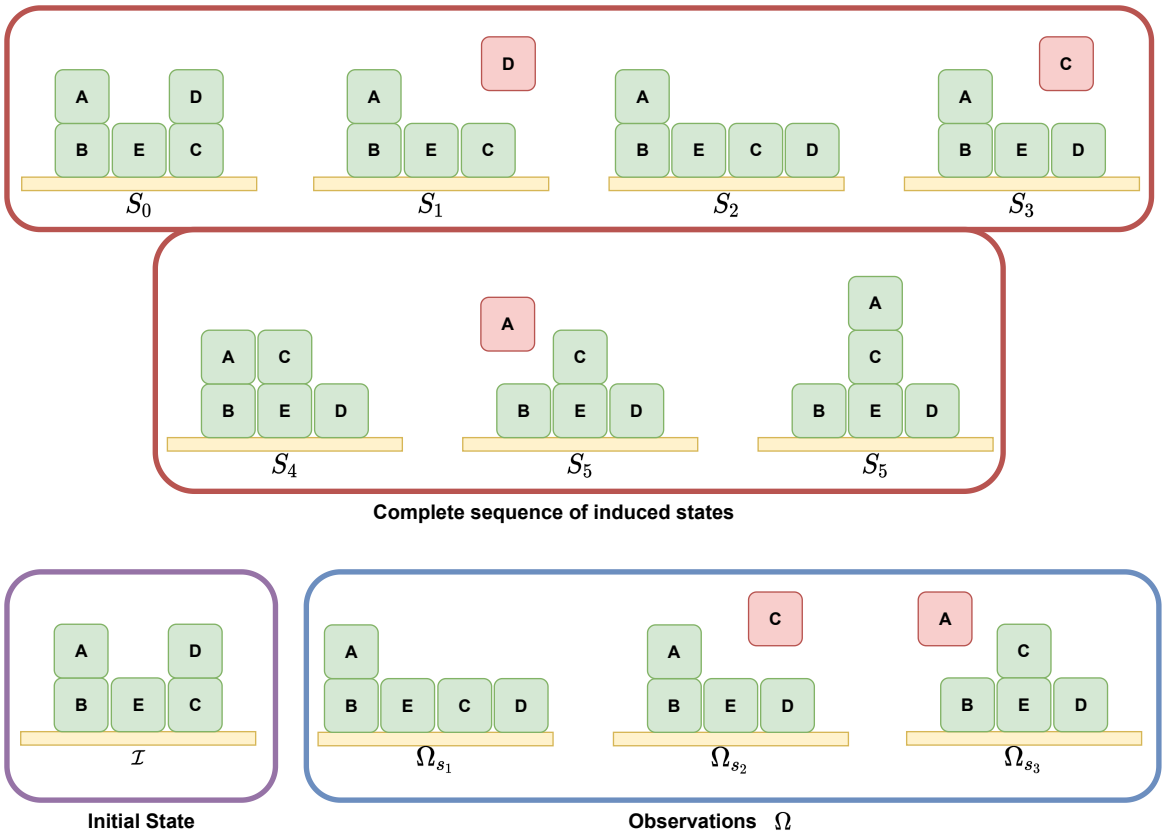


Figure 2.2: Complete sequence of induced states and a set of state observations for Example 2.2.

Example 2.3 *Given a STRIPS style definition of the blocks-world domain, the initial state and the goal hypotheses illustrated in Figure 2.1 and the observations listed in Example 2.1 (or the ones illustrated in Figure 2.2), we have a goal/plan recognition problem.*

Here, we establish the key difference between plan and goal recognition as what we expect the solution to be. Specifically, we define the solutions as follows.

Definition 7 (Plan/Goal recognition solution) *Let $\langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$ be a plan recognition problem Π^{Ω}_{π} (respectively, goal recognition problem $\Pi^{\Omega}_{\mathcal{G}}$) with domain Ξ, initial state $\mathcal{I}$, goal hypotheses $\mathcal{G}$, and observations Ω. The solution π^* for plan recognition problem Π^{Ω}_{π} is a least-cost plan resulting from executing the plan that generated Ω. The solution $\mathrm{s}^*_{\mathrm{G}}$ for goal recognition problem $\Pi^{\Omega}_{\mathcal{G}}$ is the correct goal $\mathrm{s}^*_{\mathrm{G}} \in \mathcal{G}$ resulting from executing the plan that generated Ω.*

Example 2.4 *Given the recognition problem described in Example 2.3 we have a distinct solution for the goal recognition problem and one for plan recognition problem. The solution for the goal recognition problem is selecting the correct goal hypothesis G^* as the most likely goal the agent is pursing, which is the third goal hypothesis of Figure 2.1, which can be described using the following predicates: $G^* =$ [`(on A C)`, `(on C E)`, `(on E table)`]. The solution for the plan recognition problem is the optimal plan which achieves the correct goal hypothesis: $\pi^* =$ [`(unstack D C)`, `(putdown D)`, `(pickup C)`, `(stack C E)`, `(pickup A, B)`, `(stack A C)`].*

Formally, solving the plan recognition problem in Definition 6 solves the goal recognition problem. Hence goal recognition problems are a subset of plan recognition problems.

2.3 Landmarks

In the Automated Planning literature [39, 73, 90], landmarks are necessary properties (or actions) that must be true (or executed) at some point in a valid plan to achieve a particular goal. [39] define fact landmarks as follows:

Definition 8 (Fact Landmarks) *Given a planning task* $\Pi = \langle \Xi, \mathcal{I}, G \rangle$*, a formula* F_i *is a landmark in* Π *iff* F_i *is true at some point of all valid plans that achieve* G *from* $\mathcal{I}$*. Thus, a landmark of a goal* G *is a formula over a set of facts that must be satisfied at some point of all valid plans for* G*.*

Fact landmarks can be either conjunctive or disjunctive [39]. Conjunctive landmarks are a set of facts that the entirety of which must be true simultaneously at some point in all valid plans. Disjunctive landmarks are a set of facts where at least one element of the set must be true in all valid plans. Here, we focus on conjunctive fact landmarks for recognizing goals and plans. In this thesis, we focus only on fact landmarks, using the concept of landmarks to build heuristics for a plan recognition algorithm.

2.4 Machine Learning

Machine learning can be described as the field of study that gives computers the ability to learn without being explicitly programmed [2]. Machine learning studies the construction of algorithms capable of learning and building models from data. Mitchell provides a formal definition of machine learning as follows [59]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." There are three main types of machine learning algorithms [75]:

- Supervised learning.
- Unsupervised learning.
- Reinforcement learning.

In supervised learning, the objective is to infer a function or model using labeled data for training [60]. This labeled data is a set of training examples containing the expected output given a set of features. In Machine Learning, features are dimensions that we use to represent data. For example, the features of a pixel could be the RGB value of this pixel. Using the

training examples, a supervised learning algorithm produces an inferred function. After the training, the learned function should compute the expected output given a set of feature values. For example, suppose we are trying to learn a function capable of predicting how much a house costs, based on the size of the house. To train the algorithm, we receive a few examples of house pricing, containing the price of the house and the size of the house. In this example, the size of the house is our feature value, and the price of the house is the output we want to predict. We train the supervised learning algorithm using the examples of house pricing provided. The algorithm learns a function capable of predicting the house based on its size. This resulting function is called a learned model.

The idea of unsupervised learning is to infer a function to describe a hidden pattern on unlabeled data. Similar to supervised learning, in unsupervised learning, the algorithm is trained using a training data set. However, the data in this data set does not have an expected output. A usual task of unsupervised learning is to cluster the data into different categories. For example, suppose we are given a data set containing multiple instances of a category of flowers. Each instance contains the characteristics of a flower, such as color, size, number of petals. All these flowers are currently set in the same category. However, there are many differences between the flowers in this category, leading to dividing this category into two new sub-categories. Since the instances are not classified using these new sub-categories, we apply an unsupervised learning algorithm to cluster the instances into two sub-categories. After executing the algorithm, the instances are clustered, and we can know which flower belongs to which sub-category.

Machine learning tasks do not differ only based on the input given but also on each task's desired output. Some of the categorizations based on output are [16]:

- Classification task. In classification, the algorithm must determine which class an instance belongs to. The possible classes are already known beforehand.
- Regression task. Similar to classification, but instead of a discrete value, the algorithm tries to find a function that maps an input to a continuous value.
- Clustering task. In clustering, the objective is to find a hidden pattern in a set of inputs, dividing those inputs into multiple classes. Those classes are not known beforehand.
- Density estimation task. The objective is to find the distribution of inputs in some space.
- Dimensionality reduction task. The objective is to simplify inputs to a smaller representation without losing information from the data.

Since we do not use reinforcement learning in this thesis, we focus only on supervised and unsupervised learning techniques. Many current machine learning approaches use a new formalism for both supervised and unsupervised learning, called neural networks. There are many types of neural networks, and they can be tuned to solve every task we listed before.

2.5 Artificial Neural Networks and Unsupervised Autoencoding

Artificial Neural Networks are a group of models loosely inspired by the human brain, designed to recognize patterns. ANNs are composed of multiple nodes organized in layers. A node is an artificial neuron responsible for computing the input. The nodes combine input from the data with a set of coefficients (or weights) that amplifies or suppresses an input. An ANN comprises three types of layers [19]:

1. An **Input layer** responsible for receiving the signal (data) that feeds the neural network.
2. A **Output layer** responsible for receiving the signal from the hidden layer (or input if there is no hidden layer in the network) and producing the network's output. For example, in a classification problem, this layer will output which class the input belongs to.
3. A **Hidden layer** responsible for receiving the signal from the input layer. Every layer between the input and the output layer is considered a hidden layer. The hidden layer is not obligatory. There are only hidden layers in networks with three or more layers.

These layers are interconnected, sending information from one layer to another. The inter-layer connections have weights. These weights affect the values that one layer sends to the other, changing the value's impact on the next layer.

Artificial Neural Networks solve multiple machine learning tasks problems, such as classification, regression, and dimensionality reduction. For example, in a classification task, a neural network will train using provided training data, adjusting the weight of connections to predict the class of an input data correctly. As the weights adjust, the neural networks build a model representing the data and predicting the next inputs. To solve different tasks, ANNs can use supervised, unsupervised, and reinforcement learning algorithms.

A Deep Neural Network (DNN) is an ANN with multiple hidden layers between the input and the output layers. There are several types of DNNs, such as deep belief networks, deep auto-encoders, convolutional neural networks, and deep Boltzmann machines [94]. These types are defined based on the architecture of these networks. Most networks are trained using the back-propagation method.

2.5.1 Back-propagation

Backward propagation of errors (back-propagation) is a standard method for training neural networks [74]. The idea of back-propagation is that by feeding known input values, which we have known the desired output, we can use the error between the network's output and the expected output to modify the network weights. Since back-propagation requires inputs where the expected output is known, it is considered a supervised learning algorithm.

We can only apply back-propagation if the network has its weights initialized. Usually, the weights are initialized at random. After initializing the weights, we can divide back-propagation into four steps [17]:

- Forward propagation.
- Back-propagation of errors.
- Assigning blame to the weights.
- Weight update.

The first step, forward propagation, consists of using an input x_i with expected out y_i and feeding it through the network using the actual weights. In the first iteration of the algorithm, the weights are random initialized values. The network then produces an output based on the input given. We can then use this output in the back-propagation step.

After generating an input in the forward propagation step, we can compare the network's output with the expected output y_i. By comparing the two outputs, we can calculate the error using a loss function. After calculating the error between the output layer and the desired output, the error values are then propagated backward, starting from the output layer. The process continues until each neuron has an error value that approximates its contribution to the original output.

The last two steps are assigning the blame to the weights and weight update. The error derivatives for each weight are calculated by combining the input to each node and the error signal for the node to assign the blame. In the weight update phase, the weights are updated to reduce the error derivative (error assigned to the weight), metered by a learning coefficient. The learning coefficient dictates how fast the nodes learn.

The network is trained using a training set with multiple entries. The procedure described is applied to each one of the entries in the data set. At the end of the training set, the network's weights can better predict new entries of this model.

2.5.2 Auto-encoders

An auto-encoder (AE) [31] is a neural network trained to encode an arbitrary input into an n-dimensional vector representation that can be decoded, reproducing the same input as the output of the entire network. In other words, training an auto-encoder consists of learning from unlabeled data an approximation to the identity function, where the generated output $\hat{\mathcal{X}}$ is similar to the input $\mathcal{X}$ [36]. Internally, an auto-encoder consists of two parts: an encoder function $h = f(x)$ that maps the input through multiple layers to a specific hidden layer h, that encodes a latent representation ($\mathcal{L}$) of the input, and a decoder that produces a reconstruction $r = g(h)$, as illustrated in Figure 2.3. Since auto-encoders are designed to be unable to copy

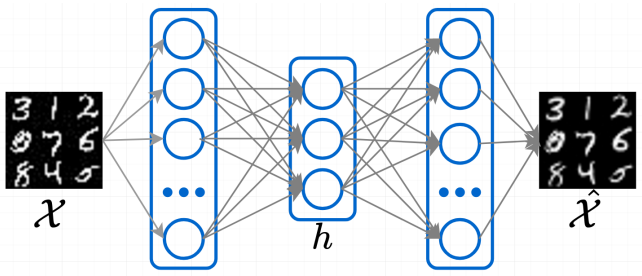


Figure 2.3: Auto-encoder represented as an input $\mathcal{X}$, output $\hat{\mathcal{X}}$ and a hidden layer (latent layer) h.

perfectly the input, they have to learn useful properties that resemble the training data, such that $r_i \approx x_i$. The purpose of an auto-encoder is to constrain the latent layer h to a smaller dimension than the input $\mathcal{X}$, forcing the auto-encoder to learn the most salient features of the training data. The auto-encoder often learns a low-dimensional representation very similar to Principal Component Analysis (PCA). Unlike PCA, auto-encoders that contain nonlinear encoder and decoder functions can learn more powerful nonlinear generalizations [31].

State auto-encoder (SAE) is a special type of auto-encoder, introduced by Asai and Fukunaga [10], that learns a bidirectional mapping between raw data and propositional states. In their work, the encode function maps images to propositional states, i.e., a symbolic representation as latent space vectors, and the decode function maps the propositional states back to images. To create an SAE, the authors use as a base a Variational auto-encoder (VAE) [46], with a Gumbel-Softmax activation [40] used in the latent layer. Gumbel-Softmax (GS) is a recently proposed reparametrization trick for categorical distribution. VAE is an auto-encoder type that imposes additional constraints on the encoded representations (latent layer), forcing it to follow a specific distribution (such as the Gaussian).

Since the distribution has to be differentiable to the application of backpropagation, VAEs use a *reparametrization trick*, which decomposes the target distribution into two distributions, a differentiable distribution and a purely random distribution. In SAE, Gumbel-Softmax performs the *reparametrization trick* for categorical distribution by approximating the Gumbel-Max [48]. It continuously approximates Gumbel-Max, a method for drawing categorical samples. Assume the output z is a one-hot vector, e.g., if the possible classes are (a, b, c), $(0, 1, 0)$ represents "b". The input is a class probability vector, such as $(.1, .1, .8)$. Thus, a one-hot vector z is generated for each class as

$$z_i = \text{Softmax}\left(\frac{g_i + log\ \pi_i}{\tau}\right) \tag{2.1}$$

where g_i is independent and identically distributed samples were drawn from Gumbel(0, 1) [33], π is the class probability vector, and τ is the "temperature" that controls the magnitude of approximation, which is annealed to 0 by a specific schedule. Thus, the Gumbel-Softmax output converges to a discrete one-hot vector when $\tau \approx 0$.

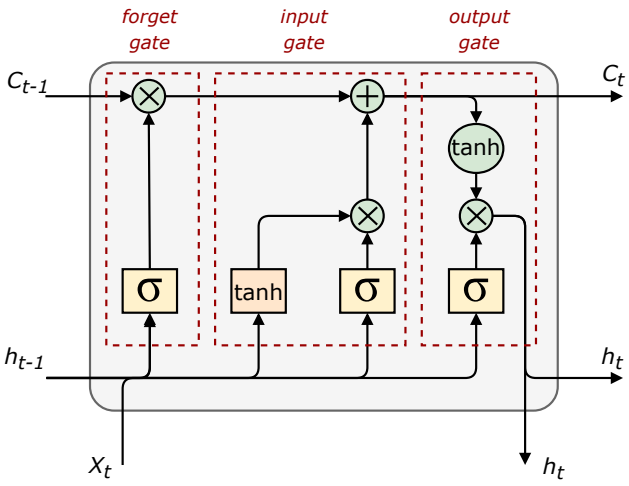


Figure 2.4: Internal structure of the LSTM cell.

2.6 Long Short-Term Memory and Self-Attention Networks

A Recurrent Neural Network (RNN) is a type of network that attempts to model a sequence of dependent events occurring through time, e.g., a financial time series [1], or language modeling [87]. The recurrence is performed by feeding the input layer of the network at time $t+1$ with the network layer's output at time t, keeping a "memory" of the past events. Unfortunately, most RNN architectures then suffer from the well-known vanishing gradient problem [15], i.e., the gradients backpropagated through the network during the training phase tend to decay or grow exponentially. Therefore, as dependencies in RNNs get longer, the gradient calculation becomes unstable, limiting the network to learn long-range dependencies.

To eliminate the vanishing gradient problem, Hochreiter *et al.*proposes an RNN architecture called Long Short-Term Memory (LSTM) network [37] that modifies the original recurrent cell to void vanishing and exploding gradients using the same training algorithm. An LSTM cell contains four key components: the cell state, the forget gate, the input gate, and the output gate. The cell state (C) is responsible for passing the memorized information from one LSTM cell to the next, as weighed by the gates. The forget gate determines what information the network should forget from the previous cell state. It contains a *sigmoid* (σ) layer that outputs a number between 0 and 1, where one (1) means "keep all information" and zero (0) means "forget this information". The input gate computes what information should the network should store in the cell state by applying a *sigmoid* layer to decide what information to keep and a hyperbolic tangent (*tanh*) layer to select new candidates for the cell state, updating the cell state. Finally, the output gate computes what information should be propagated forward by performing a pointwise multiplication of a *sigmoid* layer, which computes what part of the input is forwarded to the cell state filtered by a *tanh* operation. Figure 2.4 illustrates an LSTM cell with its respective gates. Yellow boxes represent layers, elements in green represent pointwise operations ($\otimes$ pointwise multiplication, $\oplus$ pointwise addition, and *tanh* pointwise hyperbolic

tangent function), merging arrows represent the concatenation of elements and forking arrows represent the copy of the content to multiple points.

Despite their advantages, LSTM networks still have limitations due to their fixed-length internal representations. Such limitation creates issues for the processing of longer sequences of data. Attention mechanisms [14] have become an essential component of modern sequence processing models. Attention mechanisms were initially employed in Neural Machine Translation (NMT) tasks, consisting of two RNNs serving as encoder and decoder. A context vector attributes different weights to the whole input sequence elements and mediates information exchange between both networks. Thus, it allows the decoder to focus on the most relevant input elements independently of their position in the sequence. Self-attention [18] is a mechanism that encodes relationships among elements in different positions of the same sequence, allowing the representation of its composition.

Zheng *et al.* [95] develops an alternative attention layer for regular LSTMs, where an attention matrix A captures similarities among input elements. The relationship between elements x_t and $x_{t'}$ of hidden states h_t and $h_{t'}$ at steps t and t' is stored in the similarity element $a_{t,t'} \in A$. We can compute the similarity by applying a *hyperbolic tangent* layer to the weight matrices associated with hidden states h_t and $h_{t'}$ followed by a *sigmoid* layer, both with added bias. The weighted summation of $h_{t'}$ elements and their similarities $a_{t,t'}$ to elements of h_t compose the attention hidden state l_t, which contains information on the relevance of a given element at any step in proportion to other elements in its sequence.

Due to the sequential nature of LSTMs, they are extremely useful for tasks with ordered inputs. Thus, authors have used LSTMs to solve the goal recognition task [54], since the observations are ordered sequences of states.

2.7 Planning in Latent Space

Planning algorithms are based on the *factored* transition function that represents states as discrete facts. This transition function is induced by the actions of a planning domain [1], which is traditionally encoded manually by a domain expert, and virtually all existing plan recognition approaches require varying degrees of domain knowledge in order to recognize observations [65]. Automatically generating such domain knowledge involves at least two processes: converting real-world data into a factored representation (i.e., the predicates in $\mathcal{F}$); and generating a transition function (i.e., the set of actions $\mathcal{A}$) from traces of the factored representation. Although a few approaches have tackled the challenge of applying learning to models of transition functions [41], almost no approaches have addressed the problem of generating domain models from real-world data. A recent approach to planning generates domain models from images of the visualization of the state of simple games and problems, such as the sliding blocks puzzle or towers of Hanoi [10]. This approach uses an auto-encoder [91] neural network to automatically generate two functions with regard to an input image $\mathcal{X}$ and a latent

representation $\mathcal{L}$: an encoder $\phi : \mathcal{X} \mapsto \mathcal{L}$ and a decoder $\psi : \mathcal{L} \mapsto \mathcal{X}$. In this specific case, the input is a d-dimensional image $\mathbb{R}^d$ and the output is an $n \times m$ matrix $\mathbb{R}^{n \times m}$ representing n categorical variables each of which with m categories. When m is two, this auto-encoders output corresponds to binary variables that are interpreted as propositional logic symbols comprising the $\mathcal{F}$ component of a planning domain (without the intermediary step of generating the set of predicates).

The resulting representation in latent space is amenable to automatically inducing a transition function from pairs of states under the assumption that state transitions correspond exactly to pairs of consecutive images in the observed traces. Under this assumption, they generate many propositional actions representing changes between these images as add and delete effects of STRIPS-style actions. The resulting domain representation encodes in latent-space the propositional features from the images. LatPlanα is a heuristic-based forward-search planner [10] that uses this representation to plan solutions for problems derived from images of the initial and target state using the encoded domains. Experimentation with LatPlanα [10] shows that heuristics from the planning literature [24, Chapter 3] are still applicable. However, given the propositional nature of the encoding, they are not as informative. Such lack of informativeness provides a challenge to applying heuristics for goal and plan recognition [63, 65, 66], especially those based on landmarks. To successfully employ efficient goal recognition approaches, we need to learn a consistent latent representation of states and use the propositional transition function induced from state pairs to generate STRIPS-style operators.

In Figure 2.5 we illustrate an example of a planning problem with images that can be solved using LatPlan. The input is an image of the initial state and the goal state. The output is the sequence of states that achieve this goal starting from the initial state.

