32nd International Conference on Automated Planning and Scheduling

June 13–24, 2022, virtually from Singapore

DC 2022

ICAPS Doctoral Consortium 2022

Participants

Carlos Núñez-Molina	Universidad de Granada
Devin Thomas	University of New Hampshire
Eyal Weiss	Bar-Ilan University
Johannes Schmalz	The Australian National University
Kristýna Pantůčková	Charles University
Marcel Vinzent	Saarland University
Maxence Grand	University of Grenoble Alpes
Naman Shah	Arizona State University
Pulkit Verma	Arizona State University
Songtuan Lin	The Australian National University
Thorsten Klößner	Saarland University
Ursula Addison	City University of New York
Xiaodi Zhang	The Australian National University

Mentors

Alessandro Cimatti	Center for Digital Industry, Fondazione Bruno Kessler
Christopher Beck	University of Toronto
David Smith	Consultant
Eva Onaindia	Polytechnic University of Valencia
George Konidaris	Brown University
Hector Geffner	Universitat Pompeu Fabra
Laura Hiatt	Naval Research Laboratory
checked from here Malte Helmert	University of Basel
Masataro Asai	MIT-IBM Watson AI Lab / IBM Research Cambridge
Mauro Vallati	University of Huddersfield
Sarah Keren	Technion – Israel Institute of Technology
Subbarao (Rao) Kambhampati	Arizona State University

Organizers

Pascal Bercher	The Australian National University
Sara Bernardini	Royal Holloway, University of London

Preface

The ICAPS Doctoral Consortium (DC) is intended to provide PhD students with the opportunity to interact closely with established researchers and get feedback on their research and advice on career possibilities and build a professional network, improving the cohesion of new researchers within the ICAPS community.

At ICAPS 2022, we received 13 submissions from students across Europe, Israel, North America, and Australia. Each student submitted a dissertation abstract, a list of preferred mentors, a CV, and a supervisor statement via EasyChair. We admitted all students.

We implemented a mentoring program by which almost all students could be paired with their first mentor choice. The students and the mentors met online before the start of the DC Program.

The papers covered a variety of topics, including Neurosymbolic AI, Deep Learning, Reinforcement Learning, Neural Networks, Action Policies, Verification, Predicate Abstraction, Long-Term Autonomy, Goal Formulation, Goal Management, Probabilistic Planning, Replanning, Linear Programming, HTN Planning, Plan Recognition, Conformant Planning, Model Reconciliation, Markov Decision Process, Stochastic Shortest Path Problems, Heuristic Search, Abstraction Heuristics, Integrated Task and Motion Planning, Situated Planning, Metareasoning, Interpretability, Classical Planning and Search.

The DC program took place on July 20th, from 5 pm UCT to 0:15 am UCT online of gather.town and Zoom since ICAPS 2022 was a virtual conference. It was particularly challenging to find a time and schedule that works for all participants as their time zones span from the US over Europe to Australia. Despite this challenge, all the PhD students participated in the DC.

The PhD students pre-recorded short videos with a presentation of their papers, which was made available online on YouTube prior to the start of the conference. During the DC Program, the students presented their posters. The poster presentation was arranged in two sessions to allow the students to see each other's work.

Wheeler Ruml, Professor in the Department of Computer Science at the University of New Hampshire, gave an invited talk titled "*Why Are You Doing This? Advice on Writing Papers and Giving Talks*", which was well attended by the students and the community.

Finally, the DC Program included social events at the start and end of the technical program. The students engaged in two "speed-dating" sessions (called speed.gathering) during which they had an opportunity to get to know each other.

Additional information regarding the DC can be found on the following page:
http://icaps22.icaps-conference.org/dc-2022

Pascal and Sara,
ICAPS DC 2022 Organizers,
July 2022

Invited Talk

We were more than happy to announce a very exciting speaker!

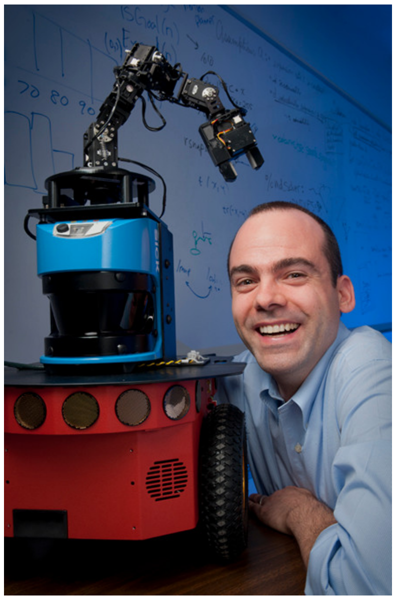
Wheeler Ruml gave an invited talk on:

Why Are You Doing This? Advice on Writing Papers and Giving Talks

Science is a grand conversation that spans the globe and continues over generations. Writing a paper is your chance to cheat death and join in this noble endeavor. Just as you formulate your thoughts before speaking during a conversation, you should understand what you are trying to say with your paper, even before you start the research. Similarly, giving a talk is your chance to share with others the good news that you've discovered. But don't overestimate the listener - they are probably jet-lagged or distracted by a child screaming in the other room. This talk will be full of my personal opinions and advice - I hope it is helpful to you!

Bio

Wheeler Ruml, Professor in the Department of Computer Science at the University of New Hampshire



Wheeler Ruml is a Professor of Computer Science at the University of New Hampshire (UNH). He was General Co-Chair of ICAPS-14, Treasurer for the ICAPS executive council, and a co-founder and President of SoCS. Before joining UNH, he led a team at Xerox's PARC lab that used AI planning techniques to build the world's fastest printer. He enjoys trying to decide which node to expand.

Table of Contents

A Generalization of Automated Planning Using Dynamically Estimated Action Models – Dissertation Abstract

Eyal Weiss
The MAVERICK Group, Bar-Ilan University, Israel
Supervisor: Prof. Gal A. Kaminka
{weissey, galk}@cs.biu.ac.il

Abstract

Representing real-world planning problems is a major open subject. Standard planning modeling languages are fully declarative, making it challenging to use them for expressing complex mathematical functions, that are often required for describing the effects of actions. Recent approaches turn to external sources of information, such as simulators or black-box modules, to overcome such modeling limitations. This paper proposes a novel approach to represent and solve planning problems, by starting with partial declarative action models and incrementally refining them during planning by invoking domain-specific external modules. Since these might be computationally expensive, we provide the planner the ability to trade-off modeling uncertainty against computation time, to meet target plan accuracy. Results that were obtained for planning with dynamic estimation of action costs are sketched, and planned work, together with open challenges, are further detailed.

Introduction

AI planning is a mature research field that has undergone major developments over the years. While its roots built on purely symbolic and highly abstract problem formulation (Fikes and Nilsson 1971), it has gradually evolved to support richer declarative representations by using more detail, which is evident, e.g., in the various PDDL versions introduced (McDermott et al. 1998), cf. (Fox and Long 2003). In addition, current planning technology, that is tailored to solve problems expressed in terms of standard formulations (such as PDDL), is based on solid theory, sophisticated search algorithms (Lipovetzky and Geffner 2017), efficient domain-independent heuristics (see the description of many heuristics in the book (Ghallab, Nau, and Traverso 2016)) and data-dependent planner portfolios (Gerevini, Saetti, and Vallati 2014), where much of this work has been translated to optimized open-source software implementations (Helmert 2006). However, it is widely accepted that the adoption of AI planning technology outside the research community is not very common, which stands in sharp contrast to its maturity.

We believe that simplistic modeling is one of the key factors that inhibit widespread use. In particular, we claim that many real-world planning problems cannot be adequately represented solely using current modeling languages, and therefore they are not applicable to existing domain-independent planning technology. This argument has been raised before at various times and contexts (McCluskey 2003; Boddy 2003; Rintanen 2015), yet the solution that was consistently suggested—to make modeling languages more expressive—seems unlikely to be sufficient on its own. This is because the effects of some actions may only be described using complex mathematical functions, or even only known in black-box form. Hence, *it is our belief that as long as exclusively fully declarative models are used, there will still be planning problems out of reach.*

A recent trend advocates coupling of external sources of information to the planner, in order to overcome modeling limitations. Presently, there are two major lines of research taking this approach: planning with simulators (Francès et al. 2017), and domain-specific attempts—notably within the framework of Task and Motion Planning (a recent review is provided in (Garrett et al. 2021)). While these may well be appropriate for some applications, they do not offer a full solution to the gap in problem modeling. Indeed, the first relies on simulators that are not always available, and furthermore, it sacrifices much mathematical structure—inherent in declarative action models—rendering many known heuristics inapplicable, while the second is, as mentioned, domain-specific, and thus does not offer a high level of generality.

Motivated by the gap suggested, and the opportunity it presents, we focus our efforts on the following question.

Research Question *How can we leverage state-of-the-art domain-independent planning technology to tackle real-world problems that cannot be adequately represented using purely declarative models?*

Breaking the Barrier between Problem Modeling and Planning

Our proposal is to postpone part of the modeling to the planning phase, and to utilize external sources of information for model completion ad hoc. Since calling external modules during planning can be computationally expensive (similar to using heuristic functions), it is advantageous to provide the planner with the ability to make educated choices, to balance computation time against allowed uncertainty. Thus, *our vision is to start with a partial declarative model, and to incrementally refine it during planning only where it appears necessary for finding a plan that meets a target accuracy.*

While this idea is fairly high-level, we offer one concrete implementation based on the following specifications:

- Keep problem *structure* symbolic and abstract, by using declarative action models that initially only specify structural preconditions and effects (e.g. via predicates).
- Acquire *numeric* model parameters online, by letting the planner call domain-specific external modules (i.e., estimators) that provide information about their values.
- Define an acceptable accuracy for the sought-after plan, and allow the planner to control the model uncertainty, so it can trade-off accuracy vs. computation time.

Consequences The immediate implication of the suggested approach is an enhanced ability to represent and solve planning problems, as clearly every declarative representation (that so far was constructed prior to planning) can be completed incrementally by the planner, given appropriate external modules. The price paid is increased planning time, due to additional computational effort spent on refining the model. This trade-off is typical for problem generalization, as it entails solving a harder problem. The main challenge that arises is thus to develop computationally efficient planners, able to balance resource allocation between search effort and model refinement effort.

We believe that this suggestion provides several appealing properties. First, any kind of estimator can be used, so there are no restrictions on the type of data being processed during planning, nor on the mathematical operations it utilizes, and in particular it can be black-box. Second, state-of-the-art domain-independent planning techniques retain relevance, as the only difference in the problem formulation is the need to dynamically acquire numeric model parameters. Namely, exiting heuristics can still get the information they need to work, where the sole modification is that they take as inputs estimations of—instead of exact—numeric parameters. More broadly, *current domain-independent planners need to be extended, rather than replaced, in order to be applicable*. Lastly, model uncertainty can be systematically controlled to meet target plan accuracy, while offering significant potential savings on redundant modeling time. This might seem somewhat unusual, as modeling time is not typically a factor considered from the planning perspective, yet richer representations of planning problems could become prohibitive if fully compiled prior to planning. Indeed, consider the implications on the time required to construct a model, in case an estimator is applied for every ground action prior to planning. This is similar to applying numerous heuristics prior to the planning phase, which is clearly a waste of resources.

PhD Research Goals We have set two goals for the PhD period that follow our proposal into concrete setups. The first is to develop a framework that supports dynamic estimation of action costs, and the second is to develop an analogous framework for dynamic estimation of action effect probabilities. Achieving these goals require appropriate problem formulations, algorithms, software implementation and finally empirical validation. We highlight that the first line of work falls into the category of deterministic planning, where the second belongs to probabilistic planning. Hence, the expectation is that pursuing each goal will require a different toolset, yet the similarities can help carry lessons learned from one line of work to the other.

Lastly, we wish to clarify a distinction between our intended setup for probabilistic planning, and standard Markov Decision Process (MDP) with unknown probabilities. While the latter is typically approached via Reinforcement Learning, so that an agent seeks to find a policy by trial and error (and in particular, *by acting*), our setup focuses on *pure planning*, where probabilities can be gradually estimated by calling appropriate estimators.

Research Status

We first briefly describe some of our achievements so far, and then continue to detail what is planned next. We note that most of the research that was carried out relates to cost estimation, where probability estimation is largely left for future work. In addition, since the results obtained for the latter have not yet passed external inspection, we do not present them here.

Dynamic Action Cost Estimation

Our framework employs the basic assumption that every ground action can potentially have multiple cost estimators, with varying degrees of accuracy and different running times. In particular we assume that once called, each estimator returns lower and upper bounds for the true action cost. Note that this does not prevent knowledge of exact costs (where the bounds are simply equal), nor the usage of bound priors, that can be specified in the initial problem model (these can be thought of as estimators that have fast $O(1)$ run time). It is worth mentioning that an anytime algorithm that serves as a cost estimator can in fact represent different estimators, where each of them is just an invocation of the same one but provided different running times.

Relying on this assumption, we then define a deterministic planning problem where the goal of the planner is to find a plan that meets a target sub-optimality multiplier as fast as possible. I.e., it aims to efficiently find a plan π^ϵ that satisfies

$$c(\pi^\epsilon) \leq c^* \times \epsilon,$$

with c^* being the optimal cost and $\epsilon \geq 1$. We proved that an algorithm which utilizes lower and upper bounds of costs, instead of exact values, can solve this problem by relying on the ratio of the accumulated bounds for the action costs composing the plan.

This lead to the development of ASEC (which stands for A^* with Synchronous Estimations of Costs) that implements this idea. ASEC serves as our principal algorithm for solving such problem instances, and we have been able to prove that it is sound, and incomplete in general, but is complete under special circumstances. Next, we developed a post-search procedure and an iterative framework, which both build on ASEC to obtain improved results. We further showed that applying a particular strategy for using ASEC within the iterative framework renders the resulting algorithm complete.

We implemented ASEC and its extensions by modifying and extending Fast Downward, and then empirically tested its performance on problems generated from planning competition benchmarks, which were added synthetic estimators. Our findings provide strong empirical evidence that ASEC outperforms alternatives w.r.t. run time, while typically meeting the target bound. These results, along with detailed analysis and another variant of ASEC, are summarized in a paper that is currently under submission process.

Planned Work We have three more objectives that we plan to pursue: 1. In the near future we intend to test various strategies for using ASEC within the iterative framework, since we have reason to believe that applying a data-dependent approach could yield run time improvements (at least in some cases). 2. The empirical results we obtained suggest that supporting a cache mechanism (for the estimated values) could provide considerable savings. Furthermore, it appears to make it simpler to develop an asynchronous version for ASEC, which might be more efficient. Hence, we plan to put it to test. 3. Lastly, we are considering to make stronger assumptions by adding meta-information about the estimators (such as expected run time), so that the planner could make more educated choices.