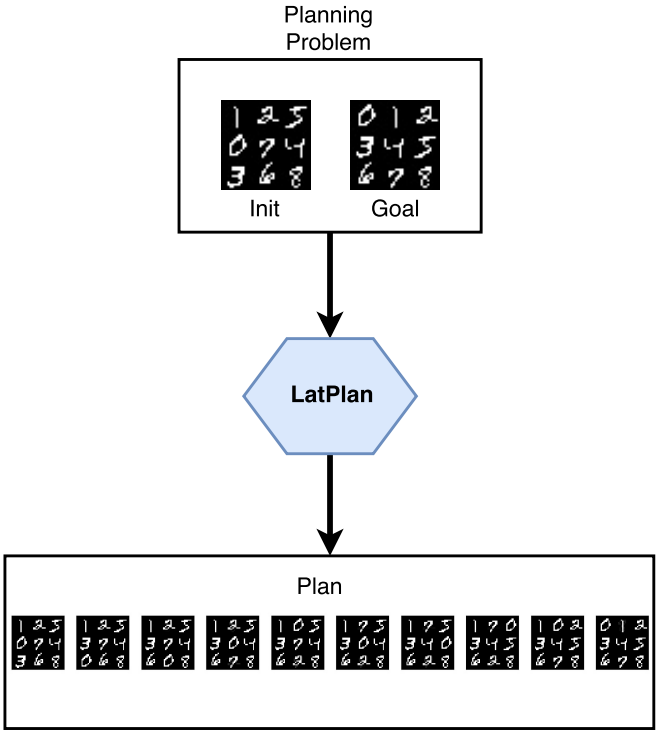


Figure 2.5: Example of a latent space planning problem.

3. GOAL RECOGNITION IN LATENT SPACE

This chapter introduces the first approach of this thesis, an approach to recognizing goals in latent space. As we detailed in Chapter 2, goal and plan recognition refer to the tasks of identifying, respectively, the desired goal towards which an observed agent intends to achieve and the specific plan to which the agent has committed to executing to achieve said goal. While the first approaches to plan recognition based on planning theories requires a substantial amount of domain knowledge [42], subsequent approaches have gradually relaxed such requirements either by using more expressive planning and plan-library based formalisms [12, 20, 28, 52] as well as allowing for different levels of accuracy and amount of information available in observations required to recognize goals [49, 63, 65, 83]. However, regardless of the type of domain model formalism describing the observed agent's behavior, all such approaches assume that a human domain engineer can provide an accurate and complete domain model for the plan recognition algorithm. Such dependence on a human domain engineer severely limits modern plan and goal recognition algorithms' applicability to abstracted domains rather than real-world ones.

We overcome the dependence on human domain engineers for goal recognition by automatically building planning domain knowledge from raw data and using the resulting model in an algorithm capable of recognizing an agent's goal from the same type of raw data. Overcoming this dependence allows us to apply goal recognition techniques in image-based domains, increasing the scope of scenarios we can apply the techniques we developed in this thesis. To automatically generate such domain knowledge, we employ a variational autoencoder (VAE) [45] to map from raw data (in this paper, images) into a latent space representing logical fluents, and using such fluents, we derive a PDDL [23] action library over which we can reason using planning techniques [10]. Specifically, we extend landmark-based goal recognition techniques [65] to infer goals from the encoded raw data and use the decoder part of the variational autoencoder to visualize the plan steps expected of the observed agent. Our main contribution, in this chapter, is a novel goal recognition mechanism that combines deep-learning and heuristic planning techniques to obviate the need for accurate domain engineered planning domains. This mechanism allows modern goal recognition algorithms to work directly on real-world data rather than rely on such data's hand-made processing into a symbolic representation. We evaluate our technique on a dataset consisting of domains from earlier work on planning in latent space [10] as well as images we generate automatically from domains from standard planning benchmarks. Our results show that our domain auto-encoding scheme approximates the encoding of ground versions of hand-coded planning domains and allows recognition accuracy that, in the best case matches and the worst case, is within 33% of hand-coded goal recognition domains.

This chapter is divided as follows. First, we define the problem of goal recognition in latent space, providing an example and a formal definition. Second, we introduce our approach for recognizing goals in such scenarios, explaining how we can compute a domain model using images. Third, we detail our experiments, introducing the dataset we developed and the results

found with many approaches. Finally, we discuss this chapter remarks detailing how we can improve this approach and how this approach builds to the next topic of this thesis.

3.1 Problem Formulation

To solve the goal recognition problems in latent space, first, we define the problem of recognizing goals in image-based domains. Following the Definition 6, which formally defines the problem of goal recognition as the tuple $\Pi^{\Omega}_{\mathcal{G}} = \langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$, the problem of recognizing goals in image domain uses the same tuple, but the initial state $\mathcal{I}$, goal hypotheses $\mathcal{G}$ and Ω are all images. Here, the planning domain Ξ is an inferred domain knowledge through images instead of a planning domain crafted by a domain expert. Hence, to solve the image goal recognition problem defined in this thesis, one must devise a method to infer a planning domain Ξ or to perform goal recognition without the need for domain knowledge. In Definition 9 we formally define the problem of recognizing goals in image-based domains.

Definition 9 (Image-based goal recognition problem) *An image-based goal recognition problem $I_{\Pi^{\Omega}_{\mathcal{G}}}$ is a tuple $\langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$, where Ξ is an inferred domain knowledge from images, $\mathcal{I}$ is an image representation of an initial state, $\mathcal{G}$ is a set of image representations of goal hypotheses, which includes a correct goal $\mathrm{s}^{*}_{\mathrm{G}}$ (unknown to the observer), and Ω is a sequence of image observations (either action observations, given by pair of image transitions, or state observations, given by a single image representing the state).*

Example 3.1 *Figure 3.1 provides an example of an image-based goal recognition problem. In this problem, we want to infer the correct image configuration that the agent is trying to achieve from the set of goal hypotheses (candidate goals) using only observations consisting of intermediate image configurations. Inside the blue box, we have the problem of goal recognition with images. Note that we have three (three) observations in the observation trace, represented by a pair of states. We use the pair of states to graphically represent an action, representing the action responsible for the transition between each pair of states. We have the planning problem in the red box that our goal recognition problem is based on and the correct plan for this planning problem. The planning problem and the correct plan are only being illustrated for clarity, so the reader can better understand the image-based goal recognition problem at hand. This information is not available during the plan recognition problem. As we can see, even for domain experts inferring the correct goal from such domain is not a trivial task when the problem provides only a small number of observations.*

Given the image-based goal recognition problem of Definition 9, it is necessary three mechanisms to solve this problem. First, we need a mechanism to infer domain knowledge through images or a device to obviate the need for domain knowledge to achieve goal recognition. Second, a mechanism to convert an image goal recognition problem into a STRIPS-like

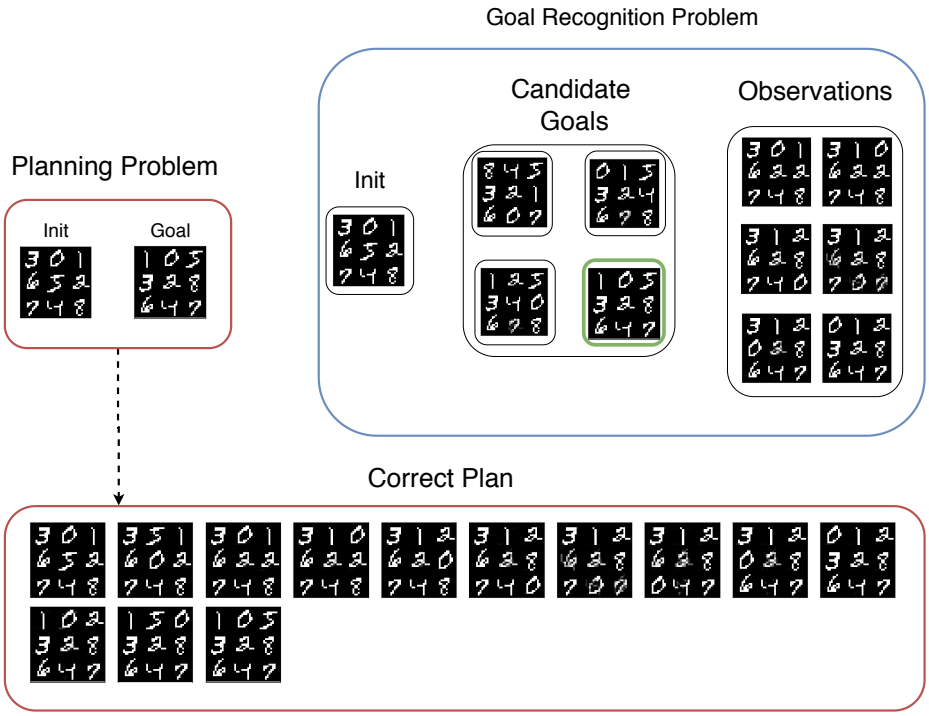


Figure 3.1: Image goal recognition problem.

representation, making it possible to apply state-of-the-art goal recognition techniques. Finally, we need a mechanism to solve the goal recognition problem itself.

3.2 Recognizing goals in image-based domains

To recognize goals in image-based domains, we devise a process with four steps to solve the image-based goal recognition problem (Definition 9). First, we develop a mechanism capable of converting an image to binary representation by training an auto-encoder capable of creating a latent representation of each state image of the domain. Second, we develop an algorithm (Algorithm 1) capable of inferring domain knowledge, deriving a PDDL domain by extracting the transitions from images when encoded in latent space (using our trained auto-encoder), obtaining an inferred planning domain Ξ. Third, we convert to a latent representation the set of images representing the initial state $\mathcal{I}$, the set of observations Ω and a set of possible goals $\mathcal{G}$, where the hidden goal s^*_G is included. Finally, once we have the domain knowledge Ξ and the converted latent representation for each of the elements of the image-based goal recognition problem, we perform goal recognition in the tuple $\langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$, using off-the-shelf state-of-the-art goal recognition techniques [64, 71].

We create the encoded representation of the image states using an autoencoder adapted from Asai and Fukunaga [10], and which has the architecture illustrated in Figure 3.2. The network's input is a 42x42 single channel (black and white) image, which corresponds to the visual representation of problems in our experimental domains. The encoder part consists of

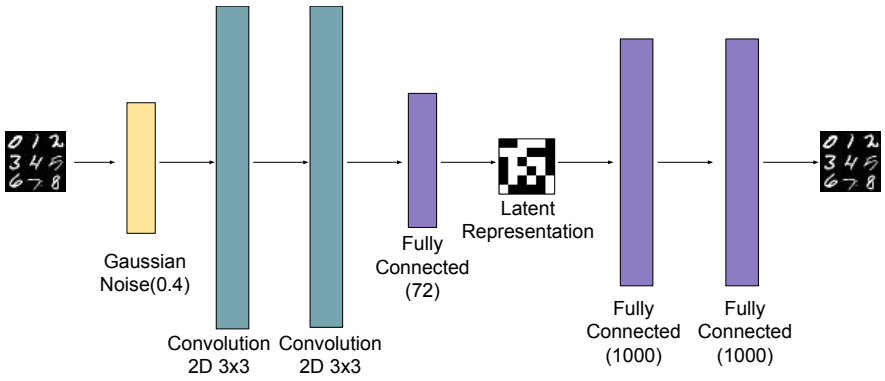


Figure 3.2: Autoencoder architecture.

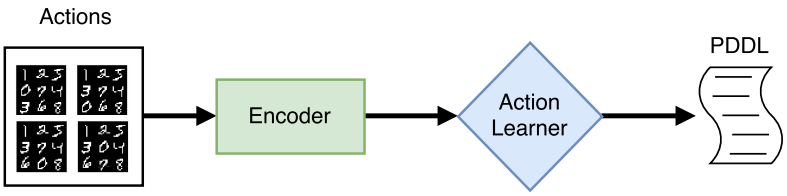


Figure 3.3: PDDL domain generation.

three layers: two 2D convolution layers [47] using a 3x3 filter in both of them, followed by a fully-connected layer with 72 nodes, using rectified linear unit (ReLU) activation [30]. Before the first convolutional layer, we add 0.4 Gaussian noise and a 0.4 dropout rate after the two convolutional layers. The fully-connected layer connects to a 6x6 (36 nodes) latent layer, using Gumbel Softmax [40] activation. This layer generates the latent representation of the image, which we use to infer the planning domain. We use a 36 bits latent layer to represent the entire state space of all problems in our experimental domains. The decoder part of the network consists of two fully connected layers using ReLU activation, connected to the latent layer's output. We also add a 0.4 dropout rate after each fully connected layer of the decoder. Finally, the decoder reconstructs the input image using an output layer of the same size as the input layer.

Our approach requires us to create one distinct auto-encoder for each domain and train them with pre-processed images sampled from the domain. We trained the auto-encoder with 20000 distinct states as images from each domain. We pre-processed the images before training by applying a grayscale filter and then binarizing the resulting images with a threshold of 0.4. Training took 150 epochs, with a 0.1 learning rate and a batch size of 1000 samples.

Having trained an auto-encoder for each domain, we can now infer domain knowledge for each domain. Our goal is to infer domain knowledge in the form of a PDDL domain representation allowing us to leverage state-of-the-art goal recognition techniques. A PDDL domain consists of multiple actions that can be performed in the domain at specific situations encoded in their preconditions and transition the environment by changing the state of binary variables representing such environment. To extract such actions, we create a list containing all

Algorithm 1 Learn actions of a PDDL domain

Require: Set of transitions T

1: **function** ACTION-LEARNER(T)
2: $E \leftarrow \langle\rangle$ ▷ Map of candidate actions
3: $A \leftarrow \langle\rangle$ ▷ Set of generated actions
4: **for all** $(s, s') \in T$ **do**
5: $eff \leftarrow XOR_E(s, s')$
6: $E(eff) \leftarrow E(eff) \cup s$
7: **for all** $eff \in E$ **do**
8: $pre \leftarrow \emptyset$ ▷ Derived pre-condition
9: **for all** $s \in E(eff)$ **do**
10: $pre \leftarrow XNOR_P(pre, s)$
11: $A \leftarrow A \cup \langle pre, eff \rangle$
12: **return** A

possible transitions of a domain in the form of a pair of images. These transitions map which binary variables change as the observed agent executes actions in the domain environment. We encode these transitions using the latent representation containing 72 bits divided into two blocks of 36 bits representing, respectively, the previous state and the state resulting from applying the action. We generate this latent representation of the transitions by seeding two images (the state s before the action and the next state s', after the action) and encoding both using the respective auto-encoder. Using this set of encoded transitions, we derive a set of PDDL actions by performing a bit-wise comparison on both states of a transition to compute the changes between state s and s', using Algorithm 1. We perform a modified XOR operation on both states to compute the effect of each transition and call this operation XOR_E standing for *Effect XOR* in Line 5. The difference of this operation is that the output is 1 for *positive effects* (i.e. when a bit has 0 in s and 1 in s') and -1 for *negative effects* (i.e. when a bit has 1 in s and 0 in s') to distinguish between these two types of effect. XOR_E allows us to group transitions that change the same set of bits into a set of candidate actions, which we further differentiate by inferring their preconditions. To compute the precondition of candidate actions, we use a variation of the XNOR operator, which we call $XNOR_P$ standing for *Precondition XNOR*, in Line 10. The $XNOR_P$ operation aims to distinguish, from the grouped action candidates, which ones share the *same* bit configuration in s, i.e., which bits in *do not change* between the preconditions of a set of candidate actions. The idea is that if a bit has the same value through all states s of every transition in this group of transitions with the same effect, it must be a necessary predicate to execute this action (Lines 9–11). Similar to our XOR_E operation $XNOR_P$ uses 1 to represent *positive preconditions* and -1 to represent negative preconditions. The behavior of both XOR_E and $XNOR_P$ is summarized in Table 3.1. With this process, we compute all the elements of a PDDL action and call this process Action Learner, as illustrated in Figure 3.3. Using the Action Learner, we can output a PDDL domain with a compressed number of actions. In Appendix APPENDIX A, we provide an example (Example APPENDIX A.2) of an action learned using our developed Action Learner.

Table 3.1: Effect XOR and Precondition XNOR operators.

A	B	XOR_E	$XNOR_P$
0	0	0	-1
0	1	1	0
1	0	-1	0
1	1	0	1

Having computed a PDDL domain, we must now set up a goal recognition problem. We represent an image-based goal recognition problem by the tuple $I_{\Pi^{\Omega}_{\mathcal{G}}} = \langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$. We already computed the domain Ξ in Algorithm 1, now we must compute the initial state $\mathcal{I}$, a set of goal hypotheses $\mathcal{G}$, and finally a set of observations Ω. To compute the initial state $\mathcal{I}$ and the set of goal hypotheses $\mathcal{G}$, we use the image representations of these states and convert them to latent representation, using the developed auto-encoder. To derive the observations Ω for action observations (Definition 4), we take pairs of images representing of the environment. These images are encoded to the latent representation. Using the PDDL domain we extracted, we compute which action from the PDDL domain was responsible for such state transition. For state observations (Definition 5), we convert the state's image to its latent representation. After building a goal recognition problem, we can now apply off-the-shelf goal recognition techniques, such as [65, 71, 72, 83]. The output of such techniques is the goal with the highest probability of being the correct one in the latent space representation. We then decode the inferred goal, obtaining its image representation using the decoder. We illustrate this process in Figure 3.4. The inferred PDDL domain and the latent goal recognition problem are used as input to an off-the-shelf goal recognizer. This process can be used for plan recognition problems, only changing the off-the-shelf goal recognizer to a plan recognizer.

3.3 Experiments

This section details the experiments we carried out to measure our approach's effectiveness for goal recognition in latent space. First, we describe the datasets we used to evaluate our approach. Second, we evaluate our approach's performance to generate PDDL domains for each one of the domains. Finally, we compare how our latent representation performs when using various state-of-the-art goal recognition algorithms, comparing the performance against handmade domains.

3.3.1 Datasets

To evaluate our approach to goal recognition, we generated several image-based datasets based on existing goal recognition problems [10, 64]. We have two main experimental objectives: first, we want to compare the performance of goal recognition approaches using domain knowl-

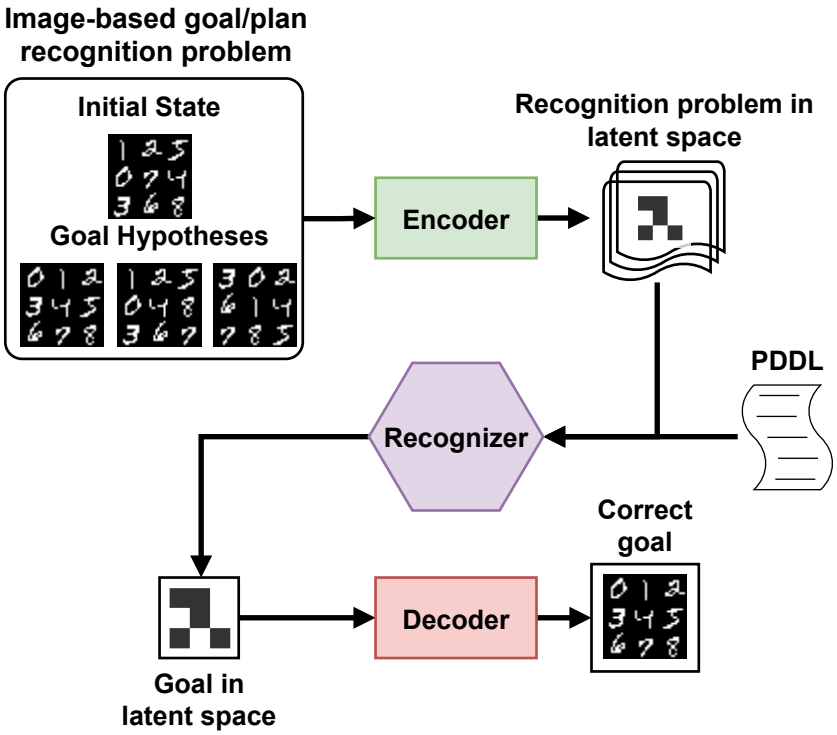


Figure 3.4: Image goal recognition process.

edge built by hand with that of automatically-learned domain knowledge; second, we want to evaluate the performance of various approaches to goal recognition using the automatically-learned domain knowledge. Thus, our evaluation dataset consists of several goal recognition problems generated by taking the generated PDDL domains and an image that serves a the initial state of the goal recognition problem $\mathcal{I}$. Using the initial state, we can set another image as a goal state and compute complete plans with an easy-to-compute image representation. With the complete plan, we can generate problems with varying degrees of observability to better test goal recognition techniques. We can generate identical problems for handmade domains to measure the impact of using learned domains against standard handcrafted models. To generate such traces, we use a standard PDDL planner [35] to search for a plan for a set of randomly generated goals. From the resulting traces, we can generate the observations at various levels of observability by omitting the states resulting from a percentage of the planner's actions.

Using this method to produce experimental datasets, we generated PDDL domains and images for six different datasets:

- three variations of the 8-Puzzle, whose goal to order a set of pieces when the player can only move the blank space:
  - the MNIST 8-puzzle uses the handwritten digits from the MNIST dataset as the pieces of the puzzle, with the number 0 representing the blank space, as illustrated in Figure 3.5a—every image of the dataset uses the same handwritten digit for every repeating number;

  - the Mandrill 8-puzzle uses the image of a Mandrill, shown in Figure 3.5b—we use the mandrill's right eye as the blank space;
  - the Spider 8-puzzle uses the image of a Spider, shown in Figure 3.5c—as the mandrill data set, we use the spider's right eye as the blank space;

- two variations of the Lights-out puzzle game [22], which consists of a four (4) by four (4) grid of lights that can be turned on and off, and which starts with a random number of lights initially on—toggling any of the lights also toggles every adjacent light—the objective is to turn every light off;
  - lights-out digital (LO Digital) is a standard Lights-Out representation using crosses to represent when a light is on, illustrated in Figure 3.5d;
  - lights-out twisted (LO Twisted) is a variation of the digital version of lights out such that the image representation undergoes a distortion filter, twisting the exact position of each light, as seen in Figure 3.5e; and

- the Tower of Hanoi puzzle consists of stacked disks of different sizes and stakes—the objective is to move every disk to a different stack, and we use a version of the puzzle with three disks and four stakes illustrated in Figure 3.5f.

(a) MNIST (b) Mandrill (c) Spider (d) LO Digital (e) LO Twisted (f) Hanoi

Figure 3.5: Sample state for each domain.

3.3.2 Domain Encoding

Table 3.2 shows the performance of our approach in learning the latent representation and inferring PDDL actions for each domain. We measure this performance in terms of the accuracy with which the auto-encoder distinguishes real transitions in the underlying domain (SAE Accuracy %) and the degree of redundancy in the inferred PDDL actions (PDDL Redundancy). SAE (State auto-encoder) accuracy measures the percentage of the number of transitions from the original domain (total transitions) that was captured by the encoded domain (encoded transitions). In contrast, PDDL redundancy measures the ratio between inferred actions (Computed actions) and ground actions in the original domain (Total actions). The closer to the real number of transitions, the better since it means the auto-encoder can completely distinguish all possible transitions of a domain. When a domain includes redundant state representations, some of the latent representation bits tend to become meaningless and

Table 3.2: PDDL generation performance for each domain.

Domain	Total Transitions	Encoded Transitions	SAE Accuracy %	Computed Actions	Ground Actions	PDDL Redundancy
MNIST	967680	963795	99.6%	4946	192	25.76
Mandrill	967680	967680	100.0%	495	192	2.578
Spider	967680	967680	100.0%	763	192	3.974
LO Digital	1048576	1048576	100.0%	5940	1392	4.267
LO Twisted	1048576	1048576	100.0%	12669	1392	9.101
Hanoi	237	237	100.0%	211	38	5.552

thus constitute noise for all algorithms that rely on that representation. Similarly, when we generate redundant transitions and actions, these become noise for the goal recognition algorithms that need to deal with the computed domain model. In Table 3.2, we can see that the state auto-encoder generated a distinct representation for each of the transitions in most of the domains, except for the MNIST 8-puzzle domain, $0,4\%$ of the transitions were duplicates. The MNIST, Mandrill, and spider datasets all represent the same N-Puzzle problem. However, the redundancy in the generated PDDL differs widely. The MNIST generated domain is much more redundant, meaning there are more overlapping actions that could be pruned than the other domains, likely caused by the auto-encoders inability to generate a distinct representation for each transition. Finally, the PDDL actions represent the number of actions in the generated PDDL domains. The computed domains are already grounded and use no variables, eliminating the need for the grounding process (in Example APPENDIX A.4, we provide an example of a planning problem with only grounded predicates).

3.3.3 Goal Recognition

To test the performance of our approach of recognizing goals using only images, we created a dataset of goal recognition problems with the initial state, goal hypotheses, and observations as images. This dataset consists of six (6) distinct problems for each domain, where each problem has at least four distinct candidate goals. From each of these problems (i.e., the initial states and candidate goals), we generate five (5) different conditions for the goal recognition algorithm by altering the level of observability available to the algorithm. We set five different percentages of observability: 100%, 70%, 50%, 30% and 10%. We prune the observations from the dataset described in Section 3.3.1 so that only the specified fraction of the original observations are left. We use three distinct goal recognition approaches to evaluate our approach: the landmark-based heuristics h_{gc} (Goal Completion Heuristic) and h_{uniq} (Uniqueness Heuristic) developed by Pereira, Oren, and Meneguzzi (POM in the table) in [65], and the most accurate approach developed by Ramírez and Geffner in [71] (RG in the table). These three approaches are the current state-of-the-art in goal and plan recognition in terms of time and accuracy, respectively. Here, we analyze goal recognition results, but the RG approach is planning during the recognition process, solving the plan recognition problem.

Table 3.4 summarizes the goal recognition performance using our latent representation and learned PDDL encoding for all domains in the dataset, and three different goal recognition approaches. Each row of this table shows averages for the number of goal hypotheses goals $|\mathcal{G}|$; the percentage of the plan that is observed (&) Obs; the average number of observations per problem $|\Omega|$; and, for each goal recognition approach, the time in seconds to recognize the goal given the observations; the Accuracy % with which the approaches correctly infer the hidden goal; and "Spread $\mathcal{G}$", representing the average number of returned goals. As we can see, the approaches differ widely in accuracy and time elapsed. While the RG approach has better accuracy than the others, it does so with a large spread and long execution times. This trade-off is highlighted in the most complex domains, such as Lights out digital and lights out twisted. For comparison, Table 3.3 shows the results of solving these problems with hand-made PDDL domains. Since there are no learning inaccuracies in such domains, the results are often superior to the learned models. However, in the lights-out model, we can see that the approaches also struggle with a high amount of spread.

Table 3.3: Experimental results on Goal Recognition using handmade domains.

				POM (h_{gc})			POM (h_{uniq})			RG		
Domain	$\|\mathcal{G}\|$	(%) **Obs**	$\|\Omega\|$	**Time(s)** θ (0 / 10)	**Accuracy** % θ (0 / 10)	**Spread** $\mathcal{G}$ θ (0 / 10)	**Time(s)** θ (0 / 10)	**Accuracy** % θ (0 / 10)	**Spread** $\mathcal{G}$ θ (0 / 10)	**Time(s)**	**Acc %**	**Spread** $\mathcal{G}$
Hanoi	4.0	10	1.6	0.010 / 0.012	66.6% / 100.0%	1.6 / 2.3	0.008 / 0.008	66.6% / 66.6%	1.6 / 2.0	0.075	33.3%	1.3
		30	4.0	0.011 / 0.012	66.6% / 100.0%	1.0 / 1.3	0.009 / 0.009	100.0% / 100.0%	1.0 / 1.6	0.080	100.0%	2.3
		50	6.3	0.012 / 0.013	66.6% / 100.0%	1.0 / 1.6	0.009 / 0.010	66.6% / 66.6%	1.0 / 2.0	0.085	100.0%	1.3
		70	8.6	0.013 / 0.013	100.0% / 100.0%	1.3 / 1.3	0.010 / 0.010	66.6% / 66.6%	1.3 / 1.6	0.091	100.0%	1.3
		100	11.6	0.013 / 0.013	100.0% / 100.0%	1.6 / 2.0	0.011 / 0.011	100.0% / 100.0%	1.3 / 1.6	0.098	100.0%	1.3
8-Puzzle	6.0	10	1.0	0.098 / 0.111	16.6% / 33.3%	1.0 / 2.6	0.074 / 0.080	33.3% / 33.3%	2.6 / 2.6	0.179	100.0%	4.8
		30	3.0	0.109 / 0.120	66.6% / 100.0%	1.1 / 2.3	0.079 / 0.085	83.3% / 83.3%	1.0 / 2.5	0.188	100.0%	1.3
		50	4.0	0.117 / 0.129	66.6% / 100.0%	1.0 / 2.0	0.088 / 0.091	100.0% / 100.0%	1.1 / 1.6	0.191	100.0%	1.3
		70	5.3	0.121 / 0.135	100.0% / 100.0%	1.0 / 1.8	0.092 / 0.100	100.0% / 100.0%	1.0 / 1.0	0.210	100.0%	1.0
		100	7.3	0.133 / 0.141	100.0% / 100.0%	1.0 / 1.1	0.108 / 0.110	100.0% / 100.0%	1.0 / 1.0	0.246	83.3%	1.1
Light-Out	6.0	10	1.0	0.689 / 0.766	33.3% / 66.6%	1.3 / 3.8	0.571 / 0.602	33.3% / 66.6%	1.3 / 4.1	5.76	100.0%	5.6
		30	1.6	0.721 / 0.780	50.0% / 83.3%	1.6 / 4.5	0.590 / 0.682	50.0% / 83.3%	1.3 / 5.0	5.79	100.0%	5.3
		50	2.6	0.788 / 0.811	33.3% / 100.0%	2.6 / 5.3	0.622 / 0.704	33.3% / 83.3%	2.6 / 5.3	5.82	100.0%	5.4
		70	3.6	0.804 / 0.849	66.6% / 100.0%	3.8 / 5.0	0.669 / 0.742	66.6% / 83.3%	3.8 / 5.0	5.90	100.0%	5.3
		100	4.3	0.875 / 0.956	100.0% / 100.0%	4.6 / 6.0	0.798 / 0.833	100.0% / 100.0%	4.6 / 5.3	5.93	100.0%	4.8

Table 3.4: Experimental results on Goal Recognition in Latent Space.

				POM (h_{gc})			POM (h_{uniq})			RG		
Domain	$\|\mathcal{G}\|$	(%) **Obs**	$\|\Omega\|$	**Time(s)** θ (0 / 10)	**Acc** % θ (0 / 10)	**Spread** $\mathcal{G}$ θ (0 / 10)	**Time(s)** θ (0 / 10)	**Acc** % θ (0 / 10)	**Spread** $\mathcal{G}$ θ (0 / 10)	**Time(s)**	**Acc %**	**Spread** $\mathcal{G}$
MNIST	6.0	10	1.2	0.591 / 0.603	40.0% / 80.0%	1.6 / 4.0	0.555 / 0.562	40.0% / 60.0%	1.6 / 3.2	21.25	100.0%	6.0
		30	3.0	0.612 / 0.625	40.0% / 80.0%	1.4 / 2.8	0.587 / 0.599	20.0% / 80.0%	1.4 / 3.0	22.26	100.0%	4.8
		50	4.0	0.673 / 0.677	60.0% / 100.0%	2.2 / 3.0	0.609 / 0.628	60.0% / 80.0%	2.2 / 2.8	22.48	100.0%	4.8
		70	5.8	0.698 / 0.703	100.0% / 100.0%	2.4 / 3.0	0.631 / 0.654	60.0% / 100.0%	2.4 / 3.6	23.53	100.0%	3.2
		100	7.8	0.724 / 0.730	100.0% / 100.0%	2.4 / 3.0	0.676 / 0.681	80.0% / 100.0%	2.4 / 3.0	26.34	100.0%	3.4
Mandrill	6.0	10	1.8	0.013 / 0.014	16.6% / 83.3%	1.0 / 3.8	0.011 / 0.012	16.6% / 33.3%	1.1 / 2.8	1.02	83.3%	5.6
		30	4.8	0.015 / 0.017	16.6% / 100.0%	1.0 / 4.8	0.013 / 0.014	20.0% / 83.3%	1.1 / 4.0	1.38	83.3%	3.8
		50	6.0	0.018 / 0.018	33.3% / 83.3%	1.1 / 4.8	0.015 / 0.016	16.6% / 83.3%	1.1 / 4.8	1.44	83.3%	4.1
		70	8.1	0.020 / 0.021	50.0% / 83.3%	1.3 / 4.3	0.016 / 0.018	33.3% / 83.3%	1.3 / 4.0	1.68	66.6%	1.8
		100	11.3	0.022 / 0.023	66.6% / 100.0%	1.8 / 5.16	0.019 / 0.020	33.3% / 100.0%	2.1 / 4.5	1.71	66.6%	1.8
Spider	6.0	10	1.5	0.166 / 0.178	33.3% / 66.6%	2.3 / 4.8	0.151 / 0.154	33.3% / 66.6%	2.3 / 4.5	1.35	83.3%	4.1
		30	4.0	0.181 / 0.190	66.6% / 66.6%	4.1 / 5.1	0.159 / 0.162	66.6% / 66.6%	5.3 / 5.3	1.57	83.3%	3.0
		50	5.6	0.193 / 0.199	50.0% / 83.3%	3.5 / 5.5	0.167 / 0.175	50.0% / 66.6%	4.8 / 4.8	1.66	83.3%	2.8
		70	7.5	0.201 / 0.205	83.3% / 83.3%	4.6 / 5.5	0.016 / 0.018	83.3% / 83.3%	4.6 / 5.3	1.79	66.6%	2.3
		100	10.5	0.208 / 0.217	100.0% / 100.0%	5.5 / 6.0	0.019 / 0.020	100.0% / 100.0%	5.8 / 5.8	2.04	66.6%	1.1
LO Digital	6.0	10	1.0	0.831 / 0.902	33.3% / 33.3%	1.5 / 3.0	0.809 / 0.823	16.6% / 50.0%	1.5 / 3.6	42.52	100.0%	6.0
		30	1.6	0.884 / 1.09	33.3% / 66.6%	1.5 / 4.3	0.835 / 0.840	16.6% / 83.3%	1.5 / 4.5	43.07	100.0%	5.5
		50	2.5	0.915 / 1.13	33.3% / 83.3%	1.5 / 4.5	0.848 / 0.854	16.6% / 83.3%	1.6 / 5.0	43.41	83.3%	5.1
		70	3.6	0.970 / 1.19	83.3% / 100.0%	3.6 / 4.5	0.891 / 0.913	83.3% / 100.0%	3.6 / 4.5	43.78	100.0%	4.8
		100	4.3	1.12 / 1.24	100.0% / 100.0%	2.6 / 4.3	0.913 / 0.938	100.0% / 100.0%	2.6 / 3.3	43.91	100.0%	4.8
LO Twisted	6.0	10	1.0	1.16 / 1.21	16.6% / 16.6%	1.0 / 3.0	1.04 / 1.10	16.6% / 33.3%	1.5 / 4.1	121.97	100.0%	5.8
		30	1.6	1.25 / 1.39	16.6% / 50.0%	1.0 / 3.8	1.11 / 1.18	33.3% / 66.6%	1.3 / 5.1	123.92	100.0%	5.0
		50	2.1	1.33 / 1.46	16.6% / 50.0%	1.0 / 4.5	1.26 / 1.29	16.6% / 66.6%	1.5 / 5.0	124.42	100.0%	5.6
		70	3.3	1.48 / 1.50	16.6% / 83.3%	1.0 / 3.3	1.31 / 1.35	16.6% / 100.0%	1.5 / 5.3	127.22	100.0%	5.5
		100	4.3	1.57 / 1.62	100.0% / 100.0%	2.3 / 5.0	1.40 / 1.44	100.0% / 100.0%	2.3 / 5.5	129.99	100.0%	5.5
Hanoi	4.0	10	1.0	0.304 / 0.318	33.3% / 66.6%	1.0 / 2.3	0.293 / 0.299	33.3% / 100.0%	1.0 / 4.0	6.08	100.0%	4.0
		30	3.0	0.316 / 0.320	100.0% / 100.0%	4.0 / 4.0	0.298 / 0.303	100.0% / 100.0%	4.0 / 4.0	6.21	100.0%	4.0
		50	4.3	0.322 / 0.337	100.0% / 100.0%	4.0 / 4.0	0.306 / 0.311	100.0% / 100.0%	4.0 / 4.0	7.01	66.6%	3.3
		70	6.0	0.345 / 0.354	100.0% / 100.0%	4.0 / 4.0	0.310 / 0.319	100.0% / 100.0%	4.0 / 4.0	7.26	100.0%	4.0
		100	8.3	0.354 / 0.362	100.0% / 100.0%	4.0 / 4.0	0.327 / 0.331	100.0% / 100.0%	4.0 / 4.0	8.19	100.0%	4.0

3.4 Goal Recognition in Latent Space using LSTM

Classical approaches for goal and plan recognition require domain knowledge to perform the recognition process. Instead of inferring domain knowledge, in this section, we introduce an approach to bypass the process of inferring the PDDL domain, as this process requires a large amount of data. We propose using a machine learning model to recognize goals using only plan traces as training data rather than using the training data to generate domain knowledge. Our main goal is to reduce the amount of data necessary to infer an agent's intended goal correctly, without the need for all the data to infer domain knowledge. We propose an approach that performs predictions directly on observation traces of the goal recognition problem. We use plan traces as the training set, which can be previous agent behavior or even a plan library, leveraging a Long short-term memory (LSTM) network to perform recognition. This approach can only solve the goal recognition task, as it is unable to reconstruct the plan executed by the agent.

3.5 Data-driven goal recognition

Long short-term networks [37] are capable of solving classification problems by receiving streams of data and returning a class based on the entirety of the data received. These data streams can be used to model an agent's actions under observation in goal recognition problems, where the class to be recognized by the LSTM network is the agent's goal. Thus, we develop an LSTM to solve the problem of image-based goal recognition. Our LSTM consists of three main layers. First, we use an embedding layer to convert our input sequence into a dense representation with a dimension of 1000 that will feed the LSTM units. Second, we use an LSTM layer containing 512 units. Finally, a fully connected layer receives the output from LSTM and generates the goal representation with 36 output neurons. We use sigmoid activation on the neurons from the output layer and a binary cross-entropy loss using RMSprop as the optimizer. Figure 3.6 illustrates our LSTM architecture.

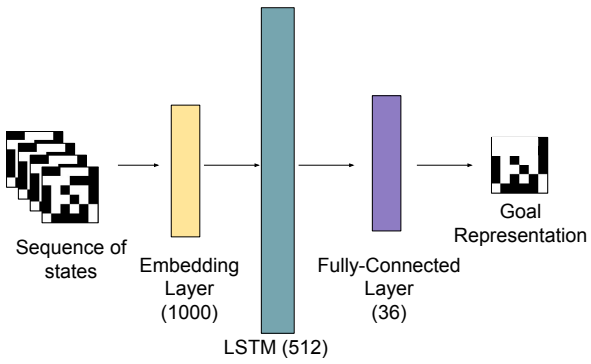


Figure 3.6: LSTM Architecture

To create a model to recognize goals, we train an LSTM that receives a sequence of encoded states and predicts an encoded goal. To perform a fair comparison to the state-of-the-art, we use as input encoded states generated by the encoder module from the autoencoder created by [10]. Thus, we convert each image-state into a latent representation (a 6x6 binary matrix). Figure 3.7 illustrates the process of training and testing our LSTM model. We highlight three main steps of this process. First, given a set of image-states representing a sequence of states and a specific plan's goal, we use the encoder (same as used in Section 3.2) to generate the latent representation of each image. Second, using the representations, we train the LSTM to predict the goal given the states. The output is a representation of this goal. Finally, we use the decoder to convert the produced representation into an image.

To train the LSTM network, we require data extracted from plans for each domain. We use plan traces generated by our previous approach, observing the states reached in each plan. Each trace generated a list of states, and then we included each trace's goal as a class to the LSTM. To improve accuracy in low observability scenarios, we included partially observable traces (which means some states were removed from the plan trace), including 10%, 30%, 50%, 70% of observability. We use early stopping to avoid overfitting during the training phase and set a limit of 10,000 epochs. Early stopping monitors validation loss ensuring training will stop when loss stops decreasing.

We manipulate LSTM inputs by converting the latent representations into a specific encoding. In our specific case, we turn each state into an integer number. Thus, we differentiate them simplifying the input. An entry example for this model is *22, 23, 33, 48, 12*, where each number is a specific state from the state-space in its domain, and the sequence is an entire

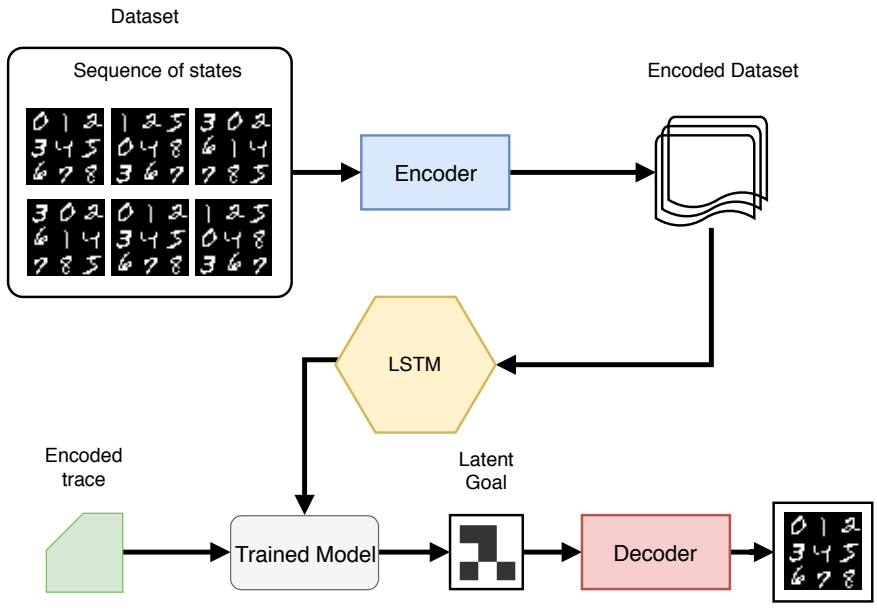


Figure 3.7: Goal recognition using LSTMs

plan. The output layer is 36 binary neurons, which we use to rebuild the latent representation by reshaping it into a 6x6 matrix.

3.5.1 LSTM Goal Recognition experiments

We use the same dataset used in our previous approach to compare our approach with the existing approaches of goal recognition for latent space. This dataset consists of 6 distinct problems for each domain, where each problem has at least four (4) distinct candidate goals. The candidate goals are not necessary for the LSTM based approach, but they are used for the sake of comparison. From each of these problems (*i.e.*, the initial states, and candidate goals), we generate five (5) different conditions for the goal recognition algorithm by altering the level of observability available to the algorithm. We set five different percentages of observability: 100%, 70%, 50%, 30% and 10%.

The observations from the dataset described in Section 3.3.1 are pruned so that only the specified fraction of the original observations are left. We use two goal recognition approaches to compare with our LSTM approach (LSTM in Table 3.5): the landmark-based heuristics h_{gc} (Goal Completion Heuristic) developed by Pereira, Oren, and Meneguzzi (POM in in Table 3.5) in [65], and the most accurate approach developed by [71] (RG in in Table 3.5). These two approaches are the current state-of-the-art in goal and plan recognition in terms of time and accuracy, respectively.

Table 3.4 summarizes each approach's goal recognition performance using the latent representation and learned PDDL encoding provided in [6], for all domains in the dataset and three different goal recognition approaches. In the LSTM approach, the learned PDDL is not needed to perform goal recognition, only the encoded traces. In this comparison, we included every hidden goal in the LSTM training set at least once. We guarantee that the traces used in this comparison were not included in the training set. Each row of this table shows averages for the number of candidate goals $|\mathcal{G}|$; the percentage of the plan that is actually observed (%) Obs; the average number of observations per problem $|O|$; and, for each goal recognition approach, the time in seconds to recognize the goal given the observations; the Accuracy % with which the approaches correctly infer the hidden goal; and Spread $\mathcal{G}$, representing the average number of returned goals. The LSTM model spread is always one, as it always returns one goal. As we can see, the LSTM achieved good overall accuracy across all domains and observability scenarios. The execution time was between 0.3 and 0.5 seconds. While the RG approach has good accuracy, it does so with a large spread and long execution times. This trade-off is highlighted in the most complex domains, such as Lights out digital and lights out twisted. The POM approach struggled with the high spread in some domains, such as the Hanoi domain, but was faster than RG in all scenarios. Overall the LSTM achieved better results, considering it always returns one goal and the other approaches struggled with the high spread. As we can see, for recognizing goals that are contained in the training set, the LSTM

is a promising approach that does well in both speed and accuracy. For comparison, Table 3.3 shows the results of solving these problems with hand-made PDDL domains.

Table 3.6 shows the results when dealing with goals that are not contained in the training set. The test set consists of 6 distinct problems with distinct goals, where each problem generates five (5) traces using different observability (10, 30, 50, 70, 100%). The LSTM was unable to recognize any of the goals that are not contained in the dataset. We present the reconstruction accuracy that estimates how close was the LSTM to reconstruct the correct goal. As we can see, our approach needs the goal to be contained in the training set, as the LSTM network is unable to reconstruct a goal that it has not seen. In such scenarios enumerating every possible goal is not recommended, as the number of possible states (and so goals) in an 8-Puzzle problem is 362,880. Thus our approach by encoding classes for classification is auspicious, as long as the training set contains many goals (and thus classes), as it removes the burden of enumerating every class.

Table 3.5: Experimental results on GR in Latent Space with LSTMs.