Challenges and Future Work

We suggest several interesting possibilities for future research. First, our work is clearly just the first offspring, and there may exist better algorithms to be discovered that solve the problems we suggested. Second, model uncertainty can be quantified using various statistical measures, leading to divergent problem setups, e.g., utilizing the Probably Approximately Correct (PAC) framework, or using standard deviations for setting a target bound on the plan cost. Third, a considerable challenge that arises from our proposed research is to connect actual external computational modules to the planner. Namely, in order to test the suggested ideas on real-world examples, one has to embark on a significant software development project, as each domain has its own relevant estimators and their unique APIs. This also makes it harder to compare different algorithms, as synthetic data (generated by synthetic estimators) might fail to reveal practical pain points. On the other hand, we believe this also presents an opportunity to increase the exposure of existing planning tools outside the research community.

References

Boddy, M. S. 2003. Imperfect match: PDDL 2.1 and real applications. *Journal of Artificial Intelligence Research*, 20: 133–137.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4): 189–208.

Fox, M.; and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20: 61–124.

Francès, G.; Ramiŕez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Descriptions are Overrated: Classical Planning with Simulators. In *IJCAI*.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4: 265–293.

Gerevini, A.; Saetti, A.; and Vallati, M. 2014. Planning through automatic portfolio configuration: The pbp approach. *Journal of Artificial Intelligence Research*, 50: 639–696.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Lipovetzky, N.; and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

McCluskey, T. L. 2003. PDDL: A Language with a Purpose?

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Rintanen, J. 2015. Impact of modeling languages on the theory and practice in planning research. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Action Model Learning based on Grammar Induction – Dissertation Abstract

Maxence Grand
Under the Supervision of Damien Pellier and Humbert Fiorino
Univ. Grenoble Alpes, LIG
3800 Grenoble, France
{Maxence.Grand, Damien.Pellier, Humbert.Fiorino}@univ-grenoble-alpes.fr

Abstract

This paper presents a novel approach to learn PDDL domain called AMLSI (*Action Model Learning with State machine Interaction*) based on grammar induction. AMLSI learns with no prior knowledge from a training dataset made up of action sequences built by random walks and by observing state transitions. The domain learnt is accurate enough to be used without human proofreading in a planner even with very highly partial and noisy observations. Thus AMLSI takles a key issue for domain learning that is the ability to plan with the learned domains. It often happens that small learning errors lead to domains that are unusable for planning. AMLSI contribution is to learn domains from partial and noisy observations with sufficient accuracy to allow planners to solve new problems. Also, this paper presents an incremental and a temporal extension.

Introduction

Hand-coding Planning domains is generally considered difficult, tedious and error-prone. The reason is that experts in charge modelling domains are not always PDDL experts and vice versa. To overcome this issue, two main approaches have been proposed. One is to develop knowledge engineering tools facilitating PDDL writing. These tools provide support for consistency and syntactic error checking, domain visualisation etc. An inconvenient aspect of these tools is that they require PDDL expertise and background in software engineering (Shah et al. 2013).

The other approach is to develop machine learning algorithms to automatically generate Planning domains as, for instance, ARMS (Yang, Wu, and Jiang 2007), SLAF (Shahaf and Amir 2006), LSONIO (Mourão et al. 2012), LOCM (Cresswell, McCluskey, and West 2013). These algorithms use training datasets made of possibly noisy and partial state/action sequences generated by a planner, or randomly generated. Classically, IPC benchmarks are used to generate training datasets. The performance of these algorithms is then measured as the syntactical distance between the learned domains and the IPC benchmarks. Machine learning approaches have three main drawbacks.

First, most of these approaches require a lot of data to perform the learning of Planning domains and in many real world applications, acquiring training datasets is difficult and costly, e.g., Mars Exploration Rover operations (Bresina

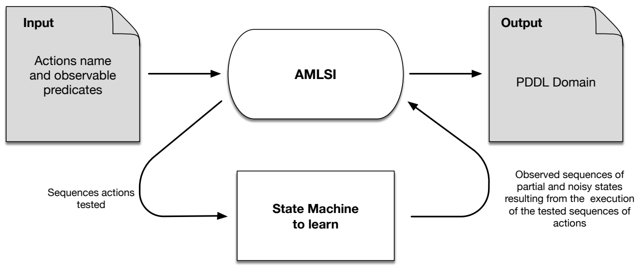

Input
Actions name and observable predicates
AMLSI
Output
PDDL Domain
Sequences actions tested
State Machine to learn
Observed sequences of partial and noisy states resulting from the execution of the tested sequences of actions

Figure 1: AMLSI: Action Model Learning with State machine Interaction

et al. 2005). Also, dataset acquisition is a long term evolutive process: in real world applications, training data become available gradually over time. Moreover, in practice, it is important to be able to update learned PDDL domains to new incoming data without restarting the learning process from scratch.

Second, the learned domains are not *accurate* enough to be used ”as is” in a planner: a step of expert proofreading is still necessary to correct them. Even small syntactical errors can make sometime the learned domains useless for planning. Therefore, we consider that domain *accuracy*, that we define as the capacity of a learned domain to solve planning problems that were not used in the training dataset, is a better performance indicator than syntactical distance in practice.

Third, even if some approaches, e.g., (Mourão et al. 2012; Segura-Muros, Pérez, and Fernández-Olivares 2018; Rodrigues, Gérard, and Rouveirol 2010) are able to learn from noisy and/or partially observable data, few approaches are able to handle very high levels of noise and partial observations as can be encountered in real world applications, especially in robotics where observations are extracted using miscalibrated or noisy sensors.

To address these challenges, we propose a novel Planning domain learning algorithm called AMLSI (*Action Model Learning with State machine Interaction*). A key idea in AMLSI is that real world can be modelled as state machines and that retro-engineering real world state machines is analogous to learning a Deterministic Finite Automaton (DFA), which is equivalent to a regular grammar: we argue that (1) it is possible to learn a DFA by querying a real world state machine (see Figure 1), and (2) to induce a PDDL domain from this regular grammar. AMLSI does not require any prior knowledge regarding the feasibility of actions in a given state, and state observations can be partial and noisy. AMLSI is highly accurate even with highly partial and noisy state observations. Thus it minimizes PDDL proofreading and correction for domain experts. Our experiments show that in many cases AMLSI does not even require any correction of the learned domains. AMLSI is lean and efficient on data consumption. It uses a supervised learning approach based on grammar induction. Training data are action/state (possibly partial and noisy) sequences labeled as feasible or infeasible. Both, feasible and infeasible action/state sequences are used by AMLSI to learn PDDL domains, thus maximizing data usability.

The rest of the paper is organized as follows. First of all, we propose some related works on PDDL domain learning. Then we present our contributions and finally we give our Future works.

Related Works

Many approaches have been proposed to learn Planning domains (see Table - 1). These approaches can be classified according to the input data of the learning process. The input data can be plan "traces" obtained by resolving a set of planning problems, annoted plans, decomposition tree or random walks. The input data can contain in addition to the tasks also states which can be fully observable (FO), partially observable (PO), no observable (NO), or noisy. Also, these approaches can be classified according to the output. The out can be PDDL domains and more precisely STRIPS and/or ADL domains, Temporal domains and HTN domains. In the rest of this section, we classify approaches according to the output.

PDDL Learning

A first group of approaches takes as input a set of plan traces and a partial domain, and tries to incrementally refine this domain to complete it, as for instance, OpMaker (McCluskey, Richardson, and Simpson 2002) or RIM (Zhuo, Nguyen, and Kambhampati 2013). In all of these approaches, it is assumed that the observations are complete and noiseless except OPMaker that induces operators with user interactions by using the GIPO tools (Simpson, Kitchin, and McCluskey 2007). A second group of approaches only takes plan traces as input. Most of them deals with partial observations. Among these approaches are ARMS (Yang, Wu, and Jiang 2007), LAMP (Zhuo et al. 2010), Louga (Kucera and Barták 2018), Plan-Milner algorithm (Segura-Muros, Pérez, and Fernández-Olivares 2018), AIA (Verma, Marpally, and Srivastava 2021) or FAMA (Aineto, Celorrio, and Onaindia 2019). Among these works, the ARMS systems is the most known. It gathers knowledge on the statistical distribution of frequent sets of actions in the plan traces. Then, it forms a weighted propositional satisfiability problem (weighted SAT) and solves it with a weighted MAX-SAT solver. Unlike ARMS, SLAF is able to learn actions with conditional effects. To that end, SLAF relies on building logical constraint formula based on a direct acyclic graph representation. Louga takes also as input plan traces and work with partial noiseless observations. However, Louga is able to learn actions with static properties and negative preconditions. Louga uses a genetic algorithm to learn action effects and an ad-hoc algorithm to learn action preconditions. FAMA takes as input partial plan traces, i.e. plan traces where some actions are missing, and observations are partial. This algorithm turns the task of learning into a planning problem: the learning problem is translated into a planning problem, and it resolves it by using a classical planner. Plan-Milner uses a classification algorithm based on inductive rule learning techniques: it learns action models with discrete numerical values from partial and noisy observations. The AIA algorithm learns planning domains by using a rudimentary query system. It generates plans and queries a black-box AI system with these plans to test them and updates its action model from the black-box responses. Finally, the LOCM family of action model learning approaches (Cresswell, McCluskey, and West 2013; Gregory and Cresswell 2015) works without information about initial, intermediate and final states. These algorithms extract, from plan traces, parameterized automata representing the behaviour of each object of the planning problems. Then preconditions and effects are generated from these automata. The last group of approaches takes as input random walks, that is, sets of action sequences randomly generated. Random walk approaches like IRALe (Rodrigues, Gérard, and Rouveirol 2010) deal with complete but noisy observations. IRALe is based on an online active algorithm to explore and to learn incrementally the action model with noisy observations. Other approaches such as LSONIO (Mourão et al. 2012) deal with both partial and noisy observations. LSONIO uses a classifier based on a kernel trick method to learn action models. It consists of two steps: (1) it learns a state transition function as a set of classifiers, and (2) it derives the action model from the parameters of the classifiers.

Temporal Learning

Temporal PDDL domains have different levels of action concurrency (Cushing et al. 2007). Some are sequential, which means that all plans can be rescheduled into a completely sequential succession of durative actions: each durative action starts after the previous durative action is terminated. Some temporal domains require other forms of action concurrency such as Single Hard Envelope (SHE) (Coles et al. 2009). SHE is a form of action concurrency where a durative action can be executed only if another durative action called the envelope extends over it. (Garrido and Jiménez 2020) have proposed an algorithm to learn temporal domains using CSP techniques. However this approach is limited to sequential temporal domains. To our best knowledge, there is no learning approach for both SHE and sequential temporal domains.

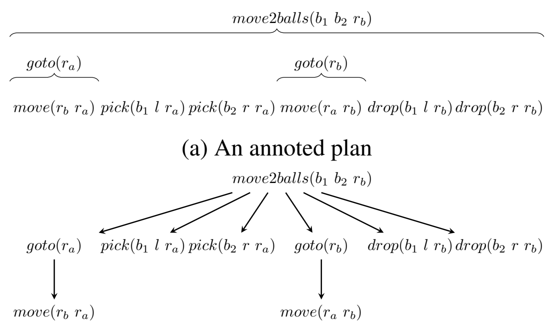
HTN Learning

First of all, the CAMEL algorithm (Ilghami et al. 2002) learns the preconditions of HTN Methods from observations of plan traces, using the version space algorithm. Then,

	Input			Output			
Algorithm	Input	Environment	Noise level	STRIPS	ADL	Temporal	HTN
EXPO	Partial domain, Plan traces	FO	0%	•			
RIM	Partial domain, Plan traces	FO	0%	•			
OPMaker	Partial domain, Plan traces	FO	0%	•			
ARMS	Plan traces	PO	0%	•			
Louga	Plan traces	PO	0%	•			
FAMA	Plan traces	PO	0%	•			
Plan-Milner	Plan traces	PO	10%	•			
AIA	Plan traces	FO	0%	•			
LOCM	Plan traces	NO	0%	•			
LOP	Plan traces	NO	0%	•			
NLOCM	Plan traces	NO	0%	•			
ERA	Random walk	FO	0%	•			
IRALe	Random walk	FO	20%	•			
LSONIO	Random walk	PO	5%	•			
SLAF	Plan traces	PO	0%	•	•		
LAMP	Plan traces	PO	0%	•	•		
(Garrido and Jiménez 2020)	Plans	PO	0%			•	
(Xiaoa et al. 2019)	Plans	FO	0%				•
HTN-Maker	Plans	FO	0%				•
(Hogg, Kuter, and Munoz-Avila 2010)	Plans	FO	0%				•
(Li et al. 2014)	Plans	FO	0%				•
(Garland and Lesh 2003)	Annoted Plans	FO	0%				•
CAMEL	Plans	FO	0%				•
HDL	Plans	FO	0%				•
LHTNDT	Plans	FO	0%				•
HTN-Learner	Decomposition Trees	PO	0%	•			•

Table 1: State-of-the-art action model learning algorithms. From left to right: the kind of input data, the kind of environment: **F**ully **O**bservable, **P**artially **O**bservable or **N**on **O**bservable, the maximum level of noise in observations, and and the domain expressivity STRIPS, ADL, Temporal and HTN

.


move2balls(b1 b2 rb)
goto(ra)
goto(rb)
move(rb ra) pick(b1 l ra) pick(b2 r ra) move(ra rb) drop(b1 l rb) drop(b2 r rb)
move2balls(b1 b2 rb)
goto(ra) pick(b1 l ra) pick(b2 r ra) goto(rb) drop(b1 l rb) drop(b2 r rb)
move(rb ra)
move(ra rb)

(a) An annoted plan

(b) A decomposition tree

Figure 2: HTN Learning Input examples

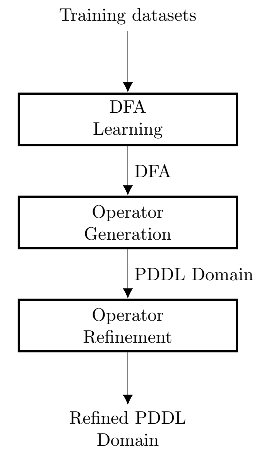
the HDL algorithm (Ilghami, Nau, and Muñoz-Avila 2006) takes as input plan traces. For each decomposition in plan traces, HDL checks if there exist a method responsible of this decomposition. If not, HDL creates a new method and initializes a new version space to capture its preconditions. Preconditions are learned in the same way as in the CAMEL algorithm. Then, (Xiaoa et al. 2019) propose an algorithm to update incomplete methods by task insertions. Then HTN-Maker (Hogg, Munoz-Avila, and Kuter 2008) takes as input plan trace generated from PDDL planner and annoted task provided by a domain expert. These approach use annoted task to build incrementally HTN Methods with preconditions. Then, (Li et al. 2014) proposed an algorithm taking as input only plan traces. This algorithm builds, from plan traces, a context free grammar (CFG) allowing to regenerate all plans. Then, methods are generated using CFG: one method for each production rule in the CFG. Then (Garland and Lesh 2003) and (Nargesian and Ghassem-Sani 2008) proposed to learn HTN Methods from annoted plan. Annoted plan (see Figure - 2a) are plan segmented with the different takes solved. Then, (Hogg, Kuter, and Munoz-Avila 2010) proposed an algorithm based on reinforcement learning. Only HTN-Learner proposes to learn Action Model and HTN Methods from decomposition trees. A decomposition tree (see Figure - 2b) is a tree corresponding to the decomposition of a method.