				POM (h_{qc})			LSTM			RG		
Domain	$\|\mathcal{G}\|$	(%) **Obs**	$\|O\|$	**Time(s))** θ (0 / 10)	**Accuracy** % θ (0 / 10)	**Spread** $\mathcal{G}$ θ (0 / 10)	**Time(s)**	**Acc** %	**Spread** $\mathcal{G}$	**Time(s)**	**Acc** %	**Spread** $\mathcal{G}$
MNIST	6.0	10	1.2	0.591 / 0.603	33.3% / 83.3%	1.6 / 4.0	0.346	16.6%	1.0	21.25	100.0%	6.0
		30	3.0	0.612 / 0.625	33.3% / 83.3%	1.4 / 2.8	0.335	100.9%	1.0	22.26	100.0%	4.8
		50	4.0	0.673 / 0.677	60.0% / 100.0%	2.2 / 3.0	0.326	100.0%	1.0	22.48	100.0%	4.8
		70	5.8	0.698 / 0.703	100.0% / 100.0%	2.4 / 3.0	0.394	100.0%	1.0	23.53	100.0%	3.2
		100	7.8	0.724 / 0.730	100.0% / 100.0%	2.4 / 3.0	0.357	100.0%	1.0	26.34	100.0%	3.4
Mandrill	6.0	10	1.8	0.013 / 0.014	16.6% / 83.3%	1.0 / 3.8	0.335	50%	1.0	1.02	83.3%	5.6
		30	4.8	0.015 / 0.017	16.6% / 100.0%	1.0 / 4.8	0.366	100.0%	1.0	1.38	83.3%	3.8
		50	6.0	0.018 / 0.018	33.3% / 83.3%	1.1 / 4.8	0.389	100.0%	1.0	1.44	83.3%	4.1
		70	8.1	0.020 / 0.021	50.0% / 83.3%	1.3 / 4.3	0.353	100.0%	1.0	1.68	66.6%	1.8
		100	11.3	0.022 / 0.023	66.6% / 100.0%	1.8 / 5.16	0.347	100.0%	1.0	1.71	66.6%	1.8
Spider	6.0	10	1.5	0.166 / 0.178	33.3% / 66.6%	2.3 / 4.8	0.375	83.3%	1.0	1.35	83.3%	4.1
		30	4.0	0.181 / 0.190	66.6% / 66.6%	4.1 / 5.1	0.423	83.3%	1.0	1.57	83.3%	3.0
		50	5.6	0.193 / 0.199	50.0% / 83.3%	3.5 / 5.5	0.431	100.0%	1.0	1.66	83.3%	2.8
		70	7.5	0.201 / 0.205	83.3% / 83.3%	4.6 / 5.5	0.384	100.0%	1.0	1.79	66.6%	2.3
		100	10.5	0.208 / 0.217	100.0% / 100.0%	5.5 / 6.0	0.368	100.0%	1.0	2.04	66.6%	1.1
LO Digital	6.0	10	1.0	0.831 / 0.902	33.3% / 33.3%	1.5 / 3.0	0.315	83.3%	1.0	42.52	100.0%	6.0
		30	1.6	0.884 / 1.09	33.3% / 66.6%	1.5 / 4.3	0.317	100.0%	1.0	43.07	100.0%	5.5
		50	2.5	0.915 / 1.13	33.3% / 83.3%	1.5 / 4.5	0.336	100.0%	1.0	43.41	83.3%	5.1
		70	3.6	0.970 / 1.19	83.3% / 100.0%	3.6 / 4.5	0.371	83.3%	1.0	43.78	100.0%	4.8
		100	4.3	1.12 / 1.24	100.0% / 100.0%	2.6 / 4.3	0.330	83.3%	1.0	43.91	100.0%	4.8
LO Twisted	6.0	10	1.0	1.16 / 1.21	16.6% / 16.6%	1.0 / 3.0	0.376	66.6%	1.0	121.97	100.0%	5.8
		30	1.6	1.25 / 1.39	16.6% / 50.0%	1.0 / 3.8	0.320	100.0%	1.0	123.92	100.0%	5.0
		50	2.1	1.33 / 1.46	16.6% / 50.0%	1.0 / 4.5	0.339	100.0%	1.0	124.42	100.0%	5.6
		70	3.3	1.48 / 1.50	16.6% / 83.3%	1.0 / 3.3	0.312	100.0%	1.0	127.22	100.0%	5.5
		100	4.3	1.57 / 1.62	100.0% / 100.0%	2.3 / 5.0	0.327	100.0%	1.0	129.99	100.0%	5.5
Hanoi	4.0	10	1.0	0.304 / 0.318	33.3% / 66.6%	1.0 / 2.3	0.334	66.6%	1.0	6.08	100.0%	4.0
		30	3.0	0.316 / 0.320	100.0% / 100.0%	4.0 / 4.0	0.365	100.0%	1.0	6.21	100.0%	4.0
		50	4.3	0.322 / 0.337	100.0% / 100.0%	4.0 / 4.0	0.371	100.0%	1.0	7.01	66.6%	3.3
		70	6.0	0.345 / 0.354	100.0% / 100.0%	4.0 / 4.0	0.372	66.6%	1.0	7.26	100.0%	4.0
		100	8.3	0.354 / 0.362	100.0% / 100.0%	4.0 / 4.0	0.329	66.6%	1.0	8.19	100.0%	4.0

Table 3.6: LSTM results with unknown goals.

Domain	Reconstruction Accuracy (%)	# Problems	Correct Predictions
MNIST	48,6%	30	0%
Mandrill	53,6%	30	0%
Spider	58,4%	30	0%
LO-Digital	53%	30	0%
LO-Twisted	51,2%	30	0%

3.6 Chapter remarks

In this chapter, we formalized the problem of image-based goal recognition and developed two approaches to solve this problem. Our first approach relies on inferring domain knowledge, obviating the need for human engineering to create a domain model for goal recognition, and then applying state-of-the-art goal recognition techniques. We compared three state-of-the-art approaches of goal recognition to evaluate the domain we derived from image evidence. Empirical evaluation on multiple datasets shows that while we can solve some problems with the same or higher accuracy than hand-coded problems, many others come to within 33% of the accuracy of recognizing such problems. Regardless, our approach allows breakthroughs in goal/plan recognition, allowing the usage of recognition techniques in image-based domains without the need for a domain expert. This approach has two main limitations. First, we need all possible transitions of the domain to infer its encoding. Without every single transition of the domain, we can not ensure that the actions generalize enough to every possible state. We plan on solving this using degrees of uncertainty in each predicate when generating the domain.

Our second approach for image-based goal recognition uses an LSTM network, obviating the need for inferring the domain model. Empirical evaluation on multiple datasets shows that while we can solve problems with the same or higher accuracy than hand-coded problems, our LSTM approach does not easily generalize for goals outside the training dataset. Nevertheless, our approach provides a meaningful initial step towards goal recognition without human domain engineering and minimal training data. In summary, the advantages of using our LSTM approach to recognize goals are high accuracy and fast prediction time when dealing with known goals; no false-positive predictions, given that it only predicts a single goal; no need for a PDDL domain, which requires extensive domain knowledge.

However, both of our approaches have limitations. First, like most pure machine learning approaches, their performance is tied to the training dataset's robustness. Second, it requires training, either for inferring the domain model or for the process of goal recognition itself, which is unnecessary for classical goal recognition approaches. Third, it has very limited generalizability with small datasets. We can only guarantee complete PDDL domains if we have all states of the domain and an auto-encoder capable of generating a distinct latent representation for each state of the domain model. Finally, our approach that infers the domain model has its performance tied to off-the-shelf goal recognition approaches, which, as we show in Table 3.4, can perform poorly in the latent space domains. Moreover, for future work we must find common ground to compare with other approaches that learn domain knowledge in the literature [9, 93].

Inspired by these developments, in the next chapter, we develop an approach that uses data-driven techniques to improve the accuracy of state-of-the-art goal recognition approaches instead of directly solving the problem of goal and plan recognition itself. The idea is to leverage

the strengths of both the data-driven approach developed in this chapter and state-of-the-art goal recognition techniques.

4. ENHANCING OBSERVATIONS IN GOAL RECOGNITION PROBLEMS

The accuracy of most recognition approaches is directly related to the amount of observations (e.g., sequences of states) available in order to recognize correctly the intended goal [49, 63, 65, 83]. Such limitation appears in most goal recognition techniques, including the state-of-the-art in goal recognition as planning [67], yielding poor results in some domain models when dealing with fewer observations.

In Chapter 3, we developed an approach to recognize goals in image-based domains. Our results show that standard goal recognition approaches struggle with high-spread in such domains. Moreover, data-driven approaches have their own set of limitations, such as the inability to recognize goals not included in the training dataset. Inspired by the developments of Chapter 3, we introduce a novel method for enhancing the observations in goal and plan recognition problems that improves the quality of the observation trace used in goal and plan recognition problems. The idea is that by using the precision of learning techniques to improve the observability trace instead of actively performing goal recognition and leaving the goal recognition process to standard goal recognition approaches [49, 67, 83], we can leverage the strengths of both machine learning techniques and classical planning approaches. To do so, we use a learning model using Recurrent Neural Networks (RNNs) and attention mechanisms to predict missing states in between the observations provided of a goal/plan recognition problem. Once We train four models, one to each of the domains we experiment with and apply them to a dataset of goal recognition problems. We evaluate our method in two types of goal recognition problems: (1) classical goal recognition problems, where domains are formalized in Planning Domain Definition Language (PDDL) [51], a well-known domain language used in AI Planning; (2) goal recognition problems in latent space, where the domains are learned using convolutional neural networks and autoencoders [11], which we computed in Chapter 3. For the classical goal recognition problems, we train two domain-specific models based on two Artificial Intelligence Planning System (AIPS) domains, i.e., blocks-world and logistics, two well-known classical domain models in the planning literature. For the goal recognition problems in latent space, we train two domain-specific models for two domains: the MNIST 8-puzzle, where the puzzle tiles are images extracted from the MNIST dataset, and the Lights-Out (LO Digital) game. Experiments and evaluation show that our novel method can improve the accuracy of state-of-the-art techniques in goal recognition as planning [67] by approximately 60%.

This chapter is divided as follows. First, we introduce the formal definition for the problem of predicting missing observations in plan and goal recognition problems. Second, we detail how our approach can enhance observations in plan and goal recognition problems using a self-attention LSTM. Third, we present the experiments we carried to test our approach, detailing datasets, how we trained the neural networks, and our results. Finally, we summarize this chapter, discussing our approach's limitations and how we can improve and build into it.

4.1 Problem Formulation

The problem of predicting missing observations in goal and plan recognition problems (respectively $\Pi^{\Omega}_{\mathcal{G}}$ and Π^{Ω}_{π}) consists of predicting n missing observations from an observation trace Ω. In this thesis, we consider the observations of this problem to be state observations Ω_s (Definition 5), thus, a complete observation trace is all states induced by a plan π, denoted by S_{π}. The complete solution to this problem is computing all induced states S_{π}. However, this can be impractical as to compute every state achievable by a plan solves both the problem of goal and plan recognition. Here, we want to predict as many missing observations as possible, but without the commitment of achieving a full solution. In Definition 10, we formally define the problem of predicting missing observations.

Definition 10 (Missing observations problem) *A missing observations problem $\mathcal{P}_{\Pi^{\Omega}_{\mathcal{G}}}$ is a tuple $\langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$ (same as the goal and plan recognition problems), where Ξ is a domain knowledge (either inferred or written by a domain expert), $\mathcal{I}$ is an initial state, $\mathcal{G}$ is a set of goal hypotheses, which includes a correct goal $\mathrm{s}^{*}_{\mathrm{G}}$ (unknown to the observer), and Ω is a sequence of state observations with n missing observations, derived from all states S_{π} produced by a plan pi (which is unknown to the observer).*

What differentiates the problem of missing observations from the goal and plan recognition problems is the solution, which we define in Definition 11.

Definition 11 (Missing observations solution) *The solution to a missing observation problem $\mathcal{P}_{\Pi^{\Omega}_{\mathcal{G}}} = \langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$, is given by the predicting the n observations missing in Ω. A complete solution consists of finding every single missing observation in o, computing every induced state S_{π} by the plan π (which was the plan used to produce the set of observations Ω), achieving the correct goal $\mathrm{s}^{*}_{\mathrm{G}}$. A partial solution consists of computing m observations, where $m < n$, without the need to achieve the correct goal $\mathrm{s}^{*}_{\mathrm{G}}$.*

Example 4.1 *Figure 4.1 illustrates a problem of missing observation and a partial and complete solution for this specific problem. We illustrate the problem with an initial state and, inside the purple box, the ordered state observations Ω. In the partial solution (yellow box), we have three predicted states, represented formally by $\mathcal{P}_{s_n}$ and illustrated by the red squares. The complete solution is the sequence of states induced by the plan π the observed agent follows. Thus, in the complete solution, we have predicted all missing states and can achieve the correct goal hypothesis the agent was pursuing (illustrated by the purple square).*

Given the problem in Definition 10 and the solution described in Definition 11, we must formalize an approach to predict missing states. Our goal in this chapter is to create an approach capable of achieving a partial solution for the missing observation problem without using planning algorithms, which can be costly.

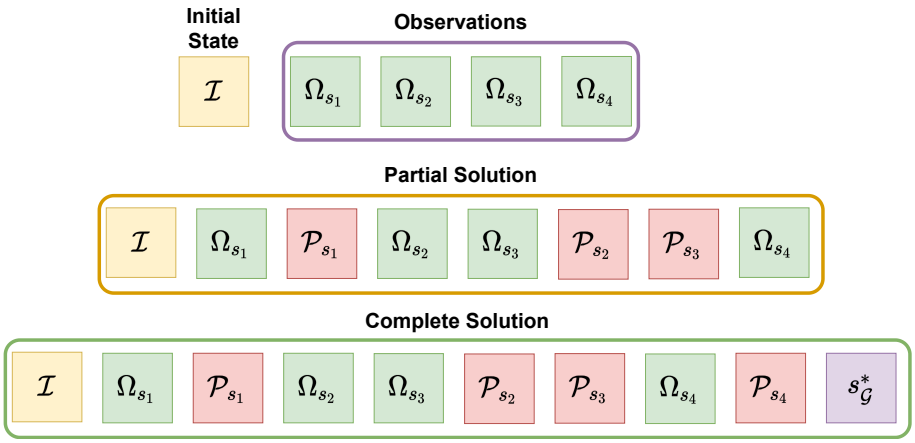


Figure 4.1: Example of a missing observation problem and its solutions.

4.2 Predicting missing observations

State-of-the-art goal recognition techniques compare their approaches with varying degrees of observability to correctly measure these techniques' efficiency in different situations. As the degree of observability decreases, which means fewer observations are given to the recognizer, it becomes harder to recognize goals, so the accuracy decreases and the spread increases. To improve goal recognition approaches, we propose a learning model capable of predicting missing observations, enhancing the information used in the goal recognition process. The model takes in incomplete observation traces as input and predicts the most likely next step. Observation traces are sequences of states given to the trained model. Thus the model returns a prediction of the next state.

A key assumption of this approach is that we either know the domain model or have access to a learned approximation via a black box transition function, such as the one developed in Chapter 3. Thus, we are given a full goal recognition problem with the tuple $\Pi^{\Omega}_{\mathcal{G}} = \langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$. Using the domain model, we check if a sequence of states in the set of observations Ω is impossible by evaluating whether a transition between two states is valid, starting from the initial state I and the first observation. If the sequence of states is impossible, we conclude that an observation is missing at the point of the invalid transition of the observation trace. Hence, we aim to predict n missing observations at the point of the invalid transition until we can achieve a state with a valid transition to the next observation. We continue iterating through every observation in the observation trace Ω, trying to fill every gap. Figure 4.2 illustrates this process, where P_{S_n} is a possible observation that can fit the set of observations Ω.

To achieve a partial solution for the problem of missing observations (Definition 10), we develop an architecture to enhance observation trace in goal and plan recognition problems. We show the developed architecture for enhancing observations in Figure 4.3. The *Grounder* element in the diagram generates the set of all possible actions for a given domain instance.

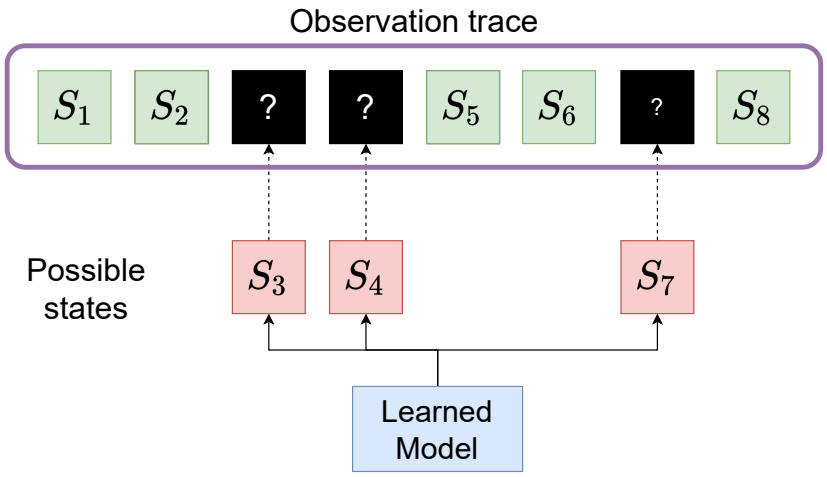


Figure 4.2: Predicting missing observations.

Any modern planner can do the grounding process. In this work, we use Pyperplan[1] for the grounding process. The *Observation Predictor* element takes as input the traces of a goal or plan recognition problem and uses the ground actions to verify whether transitions between observations in the observation trace Ω are valid. If a transition is not valid, it feeds the sequence of states until the invalid transition to the *Learned Model* element, a trained self-attention LSTM, which outputs a prediction for the missing observation. We select the most likely state for each prediction following the given input observations and test if it is a possible consecutive state. Then, the predicted observation is fed back to the *Observation Predictor*, and if it constitutes a valid transition, we append it to the sequence of observations. If the predicted state is impossible to apply in the given observations, we iterate the top-10 outputs of the network layer and evaluate whether one is a valid transition. If none of the top-10 selected predictions are valid transitions, we do not add any observations to avoid adding an observation that could lower the recognizer's performance. We then move on to the next observation o_{n+1} and check if it is a valid transition with the observation that comes next o_{n+2}. We repeat this process for every single observation in the observation trace Ω. If the predicted state is included in the set of goal hypotheses $\mathcal{G}$, we stop predicting observations further from that state. Finally, when the model can no longer predict valid transitions to the current sequence of observations, the enhanced observation traces are sent to the *Off-the-shelf Goal Recognizer* element.

To predict the missing observations, we developed a machine learning model that leverages attention mechanisms to predict missing observations given a sequence of states. Our model architecture consists of 5 layers. The first layer is an embedding in which we tokenize the input. The second is an LSTM with 1024 hidden neurons connected to a self-attention layer with softmax activation. A *flatten* layer takes this output and feeds into a fully connected layer activated with softmax. The final activated output is a vector with the size of unique tokens (vocabulary), where each token represents an observation. Figure 4.4 illustrates the model architecture, where *max-len* is the maximum length of an input sequence of observations

[1] https://github.com/aibasel/pyperplan

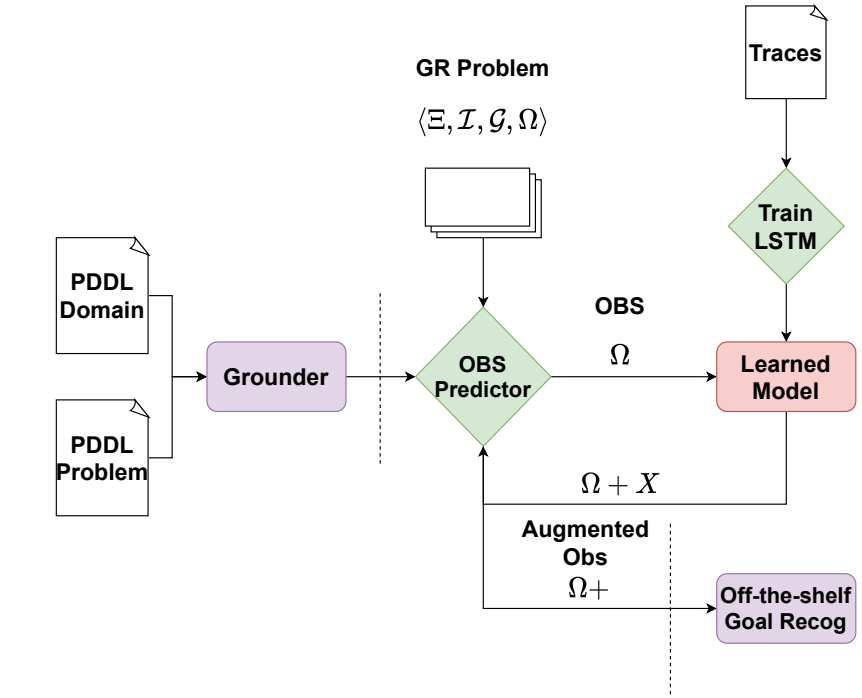


Figure 4.3: Enhancing observations architecture.

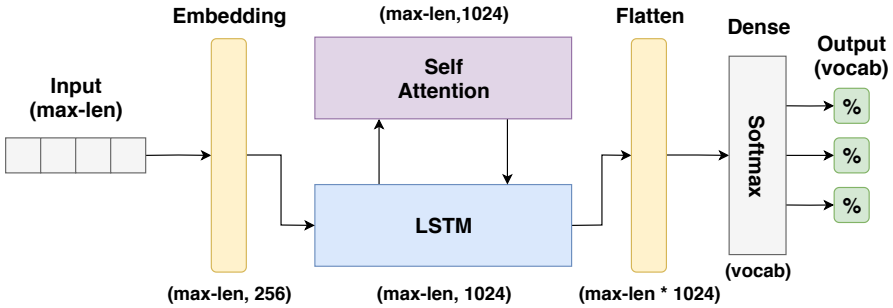


Figure 4.4: Learning model architecture.

in the dataset and *vocab* is the number of unique tokens of the problem (all possible states that the model can predict). We train the network with an Adam optimizer and a Categorical Cross-Entropy loss function.

4.3 Datasets and training

In this section, we detail how we built the datasets for training and testing the models. First, we detail the datasets used to train models for the classical AIPS (Artificial Intelligence Planning and Scheduling Systems) competition domains. Then, we detail the datasets created to train the model for domains in latent space. Finally, we detail how we train our models and test their performance.

4.3.1 Classical domains dataset

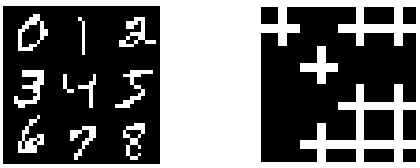
To train our models for classical problems, we use two classical PDDL domains from AIPS, i.e., blocks-world and logistics. These are well-known planning domains used in classical goal recognition approaches in the literature. We build one dataset for each domain and model, where we use 100 planning problems for each domain. We then solve these problems using a standard PDDL planner called PyperPlan[2], which computes an optimal plan to solve each problem instance. This plan is a sequence of states, the solution from the initial state to the goal state. After computing a plan for each problem, we separate the data into a training set containing 80 plan instances and a test set with 20 plan instances. We augment the data used in training by generating subsets of the training instances as new data instances. For example, if we have the train instance $x_1 = [s1, s2, s3, s4, s5]$, where s_n is a state, and $y_1 = [s6]$ is the label to this training instance, we create new training instances, such as $x_{1a} = [s1, s2, s3, s4]$ using $y_{1a} = [s5]$ as label. The subsets maintain the continuity of states and are generated until the minimum length of three states. After augmentation, plan instances have, on average, a sequence of 7 states on these domains.

4.3.2 Latent space datasets

To test our approach in real-world datasets, we generated several image-based datasets based on existing goal recognition problems we developed in Chapter 3. These datasets were first introduced in [11], using images to compose puzzle domains and employing a variational autoencoder to extract latent space features from each domain. We select two domains from the datasets we introduced in n Chapter 3, the MNIST 8-puzzle and the Lights-Out domain. The MNIST 8-puzzle, illustrated in Figure 4.5a, uses handwritten digits from the MNIST dataset as tiles of the puzzle, with the number 0 representing the blank space. Every image of the dataset uses the same handwritten digit for every repeating number. The Lights-Out puzzle game [22] consists of a four (4) by four (4) grid of lights that the player can turn on and off, thus named Lights-Out Digital (LO Digital). LO Digital starts with a random number of lights initially on—toggling any of the lights also toggles every adjacent light—and the objective is to turn every light off. The LO Digital domain uses crosses to represent when a light is on, as illustrated in Figure 4.5b.

To generate the training dataset, we create 100 planning problems for each latent space domain. After creating the problems, we use LatPlan [10] to solve them and create a plan trace (sequence of states). Latent space states are represented as binary vectors extracted from the autoencoder latent features. We then split these plan instances, using 80 of them for training and 20 for testing. We augment the training dataset for the latent space domains

[2] https://bitbucket.org/malte/pyperplan



(a) MNIST (b) LO Digital

Figure 4.5: Sample state for each domain.

following the same procedure used for the classical domains to increase the dataset size. After augmentation, plan instances have, on average, a sequence of 5 states on these domains.

4.3.3 Training and Testing

After building the training and test datasets, we train four distinct models, one for each domain, using the architecture described in Figure 4.3. For each sequence of states (trace) in the training set, we extract and use the final state as a label for the remainder of the sequence. Since LSTM models expect inputs with equal size, we identify the longest sequence and apply a zero left-padding to all instances shorter than it to have the same length. Our model receives a sequence of states as input and outputs a prediction during training, which is the probability of each state in the vocabulary being the correct next state. Training is interrupted when there is no improvement in validation loss after ten (10) epochs straight (early stop).

Table 4.1 shows the results of our training model in each dataset, where *Vocab. Size* represents the number of distinct states the model can predict, *Max. Length* the maximum sequence length that can be fed to the network, *Top-1 Acc* the standard accuracy of the model when predicting the correct missing observation, and *Top-3 Acc* the accuracy when considering the three (3) higher probability states.

4.4 Experiments

To run experiments in goal recognition problems and perform the prediction of missing observations using the trained model, we use the 20 planning problems of each domain's test dataset to generate observation traces. Goal recognition techniques use different rations of observability to test their efficiency. We remove observations from the test plans following ratios of 30, 50, and 70 percent of observability. To achieve these degrees of observability, we randomly cut a percentage of observations out of a plan, resulting in traces as illustrated in Figure 4.2. We refer to the resulting dataset with no predictions from our model as the *Standard* dataset. We generate a new dataset by feeding the observations to our model and filling gaps of missing observations, resulting in the dataset we refer to as *Enhanced* dataset.

Domain	$\lvert \mathcal{V}_{\mathcal{F}} \rvert$	Max. Length	Top-1 Acc	Top-3 Acc	Training Set
BLOCKS	473	16	0.78	0.84	3208
LOGISTICS	1021	22	0.85	0.86	5074
MNIST	566	10	0.83	0.91	1013
LO-DIGITAL	709	8	0.85	0.88	1656

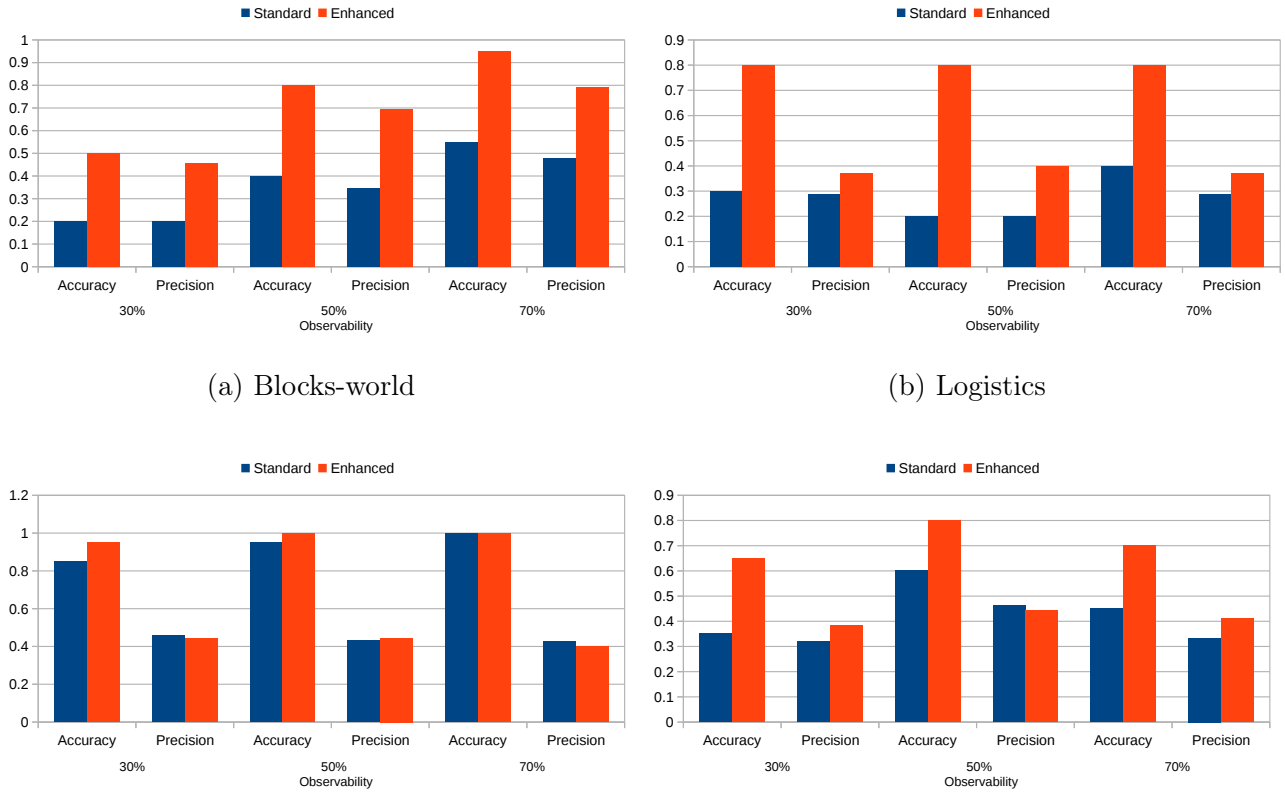
Table 4.1: Model metrics for each domain.

Finally, we generate goal recognition problems using each domain's traces in the datasets for each observability ratio, resulting in 60 goal recognition problems for each domain.

Both the *Standard* dataset (with no predictions from our approach) and the *Enhanced* dataset are used as input to a state-of-the-art goal recognizer proposed in [65]. The recognizer uses a landmark-based approach [65] (POM in Chapter 3) for recognizing actual goals from a set of goal hypotheses, given a trace of observations. The recognizer uses the Uniqueness Landmark-based Heuristic, which uses the concept of a landmark's uniqueness value, representing the landmark's information value for a particular candidate goal compared to landmarks for all goal hypotheses. This heuristic estimates the ratio between the sum of the achieved landmarks' uniqueness value and the sum of the uniqueness value of all landmarks of a goal hypothesis. The more landmarks an observation trace contains concerning a goal hypothesis, the higher the score for that goal being the correct goal pursued by the agent. Thus, enhancing observations traces that are correct concerning the plan an agent is pursuing will most likely increase the chances of recognizing the agent's actual goal.

In Figure 4.6, we show the result metrics of the goal recognizer after applying our technique in the four (4) proposed domains. We show the goal recognizer's accuracy and precision metrics to each observability ratio, in blue for the *Standard* dataset and red for the *Enhanced* dataset. Results show that our approach can improve the accuracy of the state-of-the-art goal recognition algorithm, both in classical and latent space domains. In the classical domains (Blocks-world Fig. 4.6a, and Logistics 4.6b), the accuracy gain is over 60% across all degrees of observability. In the Logistics domain, with 50% of observability, the accuracy triples, highlighting our method's ability to improve observation traces. In latent space domains, the observation traces in the LO Digital domain improve the performance of the recognizer significantly (Fig. 4.6d), while the results for the MNIST 8-puzzle show a slight increase in accuracy (Fig. 4.6c), and a slight decrease in precision for 30 and 50 percent of observability. This decrease occurs because the accuracy in the MNIST 8-puzzle domain is already high, leaving small room for improvement and a higher chance of noise being added.

In Table 4.2 we report in detail the result metrics of the *Standard* and *Enhanced* datasets using the state-of-the-art goal recognizer. As the data shows, our approach consistently improves the results across all metrics. In *Avg. # Obs*, we show the average number of observation traces given to the recognizer, and in *Recog. Goals*, the average number of identi-



(a) Blocks-world

(b) Logistics

(c) MNIST 8-Puzzle

(d) LightsOut Digital

Figure 4.6: Goal recognition accuracy in each domain.

fied goals returned by the goal recognizer (spread). Our approach can consistently add many observations across all domains, and in some cases, triples the number of observations. The improvements are significantly more prominent in the classical goal recognition problems, but our technique can still improve the results using latent space domains.

4.5 Chapter Remarks

In this chapter, we developed an approach for predicting missing observation in goal and plan recognition problems and tested it in four distinct domains, including domains based on real-world data. Results show that our approach can enhance traces with missing observations for all domains, significantly improving accuracy across varying degrees of observability. As shown in the results, in some cases, our approach more than doubles the accuracy of the recognizer, with minimal increase of spread as a drawback. Thus, the main contribution detailed in this chapter is a novel technique to aid state-of-the-art goal recognition techniques, in classical and real-world data domains, without performing goal recognition in itself. Our approach can be used in future state-of-the-art goal recognition approaches.

Table 4.2: Results for Goal Recognition problems

Domain	Observability	Dataset	Accuracy	Precision	Recall	F1-score	Avg. # Obs.	Recog. Goals
Blocks World	30%	Standard	0.20	0.20	0.20	0.20	4.15	1.00
		Enhanced	0.50	0.45	0.50	0.48	13.60	1.10
	50%	Standard	0.40	0.35	0.40	0.37	6.85	1.15
		Enhanced	0.80	0.70	0.80	0.74	16.80	1.15
	70%	Standard	0.55	0.48	0.55	0.51	9.85	1.15
		Enhanced	0.95	0.79	0.95	0.86	16.00	1.20
Logistics	30%	Standard	0.30	0.29	0.30	0.29	5.50	1.05
		Enhanced	0.80	0.37	0.80	0.51	19.30	2.15
	50%	Standard	0.20	0.20	0.20	0.20	8.65	1.00
		Enhanced	0.80	0.40	0.80	0.53	19.15	2.00
	70%	Standard	0.40	0.29	0.40	0.33	12.40	1.40
		Enhanced	0.80	0.37	0.80	0.51	19.85	2.15
MNIST 8-Puzzle	30%	Standard	0.85	0.46	0.85	0.60	2.75	1.85
		Enhanced	0.95	0.44	0.95	0.60	8.10	2.15
	50%	Standard	0.95	0.43	0.95	0.59	3.95	2.20
		Enhanced	1.00	0.44	1.00	0.62	8.80	2.25
	70%	Standard	1.00	0.43	1.00	0.60	5.50	2.35
		Enhanced	1.00	0.40	1.00	0.57	9.85	2.50
LO Digital	30%	Standard	0.35	0.32	0.35	0.33	2.70	1.10
		Enhanced	0.65	0.38	0.65	0.48	4.35	1.70
	50%	Standard	0.60	0.46	0.60	0.52	3.70	1.30
		Enhanced	0.80	0.44	0.80	0.57	5.25	1.80
	70%	Standard	0.45	0.33	0.45	0.38	5.00	1.35
		Enhanced	0.70	0.41	0.70	0.52	6.65	1.70

Throughout the development, we trained with many different architectures for predicting the missing observations, such as using the full state instead of embeddings in the input layer and the output as the full reconstruction of the state using the sigmoid function as the activation for the output layer. The results were significantly inferior to our current model. Indeed, we could improve our current model and developed a much more complex architecture. However, we do wish to tinker with machine learning models until we find the perfect model for the problem of missing observations. We aimed to develop a simple enough model that could be applied to distinct domains and improve the performance of standard goal-recognition and plan recognition approaches without the need to fine-tune the model for each domain extensively.

The approach we developed in this chapter was only able to solve the problem of missing observations partially. It is a simple approach, which relies on a state-of-the-art recognizer to perform the recognition process. In the next chapter, we use the machine learning model built in this approach to develop a novel approach capable of solving the plan and goal recognition problems, thus, achieving a complete solution for the problem of missing observation.

5. COMBINING LEARNING AND SYMBOLIC PLANNING FOR ROBUST PLAN RECOGNITION

Real-world plan recognition problems impose limitations on the quality and quantity of the observations from an agent's interactions in the environment, resulting in observations missing parts of the underlying plan or including spurious observations from silent errors in the sensors [67]. While recent approaches to goal and plan recognition have substantially improved performance under partial observability and noisy conditions [65, 67, 84, 96], dealing with these problems remains a challenge for the community.

In Chapter 4, we developed an approach to improve the performance of goal and plan recognition approaches by enhancing the observations in the observation trace. Our approach for enhancing observation can improve the recognition accuracy of state-of-the-art recognition approaches but cannot solve the goal and plan recognition problem and deal with noisy observations. Most current approaches for goal and plan recognition deal with partial observations and noise implicitly either by computing entire plans under the assumption of (near) optimality [72], by imposing costs on ignoring observations [84], or by focusing on necessary conditions [67].

To deal with low observability and noisy observations, we develop a mechanism to explicitly reason about every actual or presumed missing observation by predicting the states induced by a planning model towards each goal hypothesis, building up on top of our approach developed in Chapter 4. This results in two distinct approaches for goal and plan recognition: a *statistical* approach and a *symbolic* approach. Our statistical approach combines machine learning techniques and landmark-based planning heuristics to address the two most common problems in observations. We use the learning model discussed in Chapter 4, which allows us to fill in missing observations and rebuild the sequence of states of a complete plan from an initial state to a goal state. Then, we rely on landmark heuristics to improve the ability to predict the correct missing observations. We detect faulty (noisy) observations as we rebuild observations and generate state sequences that do not necessarily comply with all the observations if some are not consistent with the planning model. Our symbolic approach relies on planning heuristics to predict missing states, computing plans obviating a learning mechanism.

We evaluate our approaches in standard, handcrafted planning domains and automatically generated domains in latent space, using the same problems developed in Chapter 4, showing its effectiveness at recognizing both plans and goals. We compare our work to other approaches [71, 72, 89] in the literature, measuring the optimality of the computed plans and the accuracy of the predicted goals in scenarios with missing and faulty observations. In latent space domains, standard goal and plan recognition approaches struggle to achieve high precision, as their spread (i.e., returned goals) in such domains is very high. We achieve high precision in most domains, excelling in latent space domains with a precision increase of up to 60%, including in problems with noisy and spurious observations. Finally, we show that our

approaches can compute complete optimal plans in most problems, resulting in a reliable way to perform plan recognition.

This chapter is divided as follows. First, we introduce our approach to solve goal and plan recognition problems, called Predictive Plan Recognition (PPR). We detail two possible mechanisms to predict missing states and how our approach can work with any of them. Second, we show our experimentation setup and how we measured the efficacy of both our approaches in multiple domains. Finally, we conclude the chapter with remarks regarding the achieved results and how we can further improve our work.

5.1 Predictive Plan Recognition (PPR)

In this section, we develop novel approaches to solve plan recognition problems, dealing with noisy and missing observations, and all components necessary to build these approaches. First, we describe in detail our approach for solving goal and plan recognition problems given a function capable of predicting the next states as a black box. Second, we discuss how we can predict these states using machine learning techniques and landmarks, filling the black box. Finally, we introduce a method to predict missing states without the need for machine learning techniques, which we can use to compare with our learning-based approach.

5.1.1 Recognizing plans with PPR

We solve the plan recognition problem of Definition 6 by computing a sequence of intermediary states achieved by a plan π given a plan recognition problem $\Pi^{\Omega}_{\pi} = \langle \Xi, \mathcal{I}, \mathcal{G}, \Omega \rangle$. To compute a sequence of states for each goal hypothesis, we develop an algorithm capable of rebuilding the sequence of states induced by a plan by iterating through the sequence of observations Ω and filling in any gaps due to partial observability. Using a domain model Ξ, we check if a sequence of state observations in the observations Ω_s (Definition 5) is valid by evaluating whether a transition between each pair of consecutive state observations is valid, starting from the initial state $\mathcal{I}$ and the first observation. Formally, let $\Omega^{\mathcal{I}}_{s} = \langle \vec{s_0} = \mathcal{I}, \vec{s_1}, \ldots, \vec{s_n} \rangle$, we check whether $\forall_{i \in [1,\ldots,n]} \exists_{\mathrm{a} \in \mathcal{A}} (\vec{s_i} = \gamma(\vec{s_{i-1}}, \mathrm{a}))$. If the sequence of states is impossible, we conclude that an observation is missing at the point of the invalid transition of the observation trace. Thus, we must predict the next state that should have been observed at this point. To build this approach, we must devise a mechanism to predict missing observations, allowing us to estimate which states are missing from the observation trace of a given plan. We define this problem as estimating the most likely next state s' given a set of consecutive states S. To estimate s', we detail two different approaches: a *statistical* approach relying on a machine learning model and landmark heuristics (detailed in Section 5.1.2), which can compute a probabilistic value to a given vocabulary of states, measuring the probability of each state in a vocabulary

being the next state s', finally using landmark heuristics to decide the most likely state; and a *symbolic* approach (detailed in Section 5.1.3), relying on search algorithm heuristics to predict s', given the set of achievable states. In this section, we refer to this mechanism as the method PREDICTNEXTS, for which we provide different approaches in Sections 5.1.2 and 5.1.3.

Algorithm 2 Recognize a plan for a goal G.

Require: A predictive model $\mathcal{M}$.

1: **function** COMPUTESEQUENCE($\mathcal{I}, \mathcal{A}, \Omega, G, \lambda$)
2: $\quad$ S $\leftarrow \langle \mathcal{I} \rangle$
3: $\quad$ $L_G \leftarrow$ EXTRACTLANDMARKS(G)
4: $\quad$ **if** $\Omega_{|\Omega|} \models G$ **then** $\hat{\Omega} \leftarrow \Omega$
5: $\quad$ **else** $\hat{\Omega} \leftarrow \Omega \cdot G$
6: $\quad$ *predicted* $\leftarrow 0$
7: $\quad$ **for** o in $\hat{\Omega}$ **do**
8: $\quad\quad$ **while** $\neg \exists_{a \in \mathcal{A}}(o = \gamma(\mathrm{S}_{|\mathrm{S}|}, a))$ **do**
9: $\quad\quad\quad$ $s' \leftarrow$ PREDICTNEXTS($\mathcal{M}, \mathcal{A}$, S, $L_G, \langle\rangle, k$)
10: $\quad\quad\quad$ S $\cdot\ s'$
11: $\quad\quad\quad$ *predicted*$+ = 1$
12: $\quad\quad\quad$ **if** $G \in$ S **or** *predicted* $> \lambda$ **then**
13: $\quad\quad\quad\quad$ **return** S
14: $\quad\quad$ S $\cdot\ o$
15: $\quad\quad$ *predicted* $\leftarrow 0$
16: $\quad$ **return** S

Given a mechanism capable of predicting the most likely single missing state s' in a sequence of states, we can use this prediction to fill in any number of missing observations. Our approach fills in all missing observations for a sequence of observations, computing the sequence of states induced by the plan towards a single goal hypothesis in the process. Algorithm 2 formalizes our approach to compute a plan for goal hypothesis G using a predictive model $\mathcal{M}$ and a set of landmarks L_G. This algorithm takes as input an initial state $\mathcal{I}$, a a set of actions $\mathcal{A}$, a set of observations Ω, a goal G, and a threshold λ. To generate a sequence of states S that a plan π achieves for each goal, we use the initial state $\mathcal{I}$ as the starting point (Line 2). Then, we extract landmarks for goal G (Line 3). To extract the landmarks, we use the algorithm proposed by [39]. We concatenate the list of state observations Ω with the goal hypothesis G (Line 5) creating a new list $\hat{\Omega}$. We iterate through $\hat{\Omega}$ (Line 7), checking if o (an observation from $\hat{\Omega}$) is the result of a valid transition from the last known valid state of $\mathcal{S}$ (denoted as $\mathcal{S}_{|\mathcal{S}|}$). If there exists a valid transition between these two states, we add the observation to the sequence of states and move to the next observation (Line 8 and 14). Otherwise, in Line 9, we use PREDICTNEXTS to predict a missing state in the observations. We keep using this function to predict states until the predicted state supports a single action to achieve the next observation in $\hat{\Omega}$. We repeat this process for every observation, including the desired goal G, until we compute a valid sequence of states in the model that can achieve G, returning such list (Line 16). The algorithm stops when it achieves G during the prediction phase, returning the current sequence of states $\mathcal{S}$ (Line 12). If a particular goal hypothesis is improbable, trying to fill in

missing observations from the last known valid transition necessarily induces a substantially sub-optimal plan. In practice, incorrect goal hypotheses induce much longer plans than the correct hypothesis in domains where the state space is connected or infinite plans that never reach it. To prevent the algorithm from generating such plans, we stop trying to complete the plan for a goal hypothesis G if during the prediction process we predict λ (a threshold) states consecutively that are unable to achieve the current observation o, returning the current sequence of states. This threshold can be any heuristic value, estimating the maximum length of a plan.

We solve the problem of plan recognition Π_π^Ω by applying Algorithm 2 to all goals $G \in \mathcal{G}$. To decide which goal is the correct one, we compare the predicted sequence of states S_G for each goal. First, we discard any goal G that is not in the last state in the predicted sequence S_G. Then, we rank the remaining S_G based on their compliance with the set of observations Ω, selecting the sequence of states S_G that complies with most of the observations Ω. If there is a tie, we select the shortest sequence of states S_G, following the notion that agents are at least approximately rational and prefer shorter plans [71], as the most probable sequence of states S_G, and its goal G as the most likely goal. Thus, we predict a single sequence of states (from which we can derive a plan) for a single goal as the most likely goal and plan the agent is pursuing, solving the problem of Definition 7. Let S_G be the sequences predicted by $\textsc{computePlan}(\mathcal{I}, \mathcal{A}, \Omega, G, \lambda)$ such that $\mathrm{S}_G \models G$, the goal that best complies with the observations is:

$$\mathrm{s}_\mathrm{G}^* = \arg\max_{G \in \mathcal{G}} |\, \mathrm{S}_G \cap \Omega \,| \tag{5.1}$$

Figure 5.1 illustrates how our approach employs Algorithm 2 to perform the tasks of goal and plan recognition. Note that our approach PPR returns not only the agent's intended goal but also a sequence of states (i.e., plan) that possibly achieves the goal. Thus, our approach solves the goal recognition problem (Definition 6), the plan recognition problem (Definition 6), and provides a complete solution to the missing observation problem (Definition 10).