Contributions

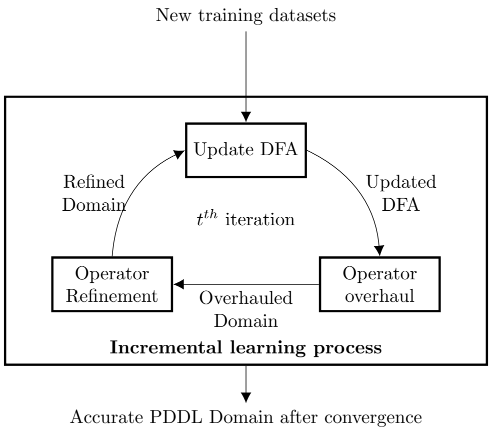
In this section we will give our different contributions. More precisely, we give the method used to generate the observations. Then (see Figure - 3) we describe the algorithm used to learn PDDL domains. Then, we show how to learn PDDL domains incrementally. Finally we show how to learn Temporal domains.

Observation Generation

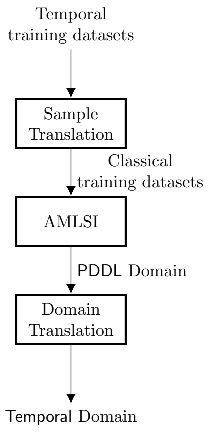
AMLSI produces a set of observations by using a random walk. AMLSI is able to efficiently exploit these observations by taking into account not only the fact that some actions are applicable in certain states but also that others are not. More precisely, to generate the observations, AMLSI uses random walks by applying a randomly selected action to the initial state of the problem. If this action is feasible, it is appended to the current action sequence. This procedure is repeated until the selected action is not feasible in the current state. The feasible prefix of the action sequence is then added to the set of positive samples, and the complete sequence, whose last action is not feasible, is added to the set of negative samples. This generation method allow to maximize data usability in situations where obtaining training



(a) AMLSI: PDDL domain learning overview



(b) IncrAMLSI: Incremental PDDL domain learning overview



(c) TempAMLSI: Temporal domain learning overview

Figure 3: Contributions overview

datasets is costly and/time-consuming.

Domain Learning

PDDL Learning An overview of the AMLSI (Grand, Fiorino, and Pellier 2020b,a) algorithm is shown in Figure 3a. AMLSI is composed of three steps:

1. *DFA Learning:* AMLSI uses an alternative version of the RPNI (Oncina and García 1992) algorithm with the DFA learning algorithm proposed by (Grand, Fiorino, and Pellier 2020b).
2. *Operator Generation:* AMLSI generates the set of PDDL operators from the learned DFA with the observed states.
3. *Operator Refinement:* AMLSI refines the operator preconditions and effects. This refinement steps is necessary to address partial and noisy observations.

Incremental Learning An overview of the IncrAMLSI algorithm (Grand, Fiorino, and Pellier 2021a) is shown in Figure 3b. IncrAMLSI learns incrementally the PDDL domain from incoming data and does not restart from scratch when new data become available at each iteration t. More precisely, IncrAMLSI consists in incrementally updating the PDDL domain with the new incoming training datasets available at iteration t to produce the new domain. This algorithm is made up of three steps:

1. *Update of the DFA* with the DFA learning algorithm proposed by (Grand, Fiorino, and Pellier 2020b).
2. *Overhaul of the PDDL operators* in order to add new operators and remove preconditions and effects that are no longer compatible with positive and negative samples, and the updated DFA at iteration t.
3. *Refinement of the PDDL operators* as in AMLSI algorithm to deal with noisy and partial states in observations.

The incremental process is operated each time new training datasets are input and until convergence of the PDDL domain.

Temporal Learning Some planners, such as Crikey (Halsey, Long, and Fox 2004), solve Temporal Problems by using non-temporal planners. To that end, they convert Temporal Problems into PDDL planning problems, solve them with a non-temporal planner. Then they convert the classical plan into a temporal plan with rescheduling techniques. TempAMLSI (Grand, Fiorino, and Pellier 2021b) reuse this idea in order to learn Temporal domains: TempAMLSI learns a PDDL domain and then infer a Temporal domain. TempAMLSI (summarized in Figure - 3c) has three steps.

1. *Sample Translation:* After having generated the samples of *temporal* sequences (including both feasible and infeasible sequences), TempAMLSI translates them into non-temporal sequences.
2. *PDDL Domain Learning:* TempAMLSI uses AMLSI to learn an intermediate and classical PDDL domain.
3. *Domain Translation:* TempAMLSI translates the PDDL domain into a Temporal domain.

This algorithm is able to learn both Sequential and SHE accurate Temporal domains.

Conclusion and Future Works

As we have seen before, we presented the AMLSI algorithm for learning PDDL domains. More precisely, the AMLSI algorithm learns STRIPS-compliant domains. We plan to extend the expressivity of the learned domains and more specifically we plan to extend AMLSI to learn ADL domains. Next, we presented two extensions: (1) an incremental extension and (2) a temporal extension. We plan to develop a hierarchical extension to learn HTN domains. Finally, the AMLSI algorithm and its extension has only been tested with IPC benchmarks, we plan to test the AMLSI algorithm and its extension with real-world dataset from several fields.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity Planning for the Mars Exploration Rovers. In *Proc of ICAPS*, 40–49.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1): 1–44.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using *LOCM*. *Knowledge Engineering Review*, 28(2): 195–213.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is Temporal Planning Really Temporal? In *Proc of IJCAI*, 1852–1859.

Garland, A.; and Lesh, N. 2003. Learning hierarchical task models by demonstration. *MERL*.

Garrido, A.; and Jiménez, S. 2020. Learning Temporal Action Models via Constraint Programming. In *Proc of ECAI*, 2362–2369.

Grand, M.; Fiorino, H.; and Pellier, D. 2020a. AMLSI: A Novel and Accurate Action Model Learning Algorithm. In *Proc of KEPS workshop*.

Grand, M.; Fiorino, H.; and Pellier, D. 2020b. Retro-engineering state machines into PDDL domains. In *Proc of ICTAI*, 1186–1193.

Grand, M.; Fiorino, H.; and Pellier, D. 2021a. IncrAMLSI: Incremental Learning of Accurate Planning Domains from Partial and Noisy Observations. In *Proc of ICTAI*, 121–128.

Grand, M.; Fiorino, H.; and Pellier, D. 2021b. TempAMLSI : Temporal Action Model Learning based on Grammar Induction. In *Proc of KEPS workshop*.

Gregory, P.; and Cresswell, S. 2015. Domain Model Acquisition in the Presence of Static Relations in the LOP System. In *Proc of ICAPS*, 97–105.

Halsey, K.; Long, D.; and Fox, M. 2004. CRIKEY-a temporal planner looking at the integration of scheduling and planning. In *Proc of the Integrating Planning into Scheduling Workshop*, 46–52.

Hogg, C.; Kuter, U.; and Munoz-Avila, H. 2010. Learning methods to generate good plans: Integrating htn learning and reinforcement learning. In *Proc of AAAI*, volume 24.

Hogg, C.; Munoz-Avila, H.; and Kuter, U. 2008. HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required. In *Proc of AAAI*, 950–956.

Ilghami, O.; Nau, D. S.; and Muñoz-Avila, H. 2006. Learning to Do HTN Planning. In *Proc of ICAPS*, 390–393.

Ilghami, O.; Nau, D. S.; Muñoz-Avila, H.; and Aha, D. W. 2002. CaMeL: Learning Method Preconditions for HTN Planning. In *Proc of ICAPS*, 131–142.

Kucera, J.; and Barták, R. 2018. LOUGA: Learning Planning Operators Using Genetic Algorithms. In *Proc of PKAW Workshop*, 124–138.

Li, N.; Cushing, W.; Kambhampati, S.; and Yoon, S. 2014. Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences. *TIST*, 5(2): 1–32.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An Interactive Method for Inducing Operator Descriptions. In *Proc of ICAPS*, 121–130.

Mourão, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. In *Proc of UAI*, 614–623.

Nargesian, F.; and Ghassem-Sani, G. 2008. LHTNDT: Learn HTN Method Preconditions using Decision Tree. In *Proc of ICINCO-ICSO*, 60–65.

Oncina, J.; and García, P. 1992. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis: Selected Papers from the IVth Spanish Symposium*, volume 1, 49–61. World Scientific.

Rodrigues, C.; Gérard, P.; and Rouveirol, C. 2010. Incremental Learning of Relational Action Models in Noisy Environments. In *Proc of ICLP*, 206–213.

Segura-Muros, J. Á.; Pérez, R.; and Fernández-Olivares, J. 2018. Learning Numerical Action Models from Noisy and Partially Observable States by means of Inductive Rule Learning Techniques. In *Proc of KEPS workshop*.

Shah, M.; Chrpa, L.; Jimoh, F.; Kitchin, D.; McCluskey, T.; Parkinson, S.; and Vallati, M. 2013. Knowledge engineering tools in planning: State-of-the-art and future challenges. *Knowledge engineering for planning and scheduling*, 53: 53.

Shahaf, D.; and Amir, E. 2006. Learning Partially Observable Action Schemas. In *Proc of AAAI Conference on Artificial Intelligence*, 913–919.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowledge Engineering Review*, 22: 117–134.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the Right Questions: Learning Interpretable Action Models Through Query Answering. In *Proc of AAAI Conference on Artificial Intelligence*, 12024–12033.

Xiaoa, Z.; Wan, H.; Zhuoa, H. H.; Herzigb, A.; Perrusselc, L.; and Chena, P. 2019. Learning HTN Methods with Preference from HTN Planning Instances. *Proc of HPlan workshop*, 31.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.

Zhuo, H. H.; Nguyen, T. A.; and Kambhampati, S. 2013. Refining Incomplete Planning Domain Models Through Plan Traces. In *Proc of IJCAI*, 2451–2458.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18): 1540–1569.

Application of Neurosymbolic AI to Sequential Decision Making – Dissertation Abstract

Carlos Núñez-Molina[1]
Juan Fernández-Olivares[2] (PhD director), Pablo Mesejo[3] (PhD co-director)
[1,2,3]Universidad de Granada, Spain
[1]ccaarlos@correo.ugr.es, [2]faro@decsai.ugr.es, [3]pmesejo@decsai.ugr.es

Abstract

In the history of AI, two main paradigms have been applied to solve Sequential Decision Making (SDM) problems: Automated Planning (AP) and Reinforcement Learning (RL). Among the many proposals to unify both fields, the one known as *neurosymbolic AI* has recently attracted great attention. It combines the Deep Neural Networks characteristic of modern RL with the symbolic representations typical of AP. The main goal of this PhD is to progress the state of the art in neurosymbolic AI for SDM. To do so, three different lines of research have been proposed. In the first one, I will perform a study of the literature and summarize my findings into a review. In the second one, I will extend my previous work (Núñez-Molina et al. 2021), which combined Deep Q-Learning with Classical Planning to improve planning performance, with the ability to manage non-determinism and will apply it to a real logistics problem. Finally, in the third line of research, I will develop a method for generating planning problems and leverage it to learn HTN domains without expert traces and to study the properties of planning domains.

Introduction

Sequential Decision Making (SDM) (Littman 1996) is the problem of solving Sequential Decision Processes (SDP). In a SDP, an agent situated in an environment must make a series of decisions in order to complete a task or achieve a goal. These ordered decisions must be selected according to some optimality criteria, generally formulated as the maximization of reward or the minimization of cost. SDPs provide a general framework which has been successfully applied to solve problems in fields as diverse as robotics (Kober, Bagnell, and Peters 2013), logistics (Schäpers et al. 2018), games (Mnih et al. 2015), finance (Charpentier, Elie, and Remlinger 2021) and natural language processing (Wang, Li, and He 2018).

Throughout the years, many AI methods have been proposed to solve SDPs. They can be grouped in two main categories: Automated Planning (AP) (Ghallab, Nau, and Traverso 2004) and Reinforcement Learning (RL) (Sutton and Barto 2018). These two paradigms mainly differ in how they obtain the solution and how they represent their knowledge:

- AP techniques exploit the existing prior knowledge about the environment dynamics, encoded in what is known as the action model or planning domain, to carry out a search process in order to find a valid plan that achieves a set of goals starting from the initial state. Regarding the knowledge representation employed, many AP techniques encode their knowledge in a symbolic manner, using a formal language based on first-order logic such as PDDL (Fox and Long 2003). These symbolic AP methods will be referred to as Symbolic Planning (SP).
- Standard RL methods learn the optimal policy, i.e., a mapping from states to actions in order to maximize reward, automatically from data, without performing any planning process whatsoever. Additionally, the vast majority of RL methods represent their learned knowledge in a subsymbolic way, often as the weights of a Deep Neural Network (DNN).

Each of these paradigms presents several advantages and drawbacks:

- In the case of SP, its symbolic representation is amenable to interpretation, whereas most RL methods represent their knowledge as a non-interpretable black-box. Additionally, SP methods can more easily reason about long-term sequences of actions than RL approaches, which tend to be greedier.
- The main advantage of RL is that it is able to learn the solution of the SDP, i.e., the optimal policy, from data automatically obtained by interacting with the environment. Due to this, RL can be easily applied to solve a wide range of problems with little effort from the human engineers. In the case of SP, these human engineers must provide extensive prior knowledge, mainly in the form of the planning domain, which requires a lot more effort.

Since the shortcomings of SP align with the strengths of RL and vice versa, it seems natural to try to reconcile these two competing paradigms into a single unified approach.

Many works have tried to bridge the gap between RL and SP. Some of these works have focused on extending the capabilities of RL methods with those of SP. Model-based RL (Moerland, Broekens, and Jonker 2020) enhances (model-free) RL with the ability to perform a look-ahead process over a model of the world. Additionally, several model-based RL methods try to learn how to actually carry out this deliberative process, i.e., they learn to plan (Tamar et al. 2016). Relational RL (Tadepalli, Givan, and Driessens 2004) represents the learned knowledge in a symbolic way, with the goal of improving the interpretability and generalization of classical RL. Regarding modern Deep RL, some works have substituted standard DNNs for architectures with relational inductive biases, such as Graph Neural Networks (GNN) (Battaglia et al. 2018), which represent the learned knowledge in terms of objects and their relations.

In parallel to all these efforts, many authors have taken inspiration from RL in order to improve SP. Some works have proposed efficient planning algorithms, such as MonteCarlo Tree Search (MCTS) (Kocsis and Szepesvári 2006), which do not require a symbolic representation of the environment dynamics. In the case of MCTS, the authors employ a RL algorithm known as UCB1 in conjunction with *rollouts* to estimate future rewards and guide the planning process. Other works have developed methods for learning the prior knowledge SP techniques require, in order to relieve some of the burden on experts. Some of them focus on learning the structure of the SDP, whose most important aspects are encoded in the planning domain (Segura-Muros, Pérez, and Fernández-Olivares 2021), whereas others learn domain-specific control knowledge to guide and speed up the search, such as planning policies and heuristics (Jiménez et al. 2012).

In recent years, a novel approach for integrating SP and RL has attracted a lot of attention. This hybrid approach, known with the name of *Neurosymbolic AI* (Besold et al. 2017), consists of methods which combine the DNNs usually employed in modern RL with the symbolic representations typical of SP. Neurosymbolic methods pose a promising approach towards a real unification of SP and RL, since they try to integrate the main strenghts of each approach: the ability to automatically extract knowledge from data as in RL with the interpretability and reasoning ability of SP.

Several works have successfully applied these methods to the field of SDM. (Asai and Fukunaga 2018) proposes *Latplan*, a model that uses a Variational Autoencoder (VAE) (Kingma and Welling 2013) to learn a symbolic action model from pairs of images, which makes possible to apply standard symbolic search algorithms to solve planning problems encoded in a subsymbolic way. (Katz et al. 2021) presents a semi-automatic method for performing scenario planning, i.e., generating a variety of possible future scenarios to support decision making and risk management. It uses DNNs to extract forces and causal relations from documents, and off-the-shelf planners that exploit the extracted information to generate plans corresponding to possible future scenarios. (Shen, Trevizan, and Thiébaux 2020) proposes STRIPS-HGNs, a special type of GNN that receives as input the symbolic description of a planning problem and predicts the heuristic value, which is then used to guide a symbolic planning procedure. Finally, (Toyer et al. 2018) presents ASNets, a novel type of DNN architecture which is able to learn a policy that generalizes to different problems of the same planning domain, thanks to its unique structure which resembles a symbolic planning process.

In the light of these recent advances, the main goal of this PhD dissertation is to progress the state of the art in neurosymbolic AI for SDM, with the development of methods for both solving SDPs and learning aspects of their structure.

Current State of the PhD

So far, the line of research has focused on the integration of SP and Deep RL. In (Núñez-Molina, Fdez-Olivares, and Pérez 2020), we proposed a neurosymbolic online execution system which combines PDDL-based planning with the Deep RL algorithm known as Deep Q-Learning (Mnih et al. 2013), in order to decrease the load of the planner. To perform this combination, we resorted to goal selection. We trained the Deep Q-Learning algorithm to select goals instead of actions, where the available goals are known a priori. Once a goal is selected, it is given to the symbolic planner, which obtains a plan from the current state to the goal. After achieving the goal, Deep Q-Learning is used to select a new one, and this process of interleaving planning and goal selection continues until the final goal is achieved. Instead of maximizing reward, Deep Q-Learning is trained to select goals in such a way that minimizes the length of the plans obtained by the planner. We tested our architecture on a deterministic version of the video game known as BoulderDash, present in the GVGAI framework (Perez-Liebana et al. 2015), where the agent must obtain a number of gems (which correspond to the goals in our method) and then exit the level. We trained our model on a set of training levels and evaluated it on a different set of test levels, in order to test its generalization ability. In this preliminary work, we compared the performance (in terms of plan length) of our model, referred to as *DQP*, with a baseline model trained with supervised learning to select the best next goal in a greedy way. The obtained results show that, as training data increases, our DQP model performs better than the greedy baseline model, since it performs long-term thinking when selecting goals. The results were presented in the ICAPS 2020 IntEx/GR workshop.

In our next work (Núñez-Molina et al. 2021), we greatly improved the performance of our approach regarding goal selection quality. In addition, we compared it against the same planner our architecture uses, but this time with no goal selection at all. The results obtained show that, on (geometric) average, our method is able to greatly reduce planning times in exchange for obtaining plans with 25% more actions. Thus, our approach strikes a good balance between time complexity and solution quality, and makes possible to apply standard symbolic planners to environments with tight time restrictions.

Then, in (Núñez-Molina, Fdez-Olivares, and Pérez submitted) we improved once again the performance of our approach, augmented our architecture with the ability to detect non-achievable goals, provided a more detailed mathematical formulation and compared our approach against standard Deep Q-Learning (with no planning). Results showed that our method generalizes better than standard Deep Q-Learning and is more sample-efficient. Thus, our neurosymbolic approach performs better than standard SP and RL when both plan quality and time requirements are considered. This third work has been submitted to a journal and is currently undergoing minor revisions.

Research Plan

The main research hypothesis explored in this PhD is the following: **Neurosymbolic AI provides a successful approach for solving and learning the structure of SDPs**. In order to verify this hypothesis, six goals corresponding to three different lines of research have been proposed.

The first line of research contains Goal 1 and simply corresponds to a study of the literature. The second one contains Goals 2 and 3 and continues the line of research on goal selection with Deep Q-Learning. The third one entails a method for automatically generating planning problems (Goal 4) and two possible applications of such method to learn aspects of the SDP structure (Goals 5 and 6). Although these three lines of research can be pursued in no particular order, the goals within each of them must be achieved following the goal numeration described below:

Goal 1: Study of the state of the art. Firstly, I will perform a comprehensive study of the state of the art regarding AI methods for SDM, focusing on the case of finite Markov Decision Processes (MDP) (Sutton and Barto 2018). I will research AI methods ranging from standard SP and RL/Deep RL techniques to hybrid approaches such as model-based RL, techniques for *learning to plan* and neurosymbolic models. I will not only study how these techniques can be applied to solve MDPs, i.e., finding the optimal policy/plan, but also how they make possible to learn the structure of the MDP, e.g., the action model and additional aspects such as landmarks (Hoffmann, Porteous, and Sebastia 2004).

This first goal has already been partially completed. As detailed above, I have researched the field of AI for SDM and summarized my findings into a review which I intend to submit very soon. In this review, I have given special emphasis to those hybrid models which try to integrate the competing SP and RL approaches. This work is also part position paper, as I discuss what properties an ideal method for SDM should possess. Based on these ideal features, I propose a theoretical framework to compare the different methods for solving SDPs. As a result of this comparison, I conclude that neurosymbolic models are currently the closest approach to this ideal method for SDM and, thus, pose a very promising direction for future work.

Goal 2: Uncertainty management with DQP. In this goal, I plan to continue my previous line of research on goal selection with Deep Q-Learning. My intention is to enhance my DQP model with the ability to manage uncertainty, so that it can be applied to stochastic environments. To achieve this, I will employ Deep Q-Learning to predict the uncertainty associated with a given goal in addition to the plan length. This uncertainty will correspond to the probability that a plan from the current state to the goal can be successfully executed from start to finish without interruptions (e.g., an obstacle suddenly appearing). This way, it will be possible to select those goals which result in a short plan overall in addition to being likely achievable by the agent, which results essential for dynamic, non-deterministic environments.

This new ability of the DQP model also serves as an execution monitoring tool. By repeatedly predicting the uncertainty associated with an already-selected goal during the execution of its plan, the agent will be able to detect unexpected situations as a result of an increase in the uncertainty value predicted by the network. Once this value surpasses a given threshold, we can consider the goal no longer achievable. In this case, the agent will simply select a new goal to replace the old one and try to achieve it.

One interesting property of this approach is that the goal selection procedure (based on Deep Q-Learning) is responsible for completely managing the uncertainty, i.e., non-determinism, in the environment. In principle, this makes possible to apply a classical, deterministic planner to a stochastic environment, instead of a probabilistic planner. However, it remains unclear if this approach (deterministic planner + goal selection with uncertainty) will be enough in highly dynamic environments or, instead, it will be necessary to also manage uncertainty at the planner level, i.e., by using a probabilistic planner.

Goal 3: Application of DQP to a real logistics problem. As the final step in this line of research, I plan to apply my DQP approach to solve a real-world problem. The goal is to manage the logistics of a company which relies on a fleet of trucks to transport and deliver packages. The main decisions to take concern which packages each truck must carry and the route each truck must follow in order to successfully deliver the packages to their recipients, while meeting a set of requirements such as time deadlines, service cost restrictions, and regulation constraints. Since this problem is too complex to be directly solved with standard SP techniques, I propose to use my DQP approach to map high-level decisions, such as to which truck assign each delivery order, to goals and train my goal selection method to select them so as to optimize a series of metrics, e.g., minimize the total distance traveled, and satisfy a set of criteria, e.g., each package must be delivered before its deadline. Once the goals have been selected, a symbolic planner can then be used to obtain the low-level plan that achieves each goal. Furthermore, the DQP model will be used to adapt to unexpected situations, such as new delivery orders and truck breakdowns. To achieve this, it will constantly monitor the state of the execution and modify the goals to achieve (and their associated plans) when needed.

Several modifications will need to be made in order to adapt the DQP model to this real-world scenario, many of which will only be clear once the specifications of the problem are well known. However, since the data for training the model will be in the form of logs, it seems likely that the DQP model will need to be able to handle symbolic data. Up to this point, the DNN trained with Deep Q-Learning to select goals corresponds to a Convolutional Neural Network (CNN) (Krizhevsky, Sutskever, and Hinton 2012), which receives an image-like encoding of the states and goals to select from. In order to adapt it to symbolic data, I plan to substitute the CNN with a DNN suitable for relational representations. A reasonable approach would be to employ a GNN and encode the information about the states and goals as a graph, which would then be given as input to the network.

Goal 4: Neurosymbolic method for generating planning problems. This goal corresponds to a new line of research, consisting on the automatic generation of planning problems for any given planning domain. This method will be useful for two main purposes: generating data for training Machine Learning methods (e.g., those that learn planning policies and heuristics) and creating a set of benchmark problems to compare the performance of different planners in planning competitions (such as those held in the ICAPS). I plan to develop a method which receives as input a PDDL domain and outputs a set of planning problems pertaining to that particular domain. The problems generated must satisfy three main properties: validity (the initial state must represent a possible situation of the world and the problem must be solvable), diversity (the problems must vary in kind) and quality (according to a metric defined by the user, such as the resolution difficulty of the problems).

In a similar fashion to (Fuentetaja and De la Rosa 2012), I will follow a *problem generation as planning* approach, where SP is used to generate planning problems. Each planning problem is composed of two parts: the initial state and the goal to achieve. For each problem, a valid initial state is first generated. Starting from an empty state, a search process is performed, where at each step either a new object is added or a predicate is instantiated on the state objects. After generating the initial state, a different search process is performed, where each step corresponds to applying a valid action of the planning domain and modifying the state predicates according to the action effects. Once a given number of actions have been executed, the problem goal is generated as a subset of the predicates of the state the search procedure has ended at. In addition to the planning domain, the user may provide two additional pieces of prior information: a validation method (e.g., a list of rules) which receives as input an initial state and returns whether it is valid or not (otherwise every possible initial state is regarded as valid), and the set of predicates which can form part of the goal (otherwise all the predicates will be considered).

The validation method is used to prune the search when a node is generated corresponding to a non-valid initial state. The other validity requirement, regarding the solvability of the generated problems, does not need to be checked, since the sequence of actions applied to obtain the goal from the initial state correspond to a valid solution plan. In order to generate problems of a certain quality, e.g., problems with properties which make them hard to solve, I will rely again on Deep Learning. A DNN (possibly a GNN) will receive as input a partially-generated problem (possibly encoded as a graph) and output its quality value, which will be used to guide the search towards problems of good quality. Once a problem has been completely generated, it will be solved with an off-the-shelf planner in order to assess its quality. For example, the resolution difficulty of a problem can be estimated as the number of states the planner needed to explore to find its solution. The computed metric will serve as a reward signal to train the DNN with RL methods. Finally, it is important that the generated problems are diverse. One possible approach is to obtain at each step the *n* nodes with highest quality and then select one of them completely at random. This provides a simple method for balancing quality and diversity which resembles the *epsilon-greedy* exploration method in RL.

Goal 5: Learning HTN domains from PDDL without plan traces. Here I plan to leverage the problem generation method previously explained in order to learn a Hierarchical Task Network (HTN) (Georgievski and Aiello 2015) domain just from a PDDL domain, with no need for plan traces. Given a planning domain, I will use the method in Goal 4 to generate a large and diverse set of planning problems. Then, the generated problems will be solved with an off-the-shelf planner and the solution plans obtained will be provided, along with the planning domain and problems, as inputs to some HTN learning method such as the one proposed in (Nejati, Langley, and Konik 2006). This way, it will be possible to learn a HTN domain just from a PDDL domain, without needing an expert to provide example problems and their solution plans, since these will be obtained with my problem generation method. If the generated problems properly represent the different types of problems in the domain, then the HTN domain obtained should be applicable to solve any of them. An alternative approach is to bias the problem generation method towards generating instances of a given type, e.g., problems of high difficulty. Then, the HTN domain learned from them should be tailored to this specific type of problems, thus helping to solve them in a very efficient way.

Goal 6: Domain characterization. As the final goal of my PhD, I plan to utilize the problem generation method previously detailed to study the properties of a particular planning domain, what I refer to as *domain characterization*. Given a planning domain, a large and diverse set of planning problems will be generated and then solved with an off-the-shelf planner. Once this data has been obtained, I will use Data Mining techniques to study the properties of the generated problems and their solutions, which will serve to characterize the domain as a whole. One possible approach is to apply clustering techniques to group the problems in clusters of similar problems according to a series of metrics, e.g., resolution difficulty, solution length and resources needed to solve the problem. By studying the properties of these clusters, it will be possible to analyze the different types of problems in the domain, which will help to understand the different situations which can arise in it. For example, in the logistics domain of Goal 3, each problem could correspond to a different logistics task, composed of a particular number of trucks, packages, locations and delivery orders. By clustering the problems according to their resource utilization (e.g., how many kilometers the trucks need to travel to deliver the packages), it will be possible to analyze which characteristics influence the utilization of resources (e.g., large packages and delivery orders for distant cities, etc.), which could be useful as a decision support system.

Acknowledgements

This work is being partially funded by the Andalusian Regional Projects B-TIC-668-UGR20 and PYC20-RE-049UGR, and the Spanish National Project RTI2018-098460-B-I00 with FEDER funds.

References

Asai, M.; and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Besold, T. R.; Garcez, A. d.; Bader, S.; Bowman, H.; Domingos, P.; Hitzler, P.; Kühnberger, K.-U.; Lamb, L. C.; Lowd, D.; Lima, P. M. V.; et al. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*.

Charpentier, A.; Elie, R.; and Remlinger, C. 2021. Reinforcement learning in economics and finance. *Computational Economics*.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*.

Fuentetaja, R.; and De la Rosa, T. 2012. A Planning-Based Approach for Generating Planning Problems. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Georgievski, I.; and Aiello, M. 2015. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*.

Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review*.