To deal explicitly with noisy observations, we develop a variation of Algorithm 2. We adopt the usual notion of noisy observation sequence from Goal Recognition [67, 84], which defines that a specific observation is noisy if it contains observations emitted without a corresponding state or action in the actual plan executed by the agent being observed, as illustrated in the center of Figure 5.2. To recognize plans with noisy observations, we compute plans that can be non-complaint with all observations in Ω. Here, we keep the assumption that the observations are ordered.

Algorithm 2 will likely fail to find a valid transition to o_i in Line 7 when dealing with noisy observations. To overcome this limitation, we must be able to compute plans that ignore the noisy observations. We assume an observation o_i is noisy if we can predict a state induced by subsequent observations (i.e. o_j, such that $m > n$) that can be reached by valid transitions between the last valid inferred state before o_i and o_j. The idea is that we can compute a plan by *skipping* the noisy observation and move to the next observation in the observation trace.

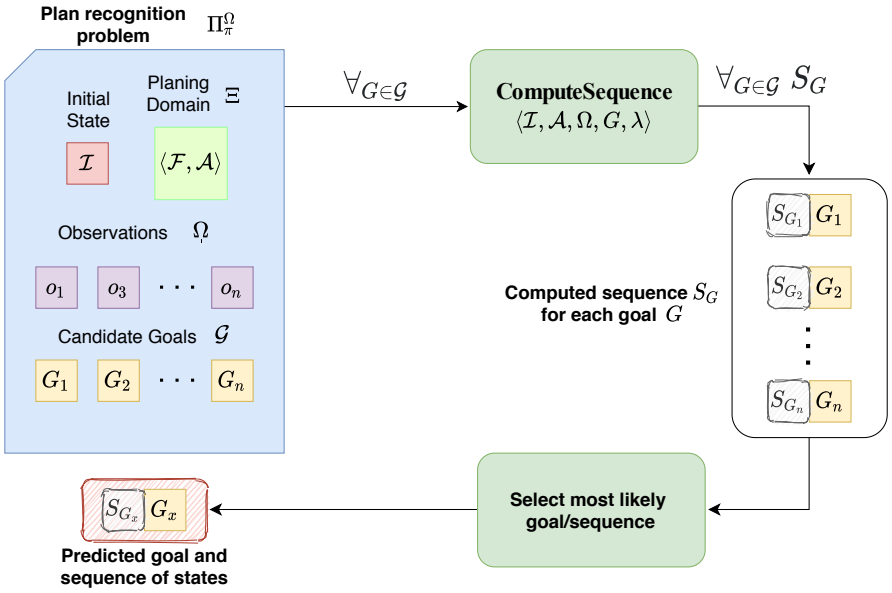


Figure 5.1: PPR Overview.

Algorithm 3 implements this intuition by mostly following the same strategy as Algorithm 2. However, when we are able to skip an observation o_i (Line 10), now assumed to be noisy, to reach o_j, we add such state o_j to the current sequence of states S and continue iterating from observation o_{j+1} (Line 12). The rest of Algorithm 3 is similar to Algorithm 2, returning a sequence of states S that achieves goal G. Figure 5.2 illustrates the process of computing a plan given three observations, where o_2 is a noisy observation. Blue boxes represent states achieved by plan π; purple boxes represent correct observations; gray boxes represent states predicted by Algorithm 4, and finally, red boxes represent noisy observations. In this case example, our approach computes the sequence of states and skips observation o_2 by predicting s_4, which corresponds to observation o_3, thus eliminating the need to achieve a valid transition to the observation o_2. Thus, with this modification, our approach can compute plans that do not comply with the noisy observations.

The idea of this modification is to enable our approach to compute plans without the need to be fully compliant with the observations Ω, thus dealing with noisy observations. Note that we do not guarantee that a skipped state is indeed a noisy observation, but we instead focus on computing valid plans in the given plan recognition problem.

5.1.2 Predicting States Using Machine Learning and Landmarks

Here, we develop a statistical approach to predict the most likely next state s' that relies on a machine learning model and landmark heuristics. First, we train a machine learning model capable of computing each state's probability, in a vocabulary $\mathcal{V}_{\mathcal{F}}$, of being the next state. While any model capable of predicting subsequent states could be used, we use a straightforward neural network, explained in Chapter 5. Figure 4.4 illustrates the model architecture, where

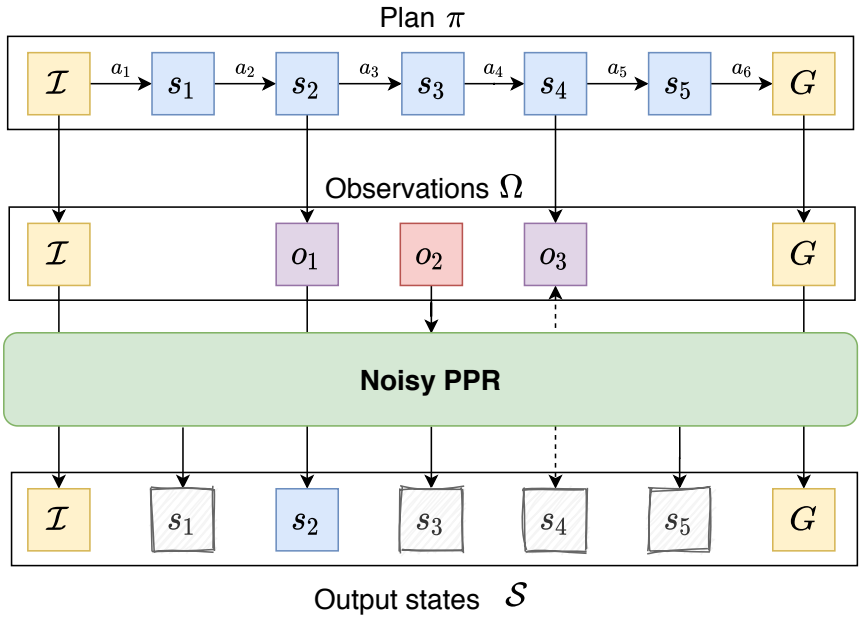


Figure 5.2: Computing S with noisy observations.

Algorithm 3 Recognize a plan for a goal G under noise.

Require: A predictive model $\mathcal{M}$.

function COMPUTESEQUENCEN($\mathcal{I}, \mathcal{A}, \Omega, G, \lambda$)
  S $\leftarrow \langle \mathcal{I} \rangle$
  $L_G \leftarrow$ EXTRACTLANDMARKS(G)
  if $\Omega_{|\Omega|} \models G$ **then** $\hat{\Omega} \leftarrow \Omega$
  else $\hat{\Omega} \leftarrow \Omega \cdot G$
  predicted $\leftarrow 0$; *skip* $\leftarrow \bot$
  for o_i in $\hat{\Omega}$ **do**
    while $\neg\exists_{a \in \mathcal{A}}(o_i = \gamma(\mathrm{S}_{|\mathrm{S}|}, a))$ **and** $\neg$*skip* **do**
      $s' \leftarrow$ PREDICTNEXTS($\mathcal{M}, \mathcal{A}, \mathrm{S}, L_G, \langle\rangle, k$)
      if $s' = o_j \mid o_j \in \hat{\Omega}$ **and** $j > i$ **then**
        skip $\leftarrow \top$
        $\hat{\Omega} \leftarrow \hat{\Omega}_{[j+1:]}$
      S $\cdot\ s'$
      predicted$+ = 1$
      if $G \in$ S **or** *predicted* $> \lambda$ **then**
        return S
    if $\neg$*skip* **then** S $\cdot\ o_i$
    else *skip* $\leftarrow \bot$
    predicted $\leftarrow 0$
  return S

max-len is the maximum length of an input sequence of observations in the dataset and *vocab* is the number of unique tokens of the problem (all possible states that the model can predict). In our approach, we use this model to compute each state's probability of being the individual successor state s', given a sequence of previous states. Formally, our model $\mathcal{M}(\Omega)$ is a function of sequences of states into a probability distribution $P(s' \mid \mathrm{S})$ for all $s' \in \mathcal{V}_{\mathcal{F}}$.

We use a learned model $\mathcal{M}$ and landmark heuristics to steer the machine learning model's predictions $\mathcal{M}$. By selecting the states that achieve more landmarks, we can improve the generated plans without computing the heuristic to all states of the vocabulary. To predict the next state s', we develop a two step process that outputs a state s', which we use to fill the gap of missing observations. We achieve this using function PREDICTNEXTS$(\mathcal{M}, \mathcal{A}, \mathrm{S}, L_G, o, k)$, formalized in Algorithm 4, which takes as parameters a predictive model $\mathcal{M}$, a set of actions $\mathcal{A}$ from a planning domain model Ξ, a sequence of states S, a set of landmarks (Definition 8) for the desired goal G, and a constant k. To compute the landmarks we use the algorithm proposed by Hoffman et. al [39], in Line 3 of Algorithm 2. First, we use the trained machine learning model $\mathcal{M}$ to compute the probability of each state in the vocabulary $\mathcal{V}_{\mathcal{F}}$ being the next state given S_π (Line 2). Recall that $\mathcal{M}(\mathrm{S}_\pi)$ computes the probability distribution of each state, in the vocabulary, being the next state. Then, we select the k most likely next states from the probability distribution given by $\mathcal{M}$ that are achievable after the current sequence of states S_π towards a goal G (Lines 4 to 8). We use k to weigh the trade-off between the probability distribution of the model $\mathcal{M}$ and the estimated distance to the goal hypothesis G. In Algorithm 4, we compute the overlap between the landmarks for a goal and the predicted states $C_{s'} \in C$ so that states with more achieved landmarks are closer to the goal. If $k = 1$, we consider the probabilistic distribution computed by the learned model $\mathcal{M}$ as absolute, ignoring landmarks. If $k = |\mathcal{V}_{\mathcal{F}}|$, we consider the closeness to the goal of every single state, ignoring the probabilistic distribution computed by $\mathcal{M}$.

Algorithm 4 Predict most likely subsequent state s'.

```
1: function PREDICTNEXTS(M, A, S, L_G, o, k)
2:     P_S ← M(S)                                  ▷ Compute P(s' | S) for all successor states s'
3:     C ← {}                                      ▷ Candidate next states.
4:     for s' ∈ P_S ordered by P(s' | S) do         ▷ Order by the computed probability.
5:         if ∃_{a∈A}(s' = γ(S_{|S|}, a)) then
6:             C ← C ∪ s'
7:             if |C| = k then                     ▷ Consider top k candidates
8:                 return arg max_{s'} |(C_{s'} ∩ L_G)|
9:     return arg max_{s'} |(C_{s'} ∩ L_G)|        ▷ Return state with max landmarks.
```

5.1.3 Predicting States Using Symbolic Approach

We develop a symbolic approach to predict the most likely next state s' without machine learning techniques, obviating the need for data. To do so, given a set of states S, we use the domain theory Ξ to compute all achievable next states from the last state of the set S. Given all the achievable states C (candidate next states), we apply domain-independent heuristics to evaluate which state in C is the most likely next state. The heuristic can be any search planning heuristic; in this work, we use the Fast-Forward (h^{FF}) heuristic [38], due to its

good balance in terms of speed and information. To choose the most likely next state in C, we compute the heuristic value h of each state $s \in C$ to the next observation o we are trying to achieve (Line 7 in Algorithm 2) and the goal hypothesis G. Thus, we measure a heuristic value that works as the likelihood of $s \in C$ being the most likely next state s' with the following equation:

$$\frac{h(s, o) + h(s, G)}{2} \tag{5.2}$$

where we measure how close the state is to the next observation and how close it is to the goal hypothesis. We then select the state with the lowest value. If there is a tie, we compare landmark achievability the same way as we did in Section 5.1.2. If there is still a tie, we choose randomly between the tied states. We develop this approach to evaluate the need to use a machine learning model to compute the most likely next state s'. If this approach can compute plans with low observability, there is no need to train a machine learning model.

5.2 Experiments

In this section, we detail our experiments to evaluate the two developed approaches in this chapter. First, we discuss our experimental setup to evaluate our approach with other state-of-the-art approaches for plan recognition. Second, we show the results in plan recognition problems with varying degrees of observability, discussing the achieved results. Finally, we create scenarios with missing observations and noisy observations, discussing how our approach fare in these more complex scenarios.

5.2.1 Experimentation

This approach uses the same four (4) machine learning models discussed in Chapter 4, using the same training samples used for our observation enhancing approach. Here, we use the problems devised in Chapter 4 as plan recognition problems.

We use the 20 planning problems from each domain's test dataset to generate observation traces to run experiments in plan recognition problems and perform the prediction of missing observations using the trained model. These are the same problems we created to evaluate our approach in Chapter 4. We split between the training set and the test set randomly, so there may be states in the test set that are not contained in the training set. Plan recognition techniques use different observability ratios to assess their robustness. We remove observations from the test plans to obtain observability ratios of 10, 30, 50, and 70 percent and full observability. This means that we randomly cut a percentage of observations out of a plan, resulting in traces similar to the one illustrated in Figure 5.2 (without the noise), where three out of seven observations are missing. We keep full observability as it serves as a sanity check

for our approaches. Given that our approaches only fill missing observations, it is expected that both our approaches achieve 100% precision with full observability, as there is no missing observation in these problems.

Finally, we generate plan recognition problems for each domain with each observability ratio, resulting in 100 plan recognition problems for each domain, each problem including six (6) distinct goals hypotheses. We compare our approaches PPR (statistical approach) and h-PPR (symbolic approach, heuristic Predictive Plan Recognition) against three other approaches, RG2009 [71], RG2010 [72], and Mirroring [89]. For h-PPR, we tested using 4 distinct heuristics, such as: h^{max} [34], h^{add} [34], h^{+} [38], and h^{FF}. However, we only show the results using h^{FF}, as h^{FF} outperforms the other three heuristics greatly. While h^{+} achieves better results if allowed to run with unlimited time, most problems take more than the 20-minute timeout to complete.

We ran all experiments in a single core of a 24 core Intel Xeon vE5-2620 CPU @2.0GHz with 160GB of RAM and a memory limit of 2GB, and an NVIDIA Titan Xp GPU. For these experiments, we set the timeout as 20 minutes (1200 seconds) for each plan recognition problem. For latent space problems (MNIST and LOGISTICS), RG2010 timed out in all plan recognition problems (evidenced by the average time of 1200 seconds in all observability cases). Since RG2010 still outputs results even timing out, we included them in Table 5.1. We note that the Mirroring approach [88, 89] timed out and had out-of-memory errors for most recognition problems. Therefore, we decided not to report the results for just a few recognition problems.

5.2.2 Results using Missing and Full Observations

Table 5.1 shows the results for all four domains with five degrees of observability comparing the three approaches. The column P measures *precision*, which is how many times each approach was able to compute a valid plan for the correct goal and select such plan as the most likely one between six goals hypotheses, divided by the number of recognized goals. Since our approaches (PPR and h-PPR) always return a single goal, precision measures our approach's accuracy. If our approaches predict the correct goal but cannot compute a valid plan for such a goal, we consider that the approach failed to predict the correct goal. We decided to use precision (P) instead of accuracy, as the spread for latent space domains for both RG2009 and RG2010 is close to 6 (the number of goals hypothesis), making accuracy a less informative metric. The column $|R|$ shows the average number of returned goals for each approach. As state above, the PPR spread is always one. The $t(s)$ column measures the average time an approach takes to solve a problem in any given domain. Finally, the π^* column measures the percentage of optimal plans found since our approach has no guarantee of optimality. We compute this value by dividing the number of optimal plans found by the number of correctly predicted goals, e.g., if the accuracy is 1.00 in 20 problems and 19 of these plans are optimal, π^* is 0.95 (19/20). PPR was able to compute plans with high precision outperforming the other four approaches in almost every scenario. For LOGISTICS, PPR struggled to compute

complete plans that achieve the correct goal. This is likely due to the inherent parallelism of the LOGISTICS domain, which enables multiple linearizations for the same plan, confusing the predictive model. The Mirroring approach failed to compute plans during our timeout for the latent space domains, so there are no results for these domains.

		PPR				RG2009				RG2010				Mirroring				h-PPR			
	Ω %	P	$\|R\|$	$t(s)$	π^*	P	$\|R\|$	$t(s)$	π^*	P	$\|R\|$	$t(s)$	π^*	P	$\|R\|$	$t(s)$	π^*	P	$\|R\|$	$t(s)$	π^*
BLOCKS	10%	0.60	1.0	7.05	0.92	0.25	3.00	0.33	1.0	0.36	1.40	99.35	1.0	0.05	1.0	0.03	1.0	0.10	1.0	9.33	1.0
	30%	0.70	1.0	6.86	1.0	0.27	2.50	0.34	1.0	0.62	1.30	132.39	1.0	0.00	1.0	0.27	1.0	0.25	1.0	6.07	0.40
	50%	0.95	1.0	6.56	1.0	0.39	2.00	0.38	1.0	0.87	1.10	170.42	1.0	0.16	1.0	0.27	1.0	0.40	1.0	6.36	0.37
	70%	1.0	1.0	6.96	0.95	0.45	1.60	0.43	1.0	0.87	1.10	197.01	1.0	0.22	1.0	0.22	1.0	0.75	1.0	5.01	0.60
	100%	1.0	1.0	7.18	1.0	0.62	1.30	0.52	1.0	0.91	1.10	244.93	1.0	1.0	1.0	0.22	1.0	1.0	1.0	4.32	1.0
LOGISTICS	10%	0.30	1.0	381.11	0.67	0.26	3.20	0.46	1.0	0.35	2.10	5.17	1.0	0.08	1.20	0.35	1.0	0.05	1.0	86.62	0.00
	30%	0.60	1.0	377.40	0.50	0.37	2.10	0.49	1.0	0.50	1.60	6.97	1.0	0.10	1.05	0.25	1.0	0.15	1.0	62.44	0.33
	50%	0.70	1.0	333.21	0.57	0.38	1.70	0.53	1.0	0.70	1.10	10.58	1.0	0.09	1.10	0.25	1.0	0.30	1.0	53.92	0.33
	70%	0.95	1.0	252.35	0.53	0.53	1.60	0.59	1.0	0.70	1.10	15.26	1.0	0.19	1.05	0.20	1.0	0.45	1.0	47.88	0.55
	100%	1.0	1.0	157.44	1.0	0.68	1.20	0.72	1.0	0.77	1.10	22.27	1.0	1.0	1.0	0.25	1.0	1.0	1.0	35.33	1.0
LO-DIGITAL	10%	0.85	1.0	238.37	0.94	0.19	5.30	79.62	1.0	0.19	5.20	1331.53	1.0	-	-	-	-	0.30	1.0	52,95	0.66
	30%	0.95	1.0	256.15	0.95	0.20	4.90	75.86	1.0	0.19	5.10	1328.23	1.0	-	-	-	-	0.55	1.0	59,71	0.27
	50%	1.0	1.0	262.05	0.95	0.20	4.90	79.46	1.0	0.19	4.80	1234.43	1.0	-	-	-	-	0.55	1.0	34,84	0.81
	70%	1.0	1.0	275.41	1.0	0.21	4.60	84.04	1.0	0.19	4.80	1210.44	1.0	-	-	-	-	1.0	1.0	14,39	0.95
	100%	1.0	1.0	253.74	1.0	0.21	4.80	86.16	1.0	0.19	4.80	1264.16	1.0	-	-	-	-	1.0	1.0	8.37	1.0
MNIST	10%	0.90	1.0	96.17	0.83	0.18	5.40	41.81	1.0	0.19	4.90	1385.02	1.0	-	-	-	-	0.25	1.0	7.55	0.40
	30%	0.95	1.0	105.95	0.84	0.18	5.30	38.02	1.0	0.19	4.90	1433.15	1.0	-	-	-	-	0.35	1.0	8.15	0.14
	50%	0.90	1.0	96.40	0.89	0.17	5.15	42.40	1.0	0.19	4.90	1249.50	1.0	-	-	-	-	0.50	1.0	6.95	0.40
	70%	1.0	1.0	77.07	0.95	0.19	5.30	42.02	1.0	0.17	5.20	1213.90	1.0	-	-	-	-	0.85	1.0	4.78	0.76
	100%	1.0	1.0	65.48	1.0	0.19	5.30	43.32	1.0	0.19	4.90	1241.75	1.0	-	-	-	-	1.0	1.0	1.43	1.0

Table 5.1: Recognition results when dealing with missing and full observations. P and π^* have values between 0 and 1.0, and higher values indicate better precision. The ideal value for $|R|$ is 1, indicating that only one goal has been recognized.

5.2.3 Results using Noise, Missing, and Full Observations

To evaluate our approaches, we introduce noise in the observations of our plan recognition problems. Thus, we create two new datasets for each of our domains, one with 10% noise and the other with 20% noise. We introduce noise by iterating through all observations of the dataset, setting a probability of either 10% or 20% of this observation becoming an invalid observation for a given plan. Thus, a dataset with 10% of noise means that, of all observations in the dataset, 10% are invalid observations through all problems, with no guarantee that a given problem will have a noisy observation. Table 5.2 shows the results for the 8 noisy datasets. PPR vastly outperforms RG2009 in terms of precision, using as trade-off time. Through all the datasets, RG2009 had a high spread, negatively affecting the precision. RG2009 had low precision in noisy domains since it tries to compute an optimal plan including noisy observations (while PPR skips noisy observations). We note that PPR has better predictions in the LOGISTICS domain when dealing with noise since we can compute plans that are not fully compliant with the observations. We assume that this helps PPR to compute plans in domains that have inherent parallelism we mentioned before.

In this comparison, we used only RG2009 as RG2010 and Mirroring were unable to compute plans with the noisy dataset in the given twenty (20) minutes timeout. Since h-PPR was outperformed in all domains by PPR, we decided to include only the results for the PPR approach.