Katz, M.; Srinivas, K.; Sohrabi, S.; Feblowitz, M.; Udrea, O.; and Hassanzadeh, O. 2021. Scenario planning in the wild: A neuro-symbolic approach. *FinPlan 2021*.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*.

Littman, M. L. 1996. *Algorithms for sequential decision-making*. Brown University.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*.

Moerland, T. M.; Broekens, J.; and Jonker, C. M. 2020. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*.

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *Proceedings of the 23rd international conference on Machine learning*.

Núñez-Molina, C.; Fdez-Olivares, J.; and Pérez, R. 2020. Improving online planning and execution by selecting goals with deep q-learning. In *ICAPS 2020 Workshop on Integrated Execution (IntEx) / Goal Reasoning (GR)*.

Núñez-Molina, C.; Fdez-Olivares, J.; and Pérez, R. submitted. Learning to select goals in Automated Planning with Deep Q-Learning. *Expert Systems With Applications*.

Núñez-Molina, C.; Vellido, I.; Nikolov-Vasilev, V.; Pérez, R.; and Fdez-Olivares, J. 2021. A Proposal to Integrate Deep Q-Learning with Automated Planning to Improve the Performance of a Planning-Based Agent. In *Conference of the Spanish Association for Artificial Intelligence*.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S. M.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2015. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games*.

Schäpers, B.; Niemueller, T.; Lakemeyer, G.; Gebser, M.; and Schaub, T. 2018. ASP-based time-bounded planning for logistics robots. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Segura-Muros, J. Á.; Pérez, R.; and Fernández-Olivares, J. 2021. Discovering relational and numerical expressions from plan traces for learning action models. *Applied Intelligence*.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tadepalli, P.; Givan, R.; and Driessens, K. 2004. Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 workshop on relational reinforcement learning*.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. *Advances in neural information processing systems*.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2018. Action schema networks: Generalised policies with deep learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Wang, W. Y.; Li, J.; and He, X. 2018. Deep reinforcement learning for NLP. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*.

Counter-Example Based Planning – Dissertation Abstract

Xiaodi Zhang[1]
Supervisors: Alban Grastien[1]**, Hanna Kurniawati**[1]**, Charles Gretton**[1]
[1] School of Computing, Australian National University
Acton, ACT, Australia, 2601
firstname.lastname@anu.edu.au

Abstract

`CPCES` is one of conformant planning problem solvers. It continuously searches candidate plans and counter-examples until finding a valid plan or no solution. The goal of my Ph.D. project includes improving its efficiency, making it compatible with more classical planners, and using it to solve contingent planning problems.

Background

Conformant Planning Problem

Conformant planning is the problem of searching for a plan in an environment that is partially known: either the initial state or the actions' effects are non-deterministic. Conformant planning problem is EXPSPACE-complete (Haslum and Jonsson 1999). In comparison, classical planning problem with no uncertainty is only PSPACE-complete. Conformant planning can be used in different areas. For example, Mars exploration robot can collect soil samples in an unknown environment; logistics company distributes trunks and airplanes in different cities; etc.

Given a set of facts V, a state s is a set of facts s. A conformant planning model is defined as a tuple $P = \langle V, A, \Phi_I, \Phi_G \rangle$, where

- V is a finite set of facts
- Φ_I is the initial states, $s \models \Phi_I$
- Φ_G is the goal states, $s \models \Phi_G$
- A is a set of actions. Each action $a \in A$ is defined as a pair $\langle pre, coneff \rangle$, where pre is the precondition, and $coneff$ is a set of conditional effects. Each conditional effect is a pair $\langle c, eff \rangle$, where c is the condition, and eff is the effects. Effect includes positive effects eff^+ and negative effect eff^-. If a state s satisfies the precondition, after executing action a, the next state s' is $s' = s \cup (\bigcup_{\langle c, eff \rangle \in coneff, s \models c} eff^+) \backslash (\bigcup_{\langle c, eff \rangle \in coneff, s \models c} eff^-)$.

The solution π to P is a sequence of actions $\pi = a_1 ... a_n$. A valid plan must reach the goal states for all initial states.

Here is an example of conformant planning problem in Figure 1. At the beginning, a robot is located in a 5×5 grid, but its exact location is unknown. The robot has access to four actions: GO_E, GO_W, GO_S, and GO_N (go East, go West, go South, and go North, respectively). The grid is surrounded by walls to prevent the robot from moving out. In other words, when robot hits the wall, it keeps its position. The robot should finally move to (3,3), denoted as "G". The unknown initial state is the trickiest part of this problem. However, if the robot does GO_E $\times$ 4, GO_S $\times$ 4, no matter where it is at beginning, it must be finally at (5,1), denoted as "T". So, one of the solutions is GO_E $\times$ 4, GO_S $\times$ 4, GO_N $\times$ 2, GO_W $\times$ 2.

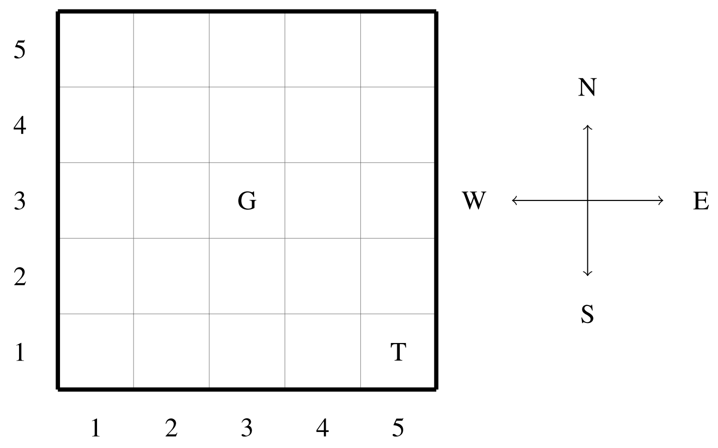


Figure 1: A robot is required to move to "G" but the initial state is unknown. Doing GO_E $\times$ 4, GO_S $\times$ 4 will move the robot to "T" no matter where it is at beginning, since the robot keeps standing when it hits the wall. One of solutions is GO_E $\times$ 4, GO_S $\times$ 4, GO_N $\times$ 2, GO_W $\times$ 2.

Literature Review

It is impractical to enumerate all the possible states during the search, so how to represent belief states decides the efficiency of a planner, and the way of representation is the most difficult parts when designing a planner. `T1` is translation-based approach (Albore, Ramirez, and Geffner 2011). It addresses the difficulty by compiling planning problem to another one which adopts a more compact representation of belief states, so it is easier to be solved. Currently, `T1` is one of the best conformant planners. `Conformant-FF` selects a specific language to represent the problem and uses heuristic functions to estimate the distance to the goal (Hoffmann and Brafman 2006). But conformant-FF has two short comings. First, when there are effects with more than one unknown condition, conformant-FF cannot get enough heuristic values. Second, it is expensive in checking repeated states in the problem of lacking different known propositions in the belief states. MBP is a model-based approach. It uses Symbolic Model Checking techniques and Binary Decision Diagrams to handle a large size of belief states (Bertoli et al. 2001). The shortcoming of MBP is that it requires high memory. POND chooses to use Labeled Uncertainty Graph (LUG) to represent GraphPlan-like information for different initial states, and use lazily enforced hill-climbing strategy to search a plan (Bryce 2006). The CNF planner uses a specific form of CNF, CNF-state, to represent belief states and solve the problem (To, Son, and Pontelli 2010), but for some benchmarks, it spends long time in searching a solution.

As we can see, most planners represents belief states by using other form of languages, diagrams or models. Scala and Grastien created a new method, CPCES, which simplifies the problem not through a specific representation, but through searching counter-examples (Grastien and Scala 2020).

CPCES

CPCES at first was designed to solve deterministic conformant planning problem (unknown initial states and deterministic actions' effects). The algorithm of CPCES can be explained as follow: CPCES addresses the problem by replacing belief states with a small set of initial states, which is called sample. In each iteration, CPCES searches a candidate plan that is valid for the sample. If no candidate plan exists, there is no solution to the problem. Then, CPCES searches a counter-example, an initial state that breaks the candidate plan. If no such counter-example exists, the candidate plan is a valid plan. Otherwise, the counter-example is added into the sample (Algorithm 1).

Algorithm 1: The conformant planner CPCES.

```
 1: input: conformant planning problem $P$
 2: output: a conformant plan, or no plan
 3: $B := \emptyset$
 4: loop
 5:     $\pi :=$ produce-candidate-plan$(P, B)$
 6:     if there is no such $\pi$ then
 7:         return no plan
 8:     end if
 9:     $q :=$ generate-counter-example$(P, \pi)$
10:     if there is no such $q$ then
11:         return $\pi$
12:     end if
13:     $B := B \cup \{q\}$
14: end loop
```

CPCES consists of two main parts. The first part (Line 9) uses SMT formula to search a counter-example to a candidate plan. The basic idea of search counter-example is to do regression, finding an initial state that breaks the precondition of an action or dissatisfies the goal. In the second part (Line 5), CPCES translates the conformant planning problem to the classical planning problem by creating a multi-interpretation domain and a multi-interpretation instance (both are PDDL files), then uses Fast Forward (FF) (Hoffmann and Nebel 2001) to search a candidate plan. The translation procedure is named the reduction of conformant planning: executing n classical plannings in parallel where all actions are synchronized. Because of the reduction, the candidate plan must be valid for all samples.

CPCES has been proved sound and complete. It only uses a small set of initial states to search candidate plans, significantly decreasing the cost. Compared with T1, CPCES performs better on complex domains while T1 is good at 1-width domains (the conformant width is the maximum number of facts in the context whose initial value is unknown).

Scala and Grastien further developed CPCES to address non-deterministic conformant planning problems (Scala and Grastien 2021). Non-deterministic conformant planning is a more complex issue in which both initial states and actions' effects are uncertain. Using Figure 1 as an example for non-deterministic planning. Now the robot is broken. When it wants to move East, it may move North-East instead. To solve this kind of problem, CPCES translates a non-deterministic conformant planning to a classical planning by synchronizing this problem with a non-deterministic finite automaton (NFA) that prevents the invalid solutions.

Work I Have Done

Improve CPCES by Searching Superior Counter-Examples

I noticed that CPCES searches counter-examples randomly. Is it possible to improve CPCES by searching a specific kind of counter-examples? Look at Figure 1. Now there are two robots in the grid, and both of them are required to move to "G". We suppose robots do not collide when they are at the same position. In Table 1, I listed several conditions of counter-examples. The first counter-example (CE1) for two robots are (1,1), and the second counter-example (CE2) for Robot2 is (5,5) but Robot1 remains (1,1). The candidate plan generated in the 2nd round may be weak, because CPCES considers Robot1 can only be at (1,1). Now we update CE2 to CE2' where Robot1 is at (4,4). The candidate plan becomes stronger, because both (1,1) and (4,4) are covered for Robot1. I was inspired from this example, believing that the more information the counter-example covers, the more efficient CPCES searches a valid plan. So I developed CPCES by searching a superior counter-example over another one in each round. A superior counter-example is computed by *tags*.

	Robot1	Robot2
CE1	(1,1)	(1,1)
CE2	(1,1)	(5,5)
CE2'	(4,4)	(5,5)

Table 1: If CE2 for Robot1 remains the same as CE1, it is not a good counter-example. Updating CE2 to CE2' in which Robot1 is located at (4,4) will be better.

A *subgoal* φ is a conjunct of an action precondition or the goal. An action a can be applied at state s only when $\varphi(s)$ is true. A variable v *depends on* another variable v' if there exists an action a that mentions variable v' in its conditional effect. The *context* $ctx(\varphi)$ of subgoal φ is a set of variables mentioned by φ, the variables they depend on, and by transitivity, all the variables they depend on. Given a context c, a *tag* for an initial state s is denoted as $tag_c(s)$, which is the intersection of c and s: $tag_c(s) = c \cap s$. Given a belief state B and a context c, the set of tags of B for c is $T_c(B) = \{tag_c(s) | s \in B\}$. We use $T(B) = \bigcup_{c \in C} T_c(B)$ to represent all the tags in the problem, where C is a set of all the contexts.

Algorithm 2: `compute-superior-counter-example`

```
input: conformant planning problem $P$
$q$ := generate-counter-example($\pi$)
if there is no such $q$ then
    return $\pi$ is valid
end if
loop
    $q'$ := improve-counter-example($B, q$)
    if there is no such $q'$ then
        return $q$
    end if
    $q := q'$
end loop
```

A counter-example s' is better than another counter-example s if $T(B \cup s) \subseteq T(B \cup s')$. This is because the more tags there are, the more information we get. The intersection of the set of plans generated from each tag is the same as the plans generated from belief state: $\Pi(B) = \bigcap_{t \in T(B)} \Pi(t)$. The experiment results illustrate that finding superior counter-examples makes `CPCES` more efficient on multi-context domains, while on 1-context domains the efficiency is not impacted. This work has been published at AAAI-2020 (Zhang, Grastien, and Scala 2020).

Use Fast Downward Planner in CPCES

`CPCES` uses `FF` (Helmert 2006) to search candidate plans. I considered adding Fast Downward (`FD`) into `CPCES` as an option of planner, since `FD` provides more options on search engines and heuristic functions.

`FD` contains two parts (Helmert 2006). The first part translates PDDL files to a SAS+ file. It computes mutex groups, translates the problem to a multi-valued planning task, and finally writes a SAS+ file to save the task. The second part is using SAS+ file to search a valid plan. Translation helps `FD` simplify the question so the problem can be solved quickly. However, this is a trade-off, since translation itself is a complex procedure and it is expensive.

Using `FD` in `CPCES` directly is impractical. `CPCES` calls `FF` in each iteration, and with the number of iterations increasing, the interpretation instance file becomes larger and larger. It is difficult for `FD` to handle with translating a large file, not to mention `FD` will be used in all iterations. I did some experiments and the results display that for all the domains, the searching time of `FD` is too long, sometimes even out of memory.

To address this issue, I designed a viable approach to use `FD` in `CPCES`. In `CPCES`'s interpretation instance file, each state variable v is replaced by a new variable v_i where i is the interpretation number (iteration number). It is easy to understand that if two variables v_i and v'_i are mutually exclusive, v_k and v'_k $(k \neq i)$ must be mutually exclusive. This is because variables represented by k and variables represented by i are two parallel classical problems but have the same properties. So separately translating interpretation file and merging them is applicable.

Having `FD` translate the whole interpretation file is difficult, but translating an interpretation file with only single interpretation number should be much easier. Based on this idea, I designed a new algorithm to replace `FF` with `FD` by letting `FD` translate the interpretation file with only one interpretation number in each iteration, and merging the result into previous SAS+ file (Figure 2).

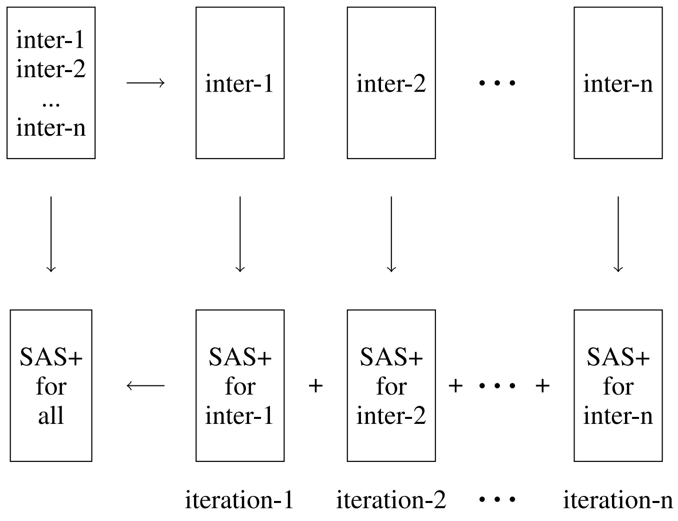


Figure 2: In each iteration, `CPCES` adds a new interpretation variable into interpretation instance file, so the interpretation instance file becomes larger and larger. Translating a large PDDL file to a SAS+ file by `FD` is impractical. I solved this problem by asking `FD` to translate the interpretation instance file with a single interpretation number in each iteration, then merge the result with previous SAS+ file.

This approach has been tested on several benchmarks. The search engine in `FD` is eager best-first search, with best-first open list and FF heuristic. The results shows that the search time is extremely shorter than using `FD` directly. However, this approach is currently worse than `FF` in terms of searching time. I am not sure whether this is because the search engine is not the best. So, I will test more search engines in the future.

Future Works

Add More Planners into CPCES

`CPCES` currently chooses `FF` to search a plan. This is not friendly, because some people may be interested in other planners or algorithms, such as `FD`, granplan, partial order planning. It is necessary to add them into `CPCES`.

Contingent Planning Problem

In contingent planning, the initial state is unknown, but the agent can observe the nearby environment during the actions. For example, in Figure 1, when the robot goes to "T", the wall at the East and the South will be detected. Then, the robot will GO_N or GO_W rather than GO_E or GO_W to avoid hitting the wall.

Given a set of facts V, a state s is a set of facts $s(v)$ where $v \in V$. A contingent planning model is defined as a tuple $P = \langle V, A, O, \Phi_I, \Phi_G \rangle$, where

- V, Φ_I, Φ_G, and A are defined as before.
- O is a set of observations (sensing actions). Each observation $o \in O$ can be defined as a pair $\langle c, v \rangle$, where c is the condition, and v is a positive variable in V. The pair indicates that the truth value of v is observable when c holds.

The solution π to P is a tree of actions, in which the fork is a sensing action, and the child node is the next action decided by what has been observed.

Compared with contingent planning problem, conformant planning problem lacks sensing actions. Because of it, conformant planning is actually a special contingent planning where there is no observation, so no branch in the plan. While `CPCES` works on conformant planning, why not contingent planning?

Increase the Efficiency of CPCES

The number of initial states in `CPCES` decides the complexity of the problem. In fact, some initial states are more important than others. Back to Figure 1, for instance, if we only consider two initial states: (1,1), (5,5), the result is the same as considering all the initial states. These two initial states are called "important initial states". Even though we ignoring all the other 23 initial states, these 2 initial states are strong enough to help us find a valid plan. Finding important initial states before searching a plan may dramatically increase the efficiency of the planner. But how to find these important initial states is a difficult problem. What is more, is the procedure of finding important initial states expensive? Is it worthwhile to find important initial states?

Probabilistic Planning Problem

Uncertainty about action's effect is a key feature of probabilistic planning problem: a probability will be assigned to each effect. As a consequence, the goal is to find a plan that is valid with a specified problem. How to develop `CPCES` to solve probabilistic planning problem? Is `CPCES` more efficient comparing other probabilistic planners? Can I solve probabilistic planning problem by combining `CPCES` with other planners?

Build CPCES Website

`CPCES` is a successful conformant planner, and it has a huge potential to be further developed. Propagating `CPCES` to the world is my duty, so build a website is necessary. We encourage everyone to participate in developing `CPCES` so that `CPCES` can become one of the most useful planners in the world.

References

Albore, A.; Ramirez, M.; and Geffner, H. 2011. Effective heuristics and belief tracking for planning with incomplete information. In *Twenty-First International Conference on Automated Planning and Scheduling*.

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. MBP: a model based planner. In *Proc. of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*.

Bryce, D. 2006. POND: The partially-observable and nondeterministic planner. *Sixteenth International Conference on Automated Planning and Scheduling*, 58.

Grastien, A.; and Scala, E. 2020. CPCES: A planning framework to solve conformant planning problems through a counterexample guided refinement. *Artificial Intelligence*, 284: 103271.

Haslum, P.; and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *European Conference on Planning*, 308–318. Springer.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J.; and Brafman, R. I. 2006. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7): 507–541.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Scala, E.; and Grastien, A. 2021. Non-Deterministic Conformant Planning Using a Counterexample-Guided Incremental Compilation to Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 299–307.

To, S. T.; Son, T. C.; and Pontelli, E. 2010. A New Approach to Conformant Planning Using CNF*. In *Twentieth International Conference on Automated Planning and Scheduling*.

Zhang, X.; Grastien, A.; and Scala, E. 2020. Computing superior counter-examples for conformant planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 10017–10024.

Data Efficient Paradigms for Personalized Assessment of Taskable AI Systems – Dissertation Abstract

Pulkit Verma
Thesis Advisor: **Siddharth Srivastava**
Autonomous Agents and Intelligent Robots Lab,
School of Computing and Augmented Intelligence, Arizona State University, USA
verma.pulkit@asu.edu

Abstract

The vast diversity of internal designs of taskable black-box AI systems and their nuanced zones of safe functionality make it difficult for a layperson to use them without unintended side effects. The focus of my dissertation is to develop algorithms and requirements of interpretability that would enable a user to assess and understand the limits of an AI system's safe operability. We develop a personalized AI assessment module that lets an AI system execute instruction sequences in simulators and answer the queries about its execution of sequences of actions. Our results show that such a primitive query-response capability is sufficient to efficiently derive a user-interpretable model of the system's capabilities in fully observable, and deterministic settings.

1 Introduction

The growing deployment of AI systems presents a pervasive problem of ensuring the safety and reliability of these systems. The problem is exacerbated because most of these AI systems are neither designed by their users nor are their users skilled enough to understand their internal working, i.e., the AI system is a black-box for them. Hence such systems may be used by non-experts who may not understand how they work or what they can and cannot do. Ongoing research on the topic focuses on the significant problem of answering such a user's questions about the system's behavior (Chakraborti et al. 2017a; Dhurandhar et al. 2018; Anjomshoae et al. 2019). However, most non-experts hesitate to ask questions about new AI tools (Mou and Xu 2017) and often do not know which questions to ask for assessing the safe limits and capabilities of an AI system. This problem is aggravated in situations where an AI system can carry out planning or sequential decision making. Lack of understanding about the limits of an imperfect system can result in unproductive usage or, in the worst-case, serious accidents (Randazzo 2018). This, in turn, limits the adoption and productivity of the AI systems.

My dissertation work aims to create general algorithms and methods for interpretability which when used with a black-box AI system, can help generate a description of its capabilities by interrogating it. Consider a situation where a logistics company buys new delivery robots. The person managing these robots is unsure whether the robots correctly understand a task, or if they can even execute it safely. If the manager was dealing with a delivery person, it might ask them questions such as "do you think it would be alright to bring refrigerated items in a regular bag?" If the answer is "yes", it might be a cause for concern. Answers to such questions can help the manager develop an understanding of the robot's frame of knowledge, or "model" while placing a minimal introspective requirement on the robot.

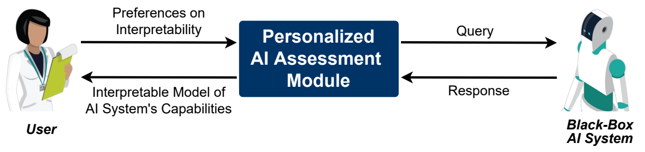

Preferences on Interpretability
Personalized AI Assessment Module
Query
Interpretable Model of AI System's Capabilities
Response
User
Black-Box AI System

Figure 1: The personalized AI assessment module uses the user's preferred vocabulary, queries the AI system, and delivers an interpretable model of the AI system's capabilities.

I will next explain the focus of my dissertation (Sec. 2), followed by a short discussion on related work (Sec. 3), and will finally discuss some preliminary results (Sec. 4).

2 Focus of My Dissertation

In my dissertation, I plan to develop a *personalized AI-assessment module* (AAM), shown in Fig. 1, which can derive the model of capabilities of a black-box AI system in terms of an user-interpretable vocabulary. AAM takes as input using as input (i) the agent (ii) a compatible simulator using which the agent can simulate its primitive action sequences; and (iii) the user's concept vocabulary, which may be insufficient to express the simulator's state representation. Such assumptions on the agent are common. In fact, use of third-party simulators for development and testing is the bedrock of most of the research on taskable AI systems today (including game playing AI, autonomous cars, and factory robots). Providing simulator access for assessment is reasonable as it would allow AI developers to retain freedom and proprietary controls on internal software while supporting calls for assessment and regulation using approaches like ours. AAM then queries the AI system and receives its responses. At the end of the querying process, AAM returns a user-interpretable model of the AI system's capabilities. This approach's advantage is that the AI system need not know the user vocabulary or the modeling language.

Most simulator-based and analytical-model-based AI systems can easily answer the kind of questions discussed earlier. However, identifying the high-level capabilites of the AI system and generating the right set of questions to ask the AI system to efficiently learn a model of system's capabilities is a challenging problem. The focus of this new direction of research is on solving this problem. In context of this work, "actions" refer to the core *functionality* of the agent, denoting the agent's decision choices, or primitive actions that the agent could execute (e.g., a keystrokes in a video game). In contrast, "capabilities" refer to the *high-level behaviors* that the AI system can perform using its AI algorithms for behavior synthesis, including planning and learning (e.g., navigating to a room, opening a door, etc.). Thus, actions refer to the set of choices that a tabular-rasa agent may possess, while capabilities are a result of its agent function (Russell 1997) and can change as a result of algorithmic updates even as the agent uses the same actions.

Additionally, this proposed method, when used with any AI system, would also help make them compliant with Level II assistive AI – systems that make it easy for users to learn how to use them safely (Srivastava 2021).

2.1 Generating Interrogation Policies

I aim to create an interrogation policy that will generate the queries for the AI system, and use the AI system's answers to estimate its model in the user-interpretable vocabulary. I plan to generate these queries by reducing the query generation to a planning problem and then use an interrogation algorithm to iteratively generate new queries actively, based on responses to previous queries.

2.2 Inferring the Action Model

Given the predicates and actions, there is an exponential number of PDDL (McDermott et al. 1998) models possible. To avoid this combinatorial explosion, I plan to use a top-down process that eliminates large classes of models, inconsistent with the AI system, by computing queries that discriminate between pairs of *abstract models*. When an abstract model's answer to a query differs from that of the AI system, we can eliminate the entire set of possible models that are refinements of this abstract model.

I plan to start research on this front with simplistic queries in deterministic fully observable environments and expand the scope to more general settings. I plan to first extend this to settings where the model of an AI system adapts itself to work with the user in a better way, or due to some other reason. This will avoid relearning the complete model from scratch, and will learn the AI system's model much faster. In the future, this mechanism can be extended to more general forms of queries. Similar to active learning, information theoretic metrics can also be utilized to ascertain which queries will be better at any given time in the querying process.

2.3 Discovering the Capabilities and Learning their Descriptions

As mentioned earlier, I want the assessment module to discover the high-level capabilities of the AI system that can plan (using search or a policy), and not just the action model of an AI system. I plan to collect a set of state observations capturing the behavior of the AI system in form of the state transitions. I would then discover the high-level capabilities of the AI system's behavior using those state transitions, and then learn the description of these capabilities similar to the learning of action model discussed earlier. I plan to extend this to settings where either the capabilities are stochastic even though the low level transition system is deterministic, or the low level transition itself is stochastic, thereby resulting in capabilities that are stochastic.

3 Related Work

Learning action models Several action model learning approaches (Gil 1994; Yang, Wu, and Jiang 2007; Cresswell, McCluskey, and West 2009; Zhuo and Kambhampati 2013; Aineto, Celorrio, and Onaindia 2019) have focused on learning the AI system's model using passively observed data. Jiménez et al. (2012) and Arora et al. (2018) present a comprehensive review of such approaches. These approaches do not feature any interventions, hence are susceptible to learning buggy models. Unlike these approaches, our approach queries the AI system and is guaranteed to converge to the true model while presenting a running estimate of the accuracy of the derived model; hence, it can be used in settings where the AI system's model changes due to learning or a software update.

Differential assessment Bryce, Benton, and Boldt (2016) address the problem of learning the updated mental model of a user using particle filtering given prior knowledge about the user's mental model. However, they make a strong assumption that the user knows enough to point out errors in the learned model if needed. Model reconciliation literature (Chakraborti et al. 2017b; Sreedharan et al. 2019; Sreedharan, Chakraborti, and Kambhampati 2021) deals with inferring the differences between the user and the agent models and removing them using explanations. These methods consider white-box known models whereas our approach works with black-box AI systems.

Learning high-level models Given a set of options encoding skills as input, Konidaris, Kaelbling, and Lozano-Perez (2018) and James, Rosman, and Konidaris (2020) propose methods for learning high-level propositional models of options representing various "skills." They assume access to predefined options and learn the high-level symbols that describe those options at the high-level. While they use options or skills as inputs to learn models defining when those skills will be useful in terms of auto-generated symbols (for which explanatory semantics could be derived in a post-hoc fashion), our approach uses user-provided interpretable concepts as a priori inputs to learn AI system capabilities: high-level actions as well as their interpretable descriptions in terms of the input vocabulary.

4 Preliminary Results

We developed three preliminary versions of the personalized AI assessment module, each focusing on one specific sub-

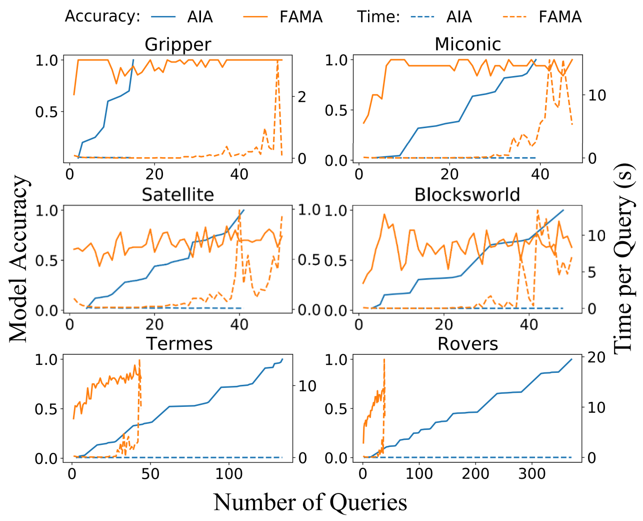


Figure 2: Performance comparison of AIA and FAMA in terms of model accuracy and time taken per query.

problem of the overall larger goal.