PPR															
Ω %	10%			30%			50%			70%			100%		
Domain (Noise %)	*P*	π^*	*t(s)*	*P*	π^*	*t(s)*	*P*	π^*	*t(s)*	*P*	π^*	*t(s)*	*P*	π^*	t(s)
Blocks (10%)	0.60	0.92	15.32	0.75	0.87	11.08	0.95	1.00	17.81	1.00	0.95	11.52	0.95	1.00	9.13
Blocks (20%)	0.70	0.93	19.93	0.65	0.85	13.06	0.80	1.00	15.70	0.85	0.94	13.78	0.85	1.00	8.84
Logistics (10%)	0.70	0.86	322.35	0.70	0.79	301.75	0.75	0.73	331.56	0.90	0.61	245.71	0.90	1.00	161.19
Logistics (20%)	0.70	0.79	290.43	0.65	0.77	312.07	0.70	0.71	287.98	0.90	0.83	303.21	0.85	1.00	191.07
LO-Digital (10%)	0.85	0.94	321.86	0.95	0.95	298.46	0.95	0.95	353.13	1.00	0.90	275.98	0.95	1.00	276.51
LO-Digital (20%)	0.85	0.94	302.62	0.90	0.94	281.01	1.00	0.95	273.39	0.85	0.88	300.21	0.95	0.84	350.13
MNIST (10%)	0.90	0.83	117.79	0.95	0.84	99.21	0.85	0.88	103.15	1.00	0.90	122.34	0.95	0.89	166.74
MNIST (20%)	0.90	0.83	98.65	0.85	0.82	112.79	0.90	0.89	101.55	1.00	0.85	113.60	1.00	0.85	152.22
RG2009															
Ω %	10%			30%			50%			70%			100%		
Domain (Noise %)	*P*	π^*	*t(s)*	*P*	π^*	*t(s)*	*P*	π^*	*t(s)*	*P*	π^*	*t(s)*	*P*	π^*	*t(s)*
Blocks (10%)	0.10	1.00	0.31	0.16	1.00	0.32	0.11	1.00	0.35	0.12	1.00	0.38	0.08	1.00	0.45
Blocks (20%)	0.16	1.00	0.30	0.09	1.00	0.33	0.03	1.00	0.35	0.03	1.00	0.36	0.03	1.00	0.43
Logistics (10%)	0.25	1.00	0.18	0.34	1.00	0.19	0.45	1.00	0.22	0.40	1.00	0.29	0.60	1.00	0.45
Logistics (20%)	0.31	1.00	0.18	0.34	1.00	0.19	0.44	1.00	0.22	0.41	1.00	0.25	0.53	1.00	0.33
LO-Digital (10%)	0.18	1.00	85.08	0.19	1.00	84.22	0.20	1.00	83.80	0.21	1.00	86.25	0.20	1.00	87.35
LO-Digital (20%)	0.19	1.00	83.77	0.20	1.00	86.48	0.21	1.00	88.27	0.19	1.00	91.42	0.20	1.00	93.28
MNIST (10%)	0.18	1.00	26.30	0.18	1.00	27.19	0.18	1.00	28.05	0.19	1.00	27.95	0.19	1.00	28.70
MNIST (20%)	0.18	1.00	36.27	0.18	1.00	35.62	0.18	1.00	26.30	0.19	1.00	27.49	0.19	1.00	28.56

Table 5.2: Recognition results when dealing noise observations.

5.3 Chapter remarks

In this chapter, we introduced two novel approaches for goal and plan recognition, one that combines machine learning statistical prediction with domain knowledge within classical planning techniques and an approach that relies only on domain theory. These approaches achieve very high precision both in handcrafted and automatically generated plan recognition domains. We empirically show that our approaches can compute plans with very low observability (up to 90% missing) and noisy observations (up to 20% noise). While our machine learning approach relies on data, we show the amount of data to be much smaller than that needed to generate the learned domains [6, 10], which already necessitates data. Our machine learning model is simple enough that the same network architecture works for all domains without requiring any model tuning for it to work. Indeed, we show we can use an entirely symbolic substitute for the neural network and predict the next states looking ahead one step using heuristic search. The symbolic approach results were inferior to the statistical approach but still competitive with other plan recognition approaches. There is always the possibility of improving the model accuracy to improve our approach. However, we argue that proving that our approach can achieve high precision with a simple model is more of a contribution than developing highly complex models for each domain.

Our statistical approach's main limitation is the requirement of data, which is absent in standard plan recognition approaches. This limitation entails that the length of the binarized vector representing state features imposes a limit on the networks' generalization within the same domain. This limitation is relatively substantial for standard plan recognition using handcrafted domain theories. However, we argue that this is less of an issue in automatically learned domains (e.g., in latent space [11]), as they are both intrinsic to the learned domain. On the flip side, the predictive model also implicitly encodes a preference relation for goals given a sequence of observations, i.e., $P(\Omega \mid G)$ used for some approaches in the literature [72, 84].

6. RELATED WORK

In this chapter, we survey the most important literature related to our thesis. First, we survey the most important works on bridging the symbolic approach with real-world data. Second, we discuss work done on goal and plan recognition using machine learning approaches. Finally, we review symbolic goal recognition approaches that are related to our contributions.

6.1 Inferring domain knowledge

One of the main aspects of this thesis is the pursuit of an approach capable of inferring domain knowledge or obviating the need for it. This avenue of research aims to obviate the need for a domain expert in planning tasks (including plan and goal recognition). One of the most relevant works related to this thesis is Asai and Fukunaga's work, titled "Planning in Latent Space" [10]. In that paper, the authors develop a planning architecture capable of planning using only pairs of images (representing the initial and goal states) from the domain by converting the images into a latent space representation. Their architecture consists of a variational autoencoder (VAE) followed by an off-the-shelf planning algorithm. The architecture converts images into discrete latent vectors using the VAE. It uses the information in such latent vectors to plan over the images and find a sequence of actions that transforms the state into one matching the goal image.

Although our first approach for recognizing goals in latent space is based on the one proposed by Asai and Fukunaga, there are two main differences between them. First, they do not create a PDDL domain file; instead, they train a neural network to act as an action discriminator. This action discriminator is responsible for defining which actions can be performed in each state and validating them. By inferring the PDDL domain, we can use classical goal and plan recognition in such domains. Having a PDDL domain greatly extends the range of classical planning techniques we can apply in these inferred domains. Second, their goal is to plan over the converted images from a domain, whereas we perform a goal and plan recognition over the generated PDDL domain from latent vectors. Thus, we have extended their architecture by allowing both planning and recognition tasks over the latent vectors.

Asai kept researching mechanisms to infer domain knowledge from raw data. In work "Unsupervised Grounding of Plannable First-Order Logic Representation from Images" [9], they developed an approach to compute PDDL domains with a lifted representation, which uses a novel First-Order State Auto-Encoder (FOSAE). Unlike our work, a lifted representation is closer to a hand-made domain by a domain expert, which allows a more compact PDDL domain with better generalization. By using FOSAE, the authors can learn PDDL from data streams and solve planning tasks in the learned PDDL. While promising, their work still has limitations regarding the computed PDDL. The learned FOL statements are quantifier-free, grounded

representation, limiting generalization for problems with a different number of objects (such as more blocks in blocks-world).

Finally, inspired by recent advances on unsupervised learning approaches to apply planning in the latent space and infer domain model (including our work), Alejandro *et al.* [85] explores the unsupervised generation of action models. Not only preconditions and effects are learned, but action signatures too. They use a compilation-based approach to classical planning, which computes a lifted PDDL domain file by using plan traces (sequence of actions) as training data. Different from our approach, they focus on inferring a lifted PDDL domain from plan traces, instead of a grounded domain from images. Nevertheless, this is a promising work to achieve a lifted domain representation without the need of a domain expert, relying purely on unsupervised learning.

6.2 Machine Learning in plan and goal recognition problems

Relying on machine learning to solve the goal recognition task, Min *et al.*develops a deep LSTM network approach to recognize a player's goals in an educational game scenario [54]. The dataset used for training the deep LSTM is a player behavior corpus consisting of specific player actions. The challenge comes from recognizing goals when handling uncertainty from noise input and non-optimal player behavior. The LSTM can do standard metric-based goal recognition, and online goal recognition as information is fed. Although their approach has some similarities to our third approach, our approach computes the entire plan using the domain model, solving the problem of plan recognition, which is more complicated than the goal recognition problem. Moreover, our approach can correctly infer goals that are not contained in the training dataset (referenced in Table 4.1 as $|\mathcal{V}_{\mathcal{F}}|$), as long as they are reachable with our vocabulary.

By combining more complex machine learning approaches, Mariane *et al.* [50] created a series of machine learning models for goal recognition and compared the performance of these models when recognizing goals in domains from the IPC. They compare the performance of LSTM networks, Convolutional networks, and Fully connected networks in IPC domains, including the Blocks-world and Logistics domain. Moreover, they test their approaches in continuous domains, which is an avenue of research we will like to explore in future work. The models developed in their approach can only solve the problem of goal recognition, and as with most machine learning models, explaining the rationale of the model's inferences can be a difficult task. They conclude their work suggesting that the exploration of hybrid approaches for recognition, combining machine learning and automated planning, are worth pursuing to achieve a more robust goal and plan recognition.

Still leveraging LSTMs, Zhuo *et al.*developed an approach to solve plan recognition problems using learned shallow models [96] capable of predicting the next action given a set of observed actions, aiming to reconstruct a plan. Similar to our plan recognition approach

(Chapter 5), they use recurrent networks for the prediction and build a complete plan. Unlike our work, they predict actions instead of states and assume substantially higher observability and no noise for their dataset. They avoid the necessity for a domain model, as only the neural network is responsible for predicting missing observations. Hence, the lack of a domain model enables the possibility of predicting actions that are not valid given the current sequence of actions. By contrast, our work deals with very low observability and noisy observations but requires domain knowledge to ensure the predicted states are valid. We wanted to compare our work results with this approach directly. However, the code available in their GitHub was missing key components and not working properly. Namely, the dataset used for training the networks was also missing.

Finally, Granada *et al.*, [32] developed a hybrid approach that combines activity and plan recognition for video streams. This approach uses deep learning to analyze video data (frames) to identify individual actions in a scene. Based on this set of identified actions, this approach uses a plan recognition algorithm and then a plan library describing possible overarching activities for recognizing the ultimate goal of the subject in a video. Unlike our work, their approach relies on handcrafted plan libraries, requiring a substantial amount of human-derived domain knowledge.

6.3 Plan and Goal Recognition as Planning

Perhaps the most relevant work to this research is the foundation of the goal and plan recognition as planning. Ramírez and Geffner [71] propose planning approaches for goal and plan recognition. Instead of using plan-libraries, they model the problem as a planning domain theory with respect to a known set of goals hypotheses. This work uses a modified heuristic, an optimal and modified sub-optimal planner to determine the distance to every goal in a set of candidate goals given a sequence of observations.

More recently, Pereira, Oren, and Meneguzzi [65] developed landmark-based approaches for goal recognition. More specifically, they develop two fast and accurate heuristics for goal recognition. Their first approach, called Goal Completion Heuristic, computes the ratio between the number of achieved landmarks and the total number of landmarks for a given candidate goal. The second approach, called Uniqueness Heuristic, uses the concept of landmark uniqueness value, representing the information value of the landmark for a particular candidate goal when compared to landmarks for all candidate goals. Thus, the heuristic estimative provided by this heuristic is the ratio between the sum of the achieved landmarks' uniqueness value and the sum of the uniqueness value of all landmarks of a candidate goal. We use these heuristics in our work to evaluate our first approach in Chapter 3.

In another recent approach, Sohrabi *et al.*, [84] develops a probabilistic approach for recognizing goals and plans that deals explicitly with unreliable and spurious observations (i.e., missing and noisy observations). They rely on a compilation of the recognition problem

into purely planning by introducing costs in the action descriptions to account for missing or noisy observation. Since their approach relies on generating numerous plans using an expensive top-k planner, its runtime is substantially high. It often takes hours to solve relatively simple problems, as shown in the results of Table 5.1.

7. CONCLUSION

We introduced three approaches that combine classical planning and machine learning techniques to increase the scope and improve the precision of plan and goal recognition approaches. We developed novel plan recognition approaches that can outperform state-of-the-art recognition approaches in inferred domains. In Section 7.1 we detail our main contributions. In Section 7.2, we explain the limitations of our approaches and still open issues. Finally, in Section 7.3, we discuss future work that can improve our contributions and the implications of our contributions.

7.1 Contributions

We now outline the three main contributions of this thesis, as follows.

1. Our first contribution, detailed in Chapter 3, is related to the task of goal and plan recognition in latent space, where we must recognize goals and plans in image-based domains, resulting in the following contributions.

   - A new problem formalization for recognition goals in image-based problems (Section 3.1), using as base the formalization of Ramirez and Geffner [71, 72]. This formalization allows image-based recognition problems to use an inferred domain model and image representation for the initial state, goal hypotheses, and observations.
   - A novel algorithm (Algorithm 1, Section 3.2) that computes a PDDL domain from pair of images of an image domain. This algorithm allows us to infer domain knowledge from image-based domains, allowing us to solve image-based recognition problems.
   - A dataset of image-based recognition problems and transitions to generate the inferred domains, based on the problems described by Asai and Fukunaga [10, 11].

2. The second contribution of this thesis is a data-driven method to enhance observations in goal and plan recognition problems, detailed in Chapter 4. The following are the specific contribution of this approach.

   - A new problem formalization for the missing observation problem, where we specify two types of solutions: partial and complete. This formalization allows us to improve the precision of recognition approaches without actually solving the goal and plan recognition problems.
   - A novel approach to enhance observations in goal and plan recognition problems (Section 4.1), which relies on training data. This approach can improve the accuracy

of state-of-the-art goal recognition and plan recognition approaches [64, 71, 72, 84] with a minimal increase in spread.

Our final contribution is a novel approach to solve the plan recognition problem, thus solving the goal recognition problem and the missing observation problem. The following algorithms are the specific contributions of this approach.

- A novel algorithm (Algorithm 2, Section 5.1.1) to recognize plans, called Predictive Plan Recognition (PPR), which allows us to compute plans using a predictor function.
- A variation of the PPR algorithm (Algorithm 3), which allows us to compute plans that do not comply with all observations, skipping noisy observations.
- An implementation of a predictor function (Algorithm 4, Section 5.1.2) which relies on machine learning techniques and landmark heuristics to predict the most likely next state.
- The second implementation of a predictor function (Section 5.1.3), which leverages planning heuristics to predict the most likely next state. This approach does not need data to predict the next state, but it is much less accurate than the machine learning and landmarks counterpart.

7.2 Open Issues and limitations

This thesis's contributions are largely related to applying machine learning techniques to well-known goal and plan recognition problems. In this section, we discuss the issues and limitations of our approaches, for both recognizing goals and plans in image-based domains, as solving the plan recognition problem.

The main limitation of our approaches is the reliance on training data for our machine learning models. Our first approach to recognize goals in latent space relies on training data to train reliable auto-encoders for each domain and data to infer the domain model. The data to train the auto-encoder may not be extensive. However, if the auto-encoder is incapable of creating a distinctive representation for each state, we are unable to guarantee a complete inferred PDDL model. Besides an auto-encoder capable of creating a distinct representation for each state in the domain, our algorithm requires every single encoded transition to guarantee a complete PDDL model. Without a complete PDDL description, there may be unreachable states in a goal or plan recognition problem.

Our second and third approaches rely on data to compute machine learning models capable of predicting the most likely next state. While the data required is not extensive, as shown in Table 4.1 (Chapter 4), this limitation is absent in standard plan recognition approaches. Given that image-based recognition problems already require data, we can expect

that training data is available for such problems. However, our approaches are directly impacted by the quality of the data and the number of distinct states in such data. Our approach can only predict states contained in the vocabulary $\mathcal{V}_{\mathcal{F}}$, so if the training set is minimal, the number of possible predicted states would be limited as well.

7.3 Discussion

While we substantially advanced the state-of-the-art in goal and plan recognition using unstructured data, specific challenges remain in each of the techniques we developed, which constituted exciting future work. First, our approach for recognition in latent space infers many redundant actions. We aim to improve the inferred domains by pruning redundant actions in the domain inference process and encoding domains with incomplete information to account for the noise and inconsistencies generated by the auto-encoder. As an example, the encoded lights-out domain has many more actions than its hand-coded counterpart. As the complexity of the problems increases, we expect the number of actions to increase. Second, investigating plan recognition algorithms for incomplete domain models to cope with the inaccuracies of the PDDL inference algorithm is a promising avenue of research. For our second and third contributions, we want to focus on improving two aspects: extend these approaches for continuous domains and online goal recognition; improve the computed machine learning model and its applicability. First, we intend to investigate the possibility of extending this work to continuous domains. A different model would be needed to apply our technique in continuous domains. However, there is much research about learning approaches capable of approximating continuous domains, some of which applied to goal recognition [68, 76, 92]. Second, we intend to investigate more complex neural networks and the ability of these networks to generalize in all our developed domains.

The contributions detailed in this thesis can be applied in many goal and plan recognition scenarios, either as a relaxation of the amount of domain knowledge needed or as a tool to solve the problem of plan recognition itself. Our first approach capable of inferring domains is already being used or serving as inspiration for future work regarding domain inference [85]. Our third approach can serve as the base for any predictor function, be it a heuristic method, a machine learning model, or a combination of both. As research on machine learning and automated planning advances, we expect various possible improvements that can be applied in all of our approaches. Regarding our first approach, as the research in auto-encoder advances, the range of image-based scenarios we can apply our approach increases. We experimented executing the auto-encoder used in Chapter 3, developed by Asai and Fukunaga [10], in a dataset using real-world photos of the Hanoi tower domain. Even after applying image processing techniques, the auto-encoder could not correctly encode and decode the states of this domain. As research in auto-encoders advances, we expect to encode such domains without much fine-tuning and even infer PDDL in video streams. For our second and third approaches, as research on machine learning advances, we expect more sophisticated models to predict

missing states, probably without the need for a vocabulary. While developing our second and third approaches, we did experiment with a variation of our LSTM (Figure 4.4, Chapter 4) with a sigmoid activation function at the end to reconstruct the state, instead of relying on a vocabulary. The results were far inferior to our current approach; however, we believe that more sophisticated state-of-the-art machine learning models can improve our second and third approaches' performance without the necessity of extensive fine-tuning for all domains. Finally, for our third approach, many possible predictors functions can be developed to assist our plan recognition technique, besides the two developed in this thesis. As research advances in automated planning and machine learning, new heuristics and machine learning models can be applied to predict missing states, allowing research of new and promising predictor functions.

BIBLIOGRAPHY

[1] Akita, R.; Yoshihara, A.; Matsubara, T.; Uehara, K. “Deep learning for stock prediction using numerical and textual information”. In: Proceedings of the International Conference on Computer and Information Science (ICIS), 2016, pp. 1–6.

[2] Alpaydin, E. “Introduction to Machine Learning”. Cambridge, MA, USA: The MIT Press, 2010, 2nd ed., 613p.

[3] Amado, L.; Aires, J. P.; Pereira, R. F.; Magnaguagno, M.; Granada, R.; Licks, G. P.; Marcon, M.; Meneguzzi, F. “LatRec+: Learning-based Goal Recognition in Latent Space (Demo)”. In: Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR), 2020, pp. 1101–1102.

[4] Amado, L.; Aires, J. P.; Pereira, R. F.; Magnaguagno, M.; Granada, R.; Licks, G. P.; Meneguzzi, F. “LatRec: Recognizing Goals in Latent Space (Demo)”. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2019, pp. 711–712.

[5] Amado, L.; Meneguzzi, F. “LatRec: Recognizing Goals in Latent Space”. In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2020, pp. 13747–13748.

[6] Amado, L.; Pereira, R. F.; Aires, J. P.; Magnaguagno, M.; Granada, R.; Meneguzzi, F. “Goal recognition in latent space”. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2018, pp. 413–420.

[7] Amado, L.; Pereira, R. F.; Aires, J. P.; Magnaguagno, M.; Granada, R.; Meneguzzi, F. “LSTM-Based Goal Recognition in Latent Space”. In: Proceedings of the ICML / IJCAI / AAMAS Workshop on Planning and Learning (PAL), 2018, pp. 1–8.

[8] Amado, L.; Pereira, R. F.; Aires, J. P.; Magnaguagno, M.; Granada, R.; Meneguzzi, F. “An LSTM-Based Approach for Goal Recognition in Latent Space”. In: Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR), 2019, pp. 256–265.

[9] Asai, M. “Unsupervised grounding of plannable first-order logic representation from images”. In: Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS), 2019, pp. 583–591.

[10] Asai, M.; Fukunaga, A. “Classical Planning in Deep Latent Space: From Unlabeled Images to PDDL (and back)”. In: Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS), 2017, pp. 27–35.

[11] Asai, M.; Fukunaga, A. “Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary”. In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2018, pp. 6094–6101.

[12] Avrahami-Zilberbrand, D.; Kaminka, G. A. “Fast and complete symbolic plan recognition”. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 653–658.

[13] Avrahami-Zilberbrand, D.; Kaminka, G. A. “Incorporating observer biases in keyhole plan recognition (efficiently!)”. In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2007, pp. 944–949.

[14] Bahdanau, D.; Cho, K.; Bengio, Y. “Neural machine translation by jointly learning to align and translate”. In: Proceedings of the International Conference on Learning Representations (ICLR), 2015, pp. 247–266.

[15] Bengio, Y.; Simard, P.; Frasconi, P. “Learning long-term dependencies with gradient descent is difficult”, *IEEE Transactions on Neural Networks*, vol. 5–2, Mar 1994, pp. 157–166.

[16] Bishop, C. M. “Pattern Recognition and Machine Learning (Information Science and Statistics)”. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, 758p.

[17] Brownlee, J. “Clever Algorithms: Nature-Inspired Programming Recipes”. Morrisville, NC, USA: Lulu.com, 2011, 1st ed., 436p.

[18] Cheng, J.; Dong, L.; Lapata, M. “Long short-term memory-networks for machine reading”. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2016, pp. 551–561.

[19] Cybenko, G. “Approximation by superpositions of a sigmoidal function”, *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2–4, December 1989, pp. 303–314.

[20] Fagundes, M.; Meneguzzi, F.; Bordini, R. H.; Vieira, R. “Dealing with ambiguity in plan recognition under time constraints”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2014, pp. 389–396.

[21] Fikes, R.; Nilsson, N. “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”, *Artificial Intelligence*, vol. 2–3-4, Sep 1971, pp. 189–208.

[22] Fleischer, R.; Yu, J. “A Survey of the Game “Lights Out!””. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, chap. 13, pp. 176–198.

[23] Fox, M.; Long, D. “PDDL2. 1: An extension to PDDL for expressing temporal planning domains.”, *Journal of Artificial Intelligence Research*, vol. abs/1106.4561, May 2003, pp. 61–124.

[24] Geffner, H.; Bonet, B. “A Concise Introduction to Models and Methods for Automated Planning”. San Rafael, CA, United States: Morgan Claypool, 2013, vol. 7, 141p.

[25] Geib, C. W. “Problems with Intent Recognition for Elder Care”. In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2002, pp. 13–17.

[26] Geib, C. W.; Goldman, R. P. “Plan Recognition in Intrusion Detection Systems”. In: Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX), 2001, pp. 46–55.

[27] Geib, C. W.; Goldman, R. P. “A probabilistic plan recognition algorithm based on plan tree grammars.”, *Artificial Intelligence*, vol. 173–11, July 2009, pp. 1101–1132.

[28] Geib, C. W.; Steedman, M. “On natural language processing and plan recognition”. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 1612–1617.

[29] Ghallab, M.; Nau, D. S.; Traverso, P. “Automated Planning and Acting”. Alpharetta, GA, USA: Elsevier, 2016, 1st ed., 373p, 368p.

[30] Glorot, X.; Bordes, A.; Bengio, Y. “Deep sparse rectifier neural networks”. In: Proceedings of the International Conference on Artificial Intelligence and Statistics (AIStat), 2011, pp. 315–323.

[31] Goodfellow, I.; Bengio, Y.; Courville, A. “Deep Learning”. Cambridge, MA, USA: The MIT Press, 2016, 800p.

[32] Granada, R.; Pereira, R. F.; Monteiro, J.; Barros, R.; Ruiz, D.; Meneguzzi, F. “Hybrid Activity and Plan Recognition for Video Streams”. In: Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR), 2017, pp. 819–825.

[33] Gumbel, E. J. “Statistical theory of extreme values and some practical applications: a series of lectures”. Washington, NW, United States: U. S. Govt. Print. Office, 1954, 60p.