Learning the Action Model The first preliminary version of the AI assessment module, called the agent interrogation algorithm (AIA) (Verma, Marpally, and Srivastava 2021), efficiently derives a user-interpretable model of the system in stationary, fully observable, and deterministic settings. In the context of this initial work, user-interpretable means STRIPS-like (Fikes and Nilsson 1971) models because such models can be easily translated into interpretable descriptions, and they also allow interventions and assessment of causality. In the future, I plan to learn more general and more expressive models of the AI system.

Also, in this version, we used *plan outcome queries* which are parameterized by an initial state and a plan; and ask the AI system, the length of the longest prefix of the plan that it can execute successfully when starting in the given initial state, as well as the final state that this execution leads to. E.g., "Given that the truck $t1$ and package $p1$ are at location $l1$, what would happen if you executed the plan $\langle load\_truck(p1, t1, l1), drive(t1, l1, l2), unload\_truck(p1, t1, l2)\rangle$?".

We compared AIA with the closest related work FAMA (Aineto, Celorrio, and Onaindia 2019) in terms of; the accuracy of the learned model, the number of queries asked, and the time taken to generate those queries. Fig. 2 summarizes our findings for systems initialized with IPC domains. AIA takes lesser time per query and shows better convergence to the correct model. FAMA sometimes reaches nearly accurate models faster, but its accuracy continues to oscillate, making it difficult to ascertain when the learning process should be stopped. This is because the solution to FAMA's internal planning problem introduces spurious palm tuples in its model if the input traces do not capture the complete domain dynamics. Also, in domains with negative preconditions like Termes, FAMA was unable to learn the correct model.

We also showed that AIA can be used with simulator-based systems that do not know about predicates and report states as images. To test this, we wrote classifiers to detect predicates from images of simulator-states in the PDDL-Gym (Silver and Chitnis 2020) framework. This framework provides ground-truth PDDL models, thereby simplifying the estimation of accuracy. We initialized the AI system with one of the two PDDLGym environments, Sokoban and Doors. AIA inferred the correct model in both cases, and the average number of queries (over 5 runs) used to predict the correct model for Sokoban and Doors were 201 and 252, respectively.

Finally, we also show that the models that we learn capture the correct causal relationships in the AI system's behavior in terms of how the system operates and interacts with its environment (Verma and Srivastava 2021), unlike the models learned by approaches that only use observational data. We call such causal model a *generalized dynamical causal model* of the AI system capturing under what conditions it executes certain actions and what happens after it executes them.

Differential Assessment We developed a *differential assessment* version of the personalized AI assessment module, called DAAISy (Nayyar, Verma, and Srivastava 2022). This addresses the problem of accurately predicting the behavior of a black-box AI system that is evolving and adapting to changes in the environment it is operating in.

The algorithm for differential assessment utilizes an initially known PDDL model of the AI system in the past, and a small set of observations of AI system's execution. It uses these observations to develop an incremental querying strategy that avoids the full cost of assessment from scratch and outputs a revised model of the system's new functionality.

We refer to a predicate in an action's precondition or effect as a *pal-tuple*, and it can have three modes; positive, negative, or absent, depending on whether that predicate is present in the action's precondition (or effect) in as a positive literal, a negative literal or is absent. To assess the performance of our approach with increasing drift, we employed two methods of generating the initial domains: (a) dropping the *pal-tuples* already present, and (b) adding new *pal-tuples*. For each experiment, we used both types of domain generation. We generated different initial models by randomly changing modes of random *pal-tuples* in the IPC domains. Thus, in all our experiments an IPC domain plays the role of ground truth model and a randomized model is used as the initial known model.

We evaluated the performance of DAAISy along two directions; the number of queries it takes to learn the updated model of the AI system with increasing amount of drift, and the correctness of the model DAAISy learns as compared to the AI system's updated model.

As shown in the plots in Fig. 3, the computational cost of assessing each AI system, measured in terms of the number of queries used by DAAISy, increases as the amount of drift in the AI system's model increases. This is expected as the amount of drift is directly proportional to the number of *pal-tuples* affected in the domain. This increases the number

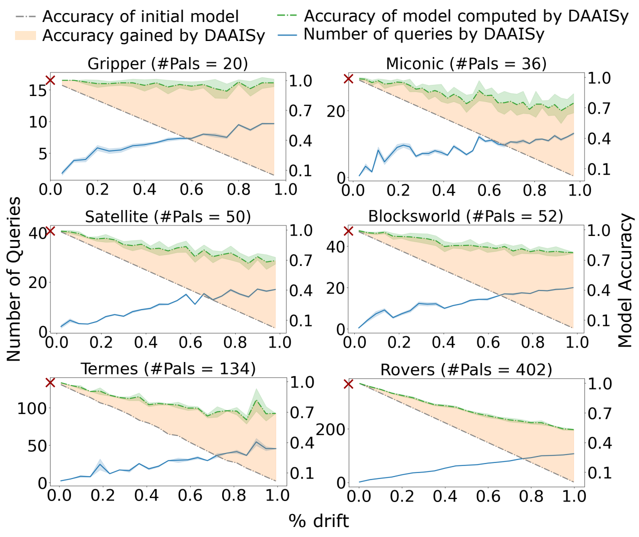


Figure 3: The number of queries used by DAAISy and AIA (marked × on y-axis), as well as accuracy of model computed by DAAISy with increasing amount of drift. Amount of drift equals the ratio of drifted *pal-tuples* and the total number of *pal-tuples* in the domains (#Pals).

of *pal-tuples* that DAAISy identifies as affected, and hence ends up asking more questions.

Also, DAAISy always took fewer queries as compared to AIA to reach reasonably high levels of accuracy because AIA does not use information about the initial known model of the AI system and thus ends up querying for all possible *pal-tuples*. DAAISy, on the other hand, predicts the set of *pal-tuples* that might have changed based on the observations collected from the AI system and thus requires significantly fewer queries.

Discovering the capabilities and learning their descriptions We also developed a version of AAM that can discover high-level capabilities of an AI planning agent expressible in terms of the user-interpretable concept vocabularies (Verma, Marpally, and Srivastava 2022). The descriptions of these capabilities as a model are returned to the user as opposed to the model of agent's primitive actions.

We initialized the agents using the General Video Game Artificial Intelligence framework (Perez-Liebana et al. 2016). For each agent, we create a random game instance with the goal of achieving one of the user's specified properties of interest (implemented as predicates). We use the solution to that instance to generate an execution trace that is used to discover the capabilities of the agent. We then ask the agent a sequence of queries and use the responses to complete the descriptions of these capabilities in a STRIPS-like form. Note that these queries are generated in high-level user vocabulary that the agent does not understand, hence we split each query into multiple sub-queries in a form that agent can answer. The multiple agent responses are also converted to the high-level responses used to complete the capability descriptions. The approach is guaranteed to compute

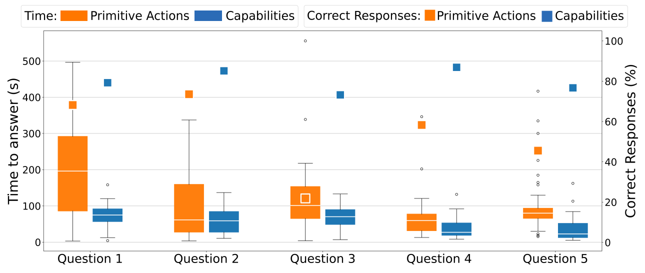


Figure 4: Data from behavior analysis shows that using computed capability descriptions took lesser time and yielded more accurate results.

capability descriptions that are correct in the sense that they are consistent with the execution traces, and refinable and executable with respect to the true capabilities of the agent.

We also conducted a user study to evaluate interpretablity of the capability descriptions computed by our approach. Intuitively, our notion of interpretability matches that of common English and its use in AI literature, e.g., as enunciated by Doshi-Velez and Kim (2018): *"the ability to explain or to present in understandable terms to a human."* We evaluate this through the following operational hypothesis:

H1. The discovered capabilities make it easier for users to analyze and predict outcome of agent's possible behaviors.

We designed a user study to evaluate H1. This study compares the predictability and analyzability of agent behavior in terms of the agent's low-level actions and high-level capabilities. Each user is explained the rules of an ATARI-like game. One group of users – called the primitive action group – are presented with text descriptions of the agent's primitive actions, while the users in the other group – called the capability group – are presented with a text description of the six capabilities discovered by our approach. The capability group users are asked to choose a short summarization for each capability description, out of the eight possible summarizations that we provide, whereas the primitive action group users are asked to choose a short summarization for each of the five primitive action description, out of the five possible summarizations that we provide. Then each user is given the same 5 questions in order. Each question contains two game state images; start and end state. The user is asked what sequence of actions or capabilities that the agent should execute to reach the end state from the start state. Each question has 5 possible options for the user to choose from, and these options differ depending on their group. We then collect the data about the accuracy of the answers, and the time taken to answer each question.

The results for the behavior analysis study are shown in (Fig. 4) The users took less time to answer questions and they got more responses correct when using the capabilities as compared to using primitive actions. This validates H1 that the discovered capabilities made it easier for the users to analyze and predict the agent's behavior correctly.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning Action Models With Minimal Observability. *Artif. Intell.*, 275: 104–137.

Anjomshoae, S.; Najjar, A.; Calvaresi, D.; and Främling, K. 2019. Explainable Agents and Robots: Results from a Systematic Literature Review. In *Proc. AAMAS*.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A Review of Learning Planning Action Models. *The Knowledge Engineering Review*, 33: E20.

Bryce, D.; Benton, J.; and Boldt, M. W. 2016. Maintaining Evolving Domain Models. In *Proc. IJCAI*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017a. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proc. IJCAI*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017b. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proc. IJCAI*.

Cresswell, S.; McCluskey, T.; and West, M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In *Proc. ICAPS*.

Dhurandhar, A.; Chen, P.-Y.; Luss, R.; Tu, C.-C.; Ting, P.; Shanmugam, K.; and Das, P. 2018. Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives. In *Proc. NeurIPS*.

Doshi-Velez, F.; and Kim, B. 2018. *Considerations for Evaluation and Generalization in Interpretable Machine Learning*, 3–17. Springer International Publishing.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4): 189–208.

Gil, Y. 1994. Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *Proc. ICML*.

James, S.; Rosman, B.; and Konidaris, G. 2020. Learning Portable Representations for High-Level Planning. In *Proc. ICML*.

Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A Review of Machine Learning for Automated Planning. *The Knowledge Engineering Review*, 27(4): 433–467.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61(1): 215–289.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Mou, Y.; and Xu, K. 2017. The Media Inequality: Comparing the Initial Human-Human and Human-AI Social Interactions. *Computers in Human Behavior*, 72: 432–440.

Nayyar, R. K.; Verma, P.; and Srivastava, S. 2022. Differential Assessment of Black-Box AI Agents. In *Proc. AAAI*.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; and Lucas, S. 2016. General Video Game AI: Competition, Challenges and Opportunities. In *Proc. AAAI*.

Randazzo, R. 2018. What went wrong with Uber's Volvo in fatal crash? Experts shocked by technology failure. *The Arizona Republic*.

Russell, S. J. 1997. Rationality and Intelligence. *Artificial Intelligence*, 94(1-2): 57–77.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. In *ICAPS 2020 Workshop on Planning and Reinforcement Learning*.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2021. Foundations of Explanations as Model Reconciliation. *Artificial Intelligence*, 103558.

Sreedharan, S.; Hernandez, A. O.; Mishra, A. P.; and Kambhampati, S. 2019. Model-Free Model Reconciliation. In *Proc. IJCAI*.

Srivastava, S. 2021. Unifying Principles and Metrics for Safe and Assistive AI. In *Proc. AAAI*.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the Right Questions: Learning Interpretable Action Models Through Query Answering. In *Proc. AAAI*.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2022. Discovering User-Interpretable Capabilities of Black-Box Planning Agents. In *Proc. KR*.

Verma, P.; and Srivastava, S. 2021. Learning Causal Models of Autonomous Agents using Interventions. In *IJCAI 2021 Workshop on Generalization in Planning*.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning Action Models from Plan Examples Using Weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.

Zhuo, H. H.; and Kambhampati, S. 2013. Action-Model Acquisition from Noisy Plan Traces. In *Proc. IJCAI*.

Domain Specific Situated Planning – Dissertation Abstract

Devin Thomas

University of New Hampshire, Advisor: Dr. Wheeler Ruml

Abstract

There has been much recent work on finding paths in grid maps among moving obstacles. However, in addition to assuming complete omniscience regarding the map and the obstacles' trajectories, previous work has also assumed that time stands still while the agent plans. My dissertation addresses situated pathfinding, in which time passes and the obstacles continue to move while the agent plans. I will study situated planning in three domains: Grid pathfinding among moving obstacles, orienteering and opportunistic science.

Introduction

Traditionally when planning we assume that we receive the problem instance as input, then formulate a plan, then the clock begins and the agent executes the plan. Sometimes we would prefer to consider time passing as we plan, which is situated planning. I am working on situated planning in three domains, Grid pathfinding among moving obstacles, orienteering and opportunistic science.

When planning, our aim is to find a quick and safe path for our agent to reach its goal. While realtime planning algorithms address this by setting a fixed time bound for the agent to return an incremental plan. In contrast a situated agent plans "as the clock ticks" (Cashmore et al. 2018a) time passes, whether the agent is moving, thinking or waiting. Situated planning is less rigid than realtime planning, the agent may choose to spend additional time planning if it believes it will benefit from doing so. This "metareasoning" about when and for how long to plan maybe be important for some situated agents.

In my dissertation we will explore situated planning in several domains. First in grid based path planning, where the environment includes both static and moving obstacles and the agent seeks to minimize it's goal achievement time while avoiding collision with any obstacles. Second in the orienteering problem, where the agent is given a graph containing nodes with varying values, and has a time limit to visit some portion of those nodes and accumulate as much total value as it can. Thirdly the problem of opportunistic science, where the agent may have a window of time to amend its plan to take advantage a transient measurement. In the remainder of this abstract I will give an overview of the background of situated planning, and each of the domains we intend to work on.

Situated Planning

(Russell and Wefald 1991) argue that computations are actions, and the utility of such an action should be derived from its effect on the agents choice of actions. This utility can be estimated from statistical knowledge of the utility of previous computation actions. More recently the problem of situated planning was posed by Cashmore et al. (2018a), where the planner understands that execution is waiting on planning. This allows the planner to prune partial plans where the planning would likely finish too late to execute. In domain independent planning with absolute deadlines this showed empirical improvements over a baseline planner which had a set planning time, and moved all the deadlines earlier by that same planning time. (Shperberg et al. 2019) formalize the metareasoning problem allocating planning effort when actions expire (AE2) where the situated agent must decide which search nodes to expend planning effort on, when they each have an expiration time, and expected completion effort in addition to the normal cost. They optimize to maximize the probability that at least one solution is found by the deadline. They develop a formal MDP solution for AE2, and empirically demonstrate a greedy scheme which was near optimal in solution quality, while also fast enough to be used in metareasoning. The delay-damage aware (DDA) greedy scheme presented by (Shperberg et al. 2021) provides an optimal pseudo-polynomial solution in the case of known deadlines and, a fast greedy scheme that shows improvements over previous schemes with unknown deadlines. Situated planning is an area of active work, and it remains unclear how much of the sophisticated theoretical work can be applied to concrete problems. This dissertation focuses on three different domains where we can test the utility of these metareasoning algorithms.