[34] Haslum, P.; Geffner, H. “Admissible heuristics for optimal planning”. In: Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS), 2000, pp. 140–149.

[35] Helmert, M. “The fast downward planning system”, *Journal of Artificial Intelligence Research*, vol. 26, Jun 2006, pp. 191–246.

[36] Hinton, G.; Salakhutdinov, R. “Reducing the dimensionality of data with neural networks”, *Science*, vol. 313–5786, Jul 2006, pp. 504–507.

[37] Hochreiter, S.; Schmidhuber, J. “Long short-term memory”, *Neural Computation*, vol. 9–8, November 1997, pp. 1735–1780.

[38] Hoffmann, J.; Nebel, B. “The FF planning system: Fast plan generation through heuristic search”, *Journal of Artificial Intelligence Research*, vol. 14, May 2001, pp. 253–302.

[39] Hoffmann, J.; Porteous, J.; Sebastia, L. “Ordered Landmarks in Planning”, *Journal of Artificial Intelligence Research*, vol. 22–1, April 2004, pp. 215–278.

[40] Jang, E.; Gu, S.; Poole, B. “Categorical reparameterization with gumbel-softmax”. In: Proceedings of the International Conference on Learning Representations (ICLR), 2017, pp. 1–13.

[41] Jiménez, S.; de la Rosa, T.; Fernández, S.; Fernández, F.; Borrajo, D. “A review of machine learning for automated planning.”, *Knowledge Engineering Review*, vol. 27–4, Nov 2012, pp. 433–467.

[42] Kautz, H. A.; Allen, J. F. “Generalized Plan Recognition.” In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 1986, pp. 32–37.

[43] Keren, S.; Gal, A.; Karpas, E. “Goal Recognition Design.” In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2014, pp. 154–162.

[44] Kim, S.; Zhang, H.; Wu, R.; Gong, L. “Dealing with noise in defect prediction”. In: Proceedings of the International Conference on Software Engineering (ICSE), 2011, pp. 481–490.

[45] Kingma, D. P.; Mohamed, S.; Rezende, D. J.; Welling, M. “Semi-supervised Learning with Deep Generative Models”. In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2014, pp. 3581–3589.

[46] Kingma, D. P.; Welling, M. “Auto-Encoding Variational Bayes”, *arXiv.org*, vol. 2, December 2013, pp. 199–213, 1312.6114v10.

[47] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. “Imagenet classification with deep convolutional neural networks”. In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2012, pp. 1097–1105.

[48] Maddison, C. J.; Tarlow, D.; Minka, T. “A vast sampling”. In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2014, pp. 3086–3094.

[49] Martín, Y. E.; Moreno, M. D. R.; Smith, D. E. “A Fast Goal Recognition Technique Based on Interaction Estimates.” In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2015, pp. 761–768.

[50] Maynard, M.; Duhamel, T.; Kabanza, F. “Cost-based goal recognition meets deep learning”, *Computing Research Repository (CoRR)*, vol. abs/1911.10074, Dec 2019, pp. 1–8, 1911.10074.

[51] McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; Wilkins, D. “PDDL − The Planning Domain Definition Language”, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS)*, vol. 4, Oct 1998, pp. 1–26.

[52] Meneguzzi, F.; Oh, J. “Proactive Assistant Agents: Papers from the AAAI Fall Symposium”, Technical Report, Proactive Assistant Agents, Arlington, Virginia, 2010, 58p.

[53] Meneguzzi, F.; Pereira, A.; Pereira, R. “An LP-Based Approach for Goal Recognition as Planning”. In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2021, pp. 11939–11946.

[54] Min, W.; Ha, E.; Rowe, J. P.; Mott, B. W.; Lester, J. C. “Deep learning-based goal recognition in open-ended digital games”. In: Proceedings of the Association for the Advancement of Artificial Intelligence (AIIDE), 2014, pp. 37–43.

[55] Mirsky, R.; Gal, Y. K.; Shieber, S. M. “CRADLE: An Online Plan Recognition Algorithm for Exploratory Domains”, *ACM Transactions on Intelligent Systems and Technology*, vol. 8–3, Apr 2017, pp. 45:1–45:22.

[56] Mirsky, R.; Gal, Y. K.; Tolpin, D. “Session analysis using plan recognition”. In: Proocedings of the Workshop on User Interfaces and Scheduling and Planning at the International Conference on Automated Planning and Scheduling (ICAPS), 2017, pp. 1–7.

[57] Mirsky, R.; Stern, R.; Gal, K.; Kalech, M. “Sequential plan recognition: An iterative approach to disambiguating between hypotheses”, *Artificial Intelligence*, vol. 260, Jul 2018, pp. 51–73.

[58] Mirsky, R.; Stern, R.; Gal, Y. K.; Kalech, M. “Sequential Plan Recognition”. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2016, pp. 401–407.

[59] Mitchell, T. M. “Machine Learning”. New York, NY, USA: McGraw-Hill, Inc., 1997, 1 ed., 421p.

[60] Mohri, M.; Rostamizadeh, A.; Talwalkar, A. “Foundations of Machine Learning”. Cambridge, MA, USA: The MIT Press, 2012, 427p.

[61] Oh, J.; Meneguzzi, F.; Sycara, K. “Probabilistic plan recognition for intelligent information agents: Towards proactive software assistant agents”. In: Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART), 2011, pp. 281–287.

[62] Oh, J.; Meneguzzi, F.; Sycara, K. P.; Norman, T. J. "An Agent Architecture for Prognostic Reasoning Assistance". In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2011, pp. 2513–2518.

[63] Pereira, R. F.; Meneguzzi, F. "Landmark-based Plan Recognition". In: Proceedings of the European Conference on Artificial Intelligence (ECAI), 2016, pp. 1706–1707.

[64] Pereira, R. F.; Meneguzzi, F. "Goal and plan recognition datasets using classical planning domains", Technical Report, PUCRS, Zenodo, 2017, 2p.

[65] Pereira, R. F.; Oren, N.; Meneguzzi, F. "Landmark-Based Heuristics for Goal Recognition". In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2017, pp. 3622–3628.

[66] Pereira, R. F.; Oren, N.; Meneguzzi, F. "Monitoring plan optimality using landmarks and domain-independent heuristics". In: Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR), 2017, pp. 867–873.

[67] Pereira, R. F.; Oren, N.; Meneguzzi, F. "Landmark-based approaches for goal recognition as planning", *Artificial Intelligence*, vol. 279, Feb 2020, pp. 103217.

[68] Pereira, R. F.; Vered, M.; Meneguzzi, F.; Ramirez, M. "Online probabilistic goal recognition over nominal models". In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2019, pp. 5547–5553.

[69] Pozanco, A.; E-Martín, Y.; Fernández, S.; Borrajo, D. "Counterplanning using goal recognition and landmarks". In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2018, pp. 4808–4814.

[70] Pynadath, D. V.; Wellman, M. P. "Accounting for Context in Plan Recognition, with Application to Traffic Monitoring", *Computing Research Repository (CoRR)*, vol. abs/1302.4980, Aug 2013, pp. 472–481.

[71] Ramírez, M.; Geffner, H. "Plan recognition as planning". In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 1778–1783.

[72] Ramírez, M.; Geffner, H. "Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners". In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2010, pp. 1121–1126.

[73] Richter, S.; Helmert, M.; Westphal, M. "Landmarks Revisited". In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2008, pp. 975–982.

[74] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. "Neurocomputing: Foundations of research". In: *Neurocomputing – Foundations of Research: Volume 1*, Anderson, J. A.; Rosenfeld, E. (Editors), Cambridge, MA, USA: The MIT Press, 1988, chap. Learning Representations by Back-propagating Errors, pp. 696–699.

[75] Russell, S.; Norvig, P. "Artificial intelligence: A Modern Approach". Hoboken, NJ, USA: Prentice Hall, 2010, 3 ed., 1132p.

[76] Say, B.; Wu, G.; Zhou, Y. Q.; Sanner, S. "Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming". In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2017, pp. 750–756.

[77] Schlimmer, J. C.; Granger, R. H. "Incremental learning from noisy data", *Machine Learning*, vol. 1–3, March 1986, pp. 317–354.

[78] Schmidt, C. F.; Sridharan, N. S.; Goodson, J. L. "The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence", *Journal of Artificial Intelligence Research*, vol. 11–1-2, May 1978, pp. 45–83.

[79] Shvo, M.; Klassen, T. Q.; Sohrabi, S.; McIlraith, S. A. "Epistemic plan recognition". In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2020, pp. 1251–1259.

[80] Shvo, M.; McIlraith, S. A. "Active goal recognition". In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2020, pp. 9957–9966,.

[81] Shvo, M.; Sohrabi, S.; McIlraith, S. A. "An AI planning-based approach to the multi-agent plan recognition problem". In: Proceedings of the Canadian Conference on Artificial Intelligence (Canadian AI), 2018, pp. 253–258).

[82] Sohrabi, S.; Riabov, A.; Katz, M.; Udrea, O. "An AI Planning Solution to Scenario Generation for Enterprise Risk Management". In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2018, pp. 160–167.

[83] Sohrabi, S.; Riabov, A. V.; Udrea, O. "Plan Recognition as Planning Revisited." In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2016, pp. 3258–3264.

[84] Sohrabi, S.; Riabov, A. V.; Udrea, O. "Plan Recognition as Planning Revisited". In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2016, pp. 3258–3264.

[85] Suárez-Hernández, A.; Aguas, J. S.; Torras, C.; Alenyà, G. "STRIPS action discovery", *CoRR*, vol. abs/2001.11457, Feb 2020, pp. 1–8, 2001.11457.

[86] Sukthankar, G.; Goldman, R. P.; Geib, C.; Pynadath, D. V.; Bui, H. H. "Plan, Activity, and Intent Recognition: Theory and Practice". Alpharetta, GA, USA: Elsevier, 2014, 424p.

[87] Sundermeyer, M.; Ney, H.; Schlüter, R. "From feedforward to recurrent lstm neural networks for language modeling", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23–3, March 2015, pp. 517–529.

[88] Vered, M.; Kaminka, G. A.; Biham, S. "Online goal recognition through mirroring: Humans and agents". In: Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems (ACS), 2016, pp. 2112–2114.

[89] Vered, M.; Pereira, R. F.; Magnaguagno, M.; Kaminka, G. A.; Meneguzzi, F. "Towards online goal recognition combining goal mirroring and landmarks (extended abstract)". In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2018, pp. 2112–2114.

[90] Vidal, V.; Geffner, H. "Solving Simple Planning Problems with More Inference and No Search". In: Proceedings of the Conference on Principles and Practice of Constraint Programming (CP), 2005, pp. 682–696.

[91] Vincent, P.; Larochelle, H.; Bengio, Y.; Manzagol, P.-A. "Extracting and composing robust features with denoising autoencoders." In: Proceedings of the International Conference on Machine Learning (ICML), 2008, pp. 1096–1103.

[92] Wu, G.; Say, B.; Sanner, S. "Scalable planning with tensorflow for hybrid nonlinear domains". In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2017, pp. 2378–2388.

[93] Yang, Q.; Wu, K.; Jiang, Y. "Learning actions models from plan examples with incomplete knowledge." In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2005, pp. 241–250.

[94] Zhang, J.; Zong, C. "Deep neural networks in machine translation: An overview", *IEEE Intelligent Systems*, vol. 30–5, Sep 2015, pp. 16–25.

[95] Zheng, G.; Mukherjee, S.; Dong, X. L.; Li, F. "Opentag: Open attribute value extraction from product profiles". In: Proceedings of the International Conference on Knowledge Discovery & Data Mining (SIGKDD), 2018, pp. 1049–1058.

[96] Zhuo, H. H.; Zha, Y.; Kambhampati, S.; Tian, X. "Discovering underlying plans based on shallow models", *ACM Transactions on Intelligent Systems and Technology*, vol. 11–2, Apr 2020, pp. 1–30.

APPENDIX A – INFERRED PDDL

This appendix shows examples of computed PDDL elements by our approach of Chapter 3, for readers curious about the result of our inferred PDDL files. First, we show an example of computed action. Second, we show an example of a problem file generated by our approach.

Grounded PDDL Actions

Here, we generate the PDDL for the MNIST 8-Puzzle domain. We use the auto-encoder developed by Asai and Fukunaga [10], with a latent layer of 36 bits. These 36 bits are used as the grounded predicates used to describe the PDDL domain. Example APPENDIX A.1, we show the header of the PDDL file of the inferred MNIST 8-Puzzle domain. As we can see, we have a predicate ($p(n)$) for every bit of the latent representation, starting from $p(0)$. Note that *negative-preconditions* are required.

Example APPENDIX A.1 *The header of the computed domain file of the MNIST 8-Puzzle domain.*

```
(define (domain generated-domain)
    (:requirements :strips :negative-preconditions)
    (:predicates
        (p0)
        (p1)
        (p2)
        (p3)
        (p4)
        (p5)
        (p6)
        (p7)
        (p8)
        (p9)
        (p10)
        (p11)
        (p12)
        (p13)
        (p14)
        (p15)
        (p16)
        (p17)
```

```
        (p18)
        (p19)
        (p20)
        (p21)
        (p22)
        (p23)
        (p24)
        (p25)
        (p26)
        (p27)
        (p28)
        (p29)
        (p30)
        (p31)
        (p32)
        (p33)
        (p34)
        (p35)
    )
    <...>
)
```

In Example APPENDIX A.2, we show a computed PDDL action $a1$ for the MNIST 8-Puzzle domain. Note that there are no parameters, as the predicates are all grounded. In this specific action, we have a precondition for every single predicate of the planning domain. In Section 3.2, we explain that our inferred actions are based on transition between two states (s_1 and s_2) converted to latent space. The transitions that have the same effect are grouped, and the precondition is the intersection of bits that are equal between all transitions. Since $a1$ has a precondition for all predicates, we can confirm that there was only one transition in the dataset with this specific effect.

Example APPENDIX A.2 *The first computed action using Algorithm 1 for the MNIST 8-Puzzle domain.*

```
(:action a1
    :parameters ()
    :precondition (and
        (not (p0))
        (p1)
        (p2)
```

```
        (not (p3))
        (p4)
        (p5)
        (not (p6))
        (not (p7))
        (p8)
        (p9)
        (not (p10))
        (p11)
        (p12)
        (p13)
        (not (p14))
        (p15)
        (p16)
        (not (p17))
        (p18)
        (not (p19))
        (not (p20))
        (p21)
        (not (p22))
        (p23)
        (p24)
        (p25)
        (p26)
        (not (p27))
        (not (p28))
        (not (p29))
        (not (p30))
        (not (p31))
        (p32)
        (not (p33))
        (not (p34))
        (not (p35))
    )
    :effect(and
        (p0)
        (p10)
        (not (p11))
        (not (p13))
        (not (p16))
```

```
        (p22)
    )
)
```

In Example APPENDIX A.3 we show a different action $a4$ for the MNIST 8-puzzle domain. This action has a precondition for only ten (10), differently from the previous action we detailed. This occurs because these ten predicates are the only bits reoccurring in every transition with this specific effect. This means that inferred actions using multiple transitions tend to be much less restrictive than actions inferred with fewer transitions and are probably much closer to this specific action's actual precondition.

Example APPENDIX A.3 *The fourth computed action using Algorithm 1 for the MNIST 8-Puzzle domain.*

```
(:action a4
    :parameters ()
    :precondition (and
        (not (p0))
        (p3)
        (not (p5))
        (p6)
        (not (p8))
        (not (p12))
        (p15)
        (p16)
        (p26)
        (not (p30))
    )
    :effect(and
        (p0)
        (not (p3))
        (p5)
        (not (p6))
        (p8)
        (p12)
        (not (p15))
        (not (p16))
        (not (p26))
        (p30)
    )
```

```
)
```

Grounded Recognition Problem

Besides the PDDL domain, we must compute a problem file for the recognition problem. The problem file includes the initial state of the problem. The goal hypotheses are included in a separate file, so the goal in the problem file is not included, only a placeholder written as "<HYPOTHESIS>". This is the standard used in most recognizers [64, 71, 72, 84]. In example APPENDIX A.4, we show a problem file for the MNIST 8-Puzzle domain.

Example APPENDIX A.4 *A computed recognition problem for the MNIST 8-Puzzle domain.*

```
(define (problem pb1)
    (:domain generated-domain)
    (:init
        (p6)
        (p7)
        (p9)
        (p15)
        (p18)
        (p19)
        (p21)
        (p22)
        (p23)
        (p25)
        (p26)
        (p27)
        (p28)
        (p29)
        (p32)
        (p34)
    )
    (:goal
        (and
            <HYPOTHESIS>
        )
    )
)
```

APPENDIX B – LATREC DEMO

In this appendix, we detail an interactive demonstration we presented in a conference of a variation of our approach to recognizing goals in latent space (Chapter 3) LSTMs to perform the recognition task. This appendix is for the readers curious about how we can use this approach in real-world scenarios with online goal recognition. First, we briefly describe our approach. Second, we detail how the demonstration is set up and how users can interact with it.

Data-Driven Goal Recognition in Latent Space

As detailed in Chapter 3, recognizing goals in latent space consists of 4 steps: 1) converting real-world data to latent space; 2) acquiring traces of executions in latent space; 3) predicting the correct goal through a recognizer; 4) converting the goal in latent space to real-world data.

We use the same the auto-encoder used in Chapter 3, developed by Asai and Fukunaga [10]. Images processed through the encoder become bit-vectors that provide a binary representation for states comprising propositional attributes. Sequences of such propositional states implicitly represent sequences of action executions, which, in turn, generate traces to feed a recognizer.

After converting real-world data and acquiring traces, we can train a recognizer. In Section 3.5, we aimed to create a learning model capable of computing the intended goal of an agent given a trace of states, reconstructing the correct goal in latent space. This approach was capable of solving goal recognition problems without having candidate goals, resulting in no spread in the recognized goals. However, since this solved the goal recognition problem as a classification problem, it was incapable of predicting goals that were not in the training data. We develop a learning approach that leverages from the information of having defined goal hypotheses.

Our approach focuses on learning the probability of each predicate of the goal being true instead of faithfully reconstructing the goal state. Thus, this approach receives a set of observed states and computes each predicate's probability in the domain, being true or false, given the observed data. Using these probabilities, we assume that we have a standard goal recognition problem, where a set $\mathcal{G}$ of candidate goals are given, and each goal of such subset is a set of n predicates. Thus, we select the most likely goal using each predicate's learned probabilities as the correct goal that the agent intends to achieve. This process is shown in Figure APPENDIX B.1, where each output node of the LSTM is the probability of such predicate being true, and the G4 is the most likely goal.

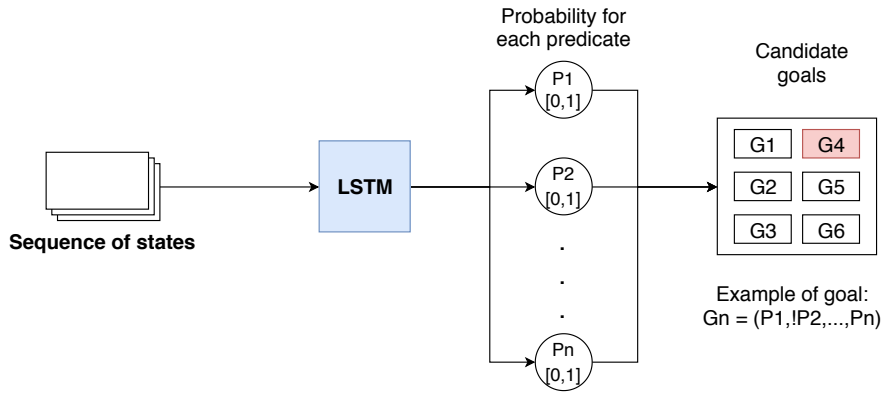


Figure APPENDIX B.1: Data-driven model for goal recognition

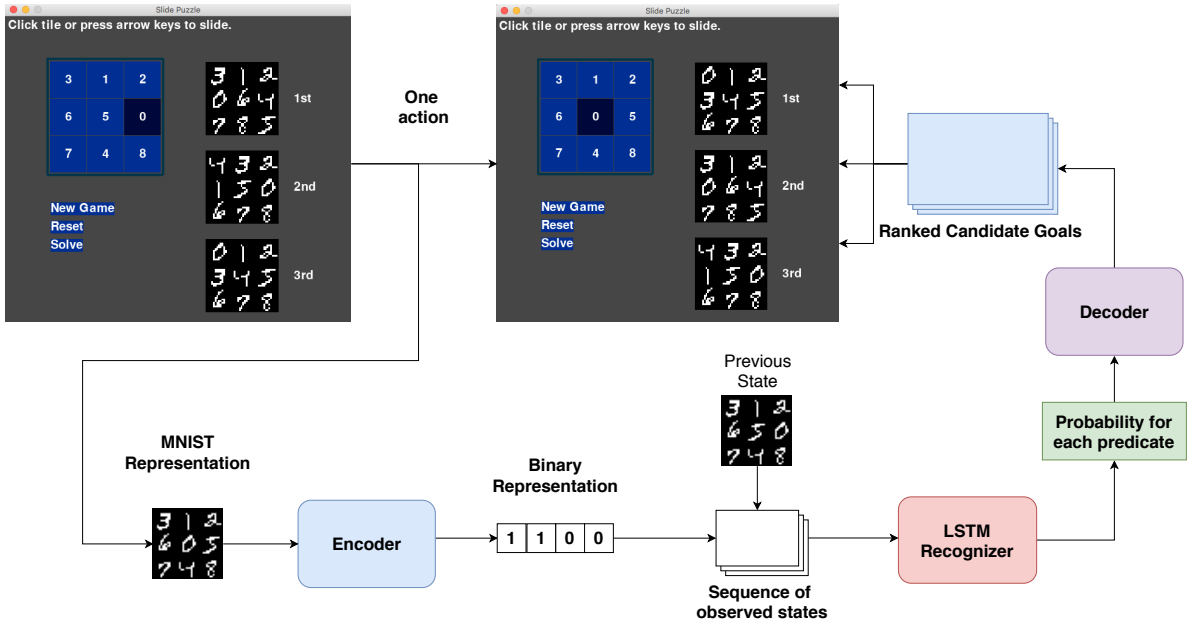


Figure APPENDIX B.2: Application screen showing the transition of two states and the candidate goals for the current state.

Application Overview

LatRec is a desktop application implementing our approach for recognizing goals in latent space. The user can start a "New Game", "Reset" the current game, or "Solve" the puzzle, where all steps to solve the puzzle is displayed. The user should solve the puzzle by playing the board with blue tiles, where the navy tile containing "0" indicates the tile that the user is allowed to move. The system outputs the most likely goals that the user is trying to achieve on the right side. Figure APPENDIX B.2 show the workflow of LatRec. In this case, the user acted swapping the tile "0" with the tile "5". The image representation of the board is fed to an encoder, which generates a binary representation. This binary representation is fed to the learned model, along with the last nine (9) observed encoded states. The learned model returns a probability for each predicate, which we compare with the candidate goals available. We then rank the candidate goals and finally decode them, displaying them to the user.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br