Three Problem Domains

We have three domains which we will explore situated planning in. This first is a 2D grid with static and moving obstacles, this setting has been the focus of the work thus far. The others are orienteering and opportunistic science which we have discussed as potential domains to explore, but have not begun working in yet.

Grid Pathfinding Among Moving Obstacles

The problem statement for situated grid pathfinding among moving obstacles is: we have an agent that is situated on a 2D grid, with 8-way movement plus the ability to wait in place. The environment contains static and moving obstacles, the safety of the agent is binary. If the agent collides with an obstacle at any point in time it dies, otherwise it is safe. The agent has a starting location, and a goal location to reach as quickly as possible avoiding collision with any obstacles. The cost of the agent's actions is the time they take to complete, and the state space is $\langle x, y, t\rangle$ where the location is discrete and the temporal dimension continuous.

More formally, situated grid pathfinding among moving obstacles is a 6-tuple $\langle S, N, A, C, s_{start}, G\rangle$ where:

- S is a set of states, which are tuples $\langle x, y, t\rangle$ representing the agent's discrete grid cell and real-valued time.
- I is a function from x, y locations to a set of intervals $\{[t_i, t_j], \ldots, [t_k, t_l]\}$, representing safe times when the agent can occupy the grid point at $\langle x, y\rangle$ without colliding with a dynamic obstacle. Static obstacles correspond to grid cells with no safe intervals. Dynamic obstacles create unsafe intervals corresponding to the grid cells they occlude.
- A is a set of actions where each action $a \in A; a : S \to S$ has a duration t_a. We use 8-way motion augmented with a wait action.
- C is the mapping of actions to their durations. $C : A \to \mathbb{R}+$, we use $C(a \in A) = t_a$. $\{up, down, left, right\}$ have a duration of 1, diagonal motions have a duration of $\sqrt{2}$.
- G is the goal grid cell, $\langle x, y\rangle$.
- $s_{start} \in S$ is the starting state.

Given a SSIPP problem presented to an agent at time t_0, a emergent solution plan is defined as a sequence of actions $(a_0, a_1, ..., a_i, ..., a_N)$ emitted by the agent, where

- The solution begins at S_{start}.
- Define $Succ(s, a) : S \times A \to S$ the successor function returning the result of applying action a at state s.
- We can then recursively define the states of the path $s_i = Succ(s_{i-1}, a_{i-1})$ with s_0 being the S_{start}.
- Each action is feasible, the states s_i are always within a safe interval, and thus never in collision with any obstacle.
- the agent ends at the goal cell after a finite sequence of actions.

The objective of the agent is to execute a plan that reaches a goal state as quickly as possible.

The offline equivalent to this problem is addressed by safe interval path planning(SIPP). The safe intervals are constructed by grouping all consecutive co-located states into a safe interval at that location, $i \in I$. This compresses the continuous time dimension into a compact discrete representation which can then be solved with optimally with A^* (Phillips and Likhachev 2011) (Yakovlev and Andreychuk 2017), or sub-optimally with variations of weighted A^* or focal search (Yakovlev, Andreychuk, and Stern 2020) or with anytime algorithms which run fast enough to be used in soft-realtime on a specific problem instance (Narayanan, Phillips, and Likhachev 2012).

SIPP and its variants take advantage of the property that when searching offline there is a built in dominance of states within an interval, with earlier arrival into an interval always being better. This is no longer the case when the agent is situated, as a hasty agent may miss out on opportunities that require more careful deliberation. Because of this our situated problem requires a more sophisticated handling of intervals to be correct. Our more sophisticated method, and how and when it is an improvement over simpler or incorrect methods is part of our upcoming paper.

In this domain we are also exploring how cutting edge methods from realtime search apply to situated planning. The situated agent must perform heuristic learning in order to escape local minima, the agent can back up information from the search frontier using methods like local search space real time A*(LSS-LRTA*) (Koenig and Sun 2009), potentially with separate learning of the static environment from the dynamic environment using partitioned learning realtime A*(PLRTA*) (Cannon, Rose, and Ruml 2014). Our agent must also have a strategy for picking which search nodes to expand first, in similar realtime settings it has been shown to be beneficial to use a $\hat{f}(n) = g(n) + \hat{h}(n)$ rather than $f(n) = g(n) + h(n)$ where $\hat{h}(n)$ is the expected cost to goal at search node n, rather than a heuristic cost to goal $h(n)$ (Kiesel, Burns, and Ruml 2015). The agent also may benefit from committing to more than one action at a time, thus increasing the amount of time it has to plan the next set of actions. One method would be to commit all the way to the search frontier, which then allows a situated planner to plan even further for the next set of actions thus dynamically increasing the size of partial plan committed to (Kiesel, Burns, and Ruml 2015). For realtime planning that scheme has been shown to be too aggressive, leading the agent to over-commit. (Cheng and Ruml 2019) found that a constrained dynamic scheme, where the size of set of actions committed to was allowed to grow, but only by a fixed amount per iteration was better.

This initial grid problem setting while simple, provides an arena to test how methods from realtime planning can be adapted to the situated setting. We have begun our experimentation using the same set of instances used by the bounded-suboptimal SIPP paper (Yakovlev, Andreychuk, and Stern 2020). Our initial results suggest that with appropriate choices of state space representation and learning algorithm, a situated agent can perform very well in these instances. As such it is not yet clear if this setting will be suitable, or sufficiently complex to explore our questions on how the agent should metareason.

The grid path planning domain has been the focus of our work so far, we are in the process of preparing a paper exploring the effects of choices in state space representation and heuristic learning on the success of a situated agent. Following the paper we plan to finish exploring the effects of other methods that have been found to be beneficial in realtime search in the situated setting, such as altering the choice of expansion algorithm and the commitment strategy. Additionally so far we have used problem instances from (Yakovlev, Andreychuk, and Stern 2020), we would like to construct instances of our own, for example instances of the game Frogger would be appropriate for this situated agent.

The Orienteering Problem

In a orienteering problem, the agent is given a starting location, and a set of checkpoints each with some score associated with it. The agent seeks to visit these checkpoints and return to the starting location, such that it maximizes the sum of the scores it accumulates while returning by some deadline (Vansteenwegen, Souffriau, and Oudheusden 2011). The orienteering problem with time windows, augments the orienteering problem with locations who's point value is only captured if the agent reaches them within a certain time interval. In this way it is similar, but distinct from the safe intervals mentioned before.

Orienteering is a natural problem to consider in a situated manner, as only limited information of the problem instance is available prior to starting the race. Additionally a situated agent may benefit from metareasoning on how long to spend planning, especially in the orienteering problem with time windows, when spending a substantial time at the start planning might mean missing out on quick deadlines. A situated agent in this setting must balance its short term goals to accumulate the most points, with its long term constraint that it must return by the time limit.

Opportunistic Science

The opportunistic science domain is an agent who is executing a long term plan which it must complete. During execution the agent is presented with an unexpected opportunity to achieve a large reward by expending some surplus resources it is holding in reserve, whether those resources are time, or battery charge or sampling capacity. These opportunities are transient, and rare or otherwise impractical to plan for as part of the long term plan. Similar to the orienteering problem, the agent must keep in mind the need to maintain adequate resources to complete its long term goals, while reacting to opportunities in such a way as to maximize its benefit.

There are numerous autonomous agents who may encounter transient scientific opportunities. This can be a martian rover who could observe a dust devil and record it, or use it to clean off it's solar panels (Lorenz and Reiss 2015). This has included a focus on on-board systems to adjust to unexpected surplus or deficits in rover resources (Gaines et al. 2006) (Rabideau et al. 2020). Similarly this problem has been explored with autonomous submersibles performing underwater maintenance (Cashmore et al. 2018b). Those agents represent systems with high latency to get assistance planning, in contrast there are systems like the CHIME radio telescope, which generates terabytes of data each second, and as such can only buffer the data for a matter of seconds while it decides what events are most worth recording (Amiri et al. 2018).

Conclusions

Situated planning in contrast with traditional offline planning has time progress while the agent plans. Prior work in situated planning has been mainly theoretical or domain independent. We aim to explore situated planning in three contrasting domains, giving us wide view of how the theory of situated planning can be applied. Our work so far has been focused on the grid path planning domain, which has shown some promising initial results.

References

Amiri, M.; Bandura, K.; Berger, P.; Bhardwaj, M.; Boyce, M.; Boyle, P.; Brar, C.; Burhanpurkar, M.; Chawla, P.; Chowdhury, J.; et al. 2018. The CHIME fast radio burst project: system overview. *The Astrophysical Journal*, 863(1): 48.

Cannon, J.; Rose, K.; and Ruml, W. 2014. Real-time Motion Planning with Dynamic Obstacles. *Ai Communications*, 27: 345–362.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2018a. Temporal planning while the clock ticks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; and Ridder, B. 2018b. Opportunistic Planning in Autonomous Underwater Missions. *IEEE Transactions on Automation Science and Engineering*, 15(2): 519–530.

Cheng, C. C.; and Ruml, W. 2019. Real-time Heuristic Search in Dynamic Environments. In *Twelfth Annual Symposium on Combinatorial Search*.

Gaines, D.; Estlin, T.; Chouinard, C.; Casta, R.; Casta, A.; Bornstein, B.; Anderson, R.; Judd, M.; Nesnas, I.; and Rabideau, G. 2006. Opportunistic Planning and Execution for Planetary Exploration.

Kiesel, S.; Burns, E.; and Ruml, W. 2015. Achieving goals quickly using real-time search: experimental results in video games. *Journal of Artificial Intelligence Research*, 54: 123–158.

Koenig, S.; and Sun, X. 2009. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3): 313–341.

Lorenz, R. D.; and Reiss, D. 2015. Solar panel clearing events, dust devil tracks, and in-situ vortex detections on Mars. *Icarus*, 248: 162–164.

Narayanan, V.; Phillips, M.; and Likhachev, M. 2012. Anytime Safe Interval Path Planning for dynamic environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4708–4715.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. 5628 – 5635.

Rabideau, G.; Wong, V.; Gaines, D.; Agrawal, J.; Chien, S.; Fosse, E.; and Biehl, J. 2020. Onboard automated scheduling for the mars 2020 rover.

Russell, S.; and Wefald, E. 1991. Principles of metareasoning. *Artificial intelligence*, 49(1-3): 361–395.

Shperberg, S. S.; Coles, A.; Cserna, B.; Karpas, E.; Ruml, W.; and Shimony, S. E. 2019. Allocating planning effort when actions expire. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2371–2378.

Shperberg, S. S.; Coles, A.; Karpas, E.; Ruml, W.; and Shimony, S. E. 2021. Situated Temporal Planning Using Deadline-aware Metareasoning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 340–348.

Vansteenwegen, P.; Souffriau, W.; and Oudheusden, D. V. 2011. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1): 1–10.

Yakovlev, K.; and Andreychuk, A. 2017. Any-Angle Pathfinding for Multiple Agents Based on SIPP Algorithm. arXiv:1703.04159.

Yakovlev, K.; Andreychuk, A.; and Stern, R. 2020. Revisiting Bounded-Suboptimal Safe Interval Path Planning. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 300–304. AAAI Press.

Domain-Independent Heuristics in Probabilistic Planning – Dissertation Abstract

Thorsten Klößner
Saarland University, Saarland Informatics Campus
Foundations of Artificial Intelligence Group
kloessner@cs.uni-saarland.de
Supervisor: Jörg Hoffmann

Abstract

It has been almost two decades since MDP heuristic search algorithms have been developed. These algorithms guarantee to find an optimal partial policy for the initial state for several optimization objectives without necessarily expanding the entire state space, if provided with a heuristic that provides optimistic objective value estimates for all states. While a large set of such domain-independent heuristic families is available in classical planning, the same cannot be said about probabilistic planning. So far, with the exception of occupation measure heuristics for (constrained) Stochastic Shortest-Path Problems, most of the research on domain-independent heuristic construction consists of using a classical heuristic on the all-outcomes determinization of the planning problem, in which the probabilistic effect of an action can be chosen at will. Because this approach is agnostic to the uncertainty in the underlying problem, these heuristics are often uninformative. In this thesis, I will develop new domain-independent heuristics for probabilistic planning which take the probabilistic nature of the problem into account. To this end, the main focus of the thesis lies in developing the foundations of abstraction heuristics for probabilistic planning, in particular Pattern Database heuristics and Merge-and-Shrink heuristics.

Introduction

AI planning is a long-standing discipline in artificial intelligence which deals with the automatic deduction of strategies for autonomous agents. In classical planning, the simplest form of planning, a single agent acts inside a fully observable and deterministic environment. Probabilistic Planning relaxes these assumptions to allow stochastic environments, where an action leads to one of multiple possible outcomes, each occurring with an associated probability that is known a priori. In this thesis, I will focus on fully observable problems, where problems are commonly modelled as a Markov Decision Process (MDP). In this setting, the behaviour of the agent is typically specified by a function from states to actions, called a *policy*.

There exist various optimization criteria can be considered to specify the desired behaviour of an agent. In this thesis, I focus on two settings in particular. In the *MaxProb* setting, we want to find a policy that maximizes the probability to reach a set of goal states when starting in the initial state of the problem. On the other hand, Stochastic Shortest-Path Problems (SSPs, Bertsekas and Tsitsiklis (1991)) associate each action application with a real-valued cost. The objective of the agent is to reach a set of goal states with probability one in the limit, while minimizing the expected accumulated cost to do so.

There exist a plethora of algorithms to solve MDPs both optimally and approximately for these settings. Heuristic search algorithms for SSPs (e.g. Hansen and Zilberstein (2001), Bonet and Geffner (2003), Trevizan et al. (2017)) have the potential to prevent the exhaustive generation of the whole state space of the problem. These algorithms require an *admissible* heuristic to ensure optimality, i.e. a function which underestimates the real minimal expected cost-to-goal of a state. These algorithms can also be extended to MaxProb (Kolobov et al. 2011), where they require an upper-bounding heuristic on the maximal goal probability of a state instead.

Although substantial effort has been invested into the development of MDP heuristic search algorithms themselves, research regarding admissible heuristics which can be supplied to these algorithms is, to this day, rather sparse. Most of the research utilizes the all-outcomes determinization (Yoon, Fern, and Givan 2007), in which the agent can simply choose the probabilistic outcome of an action. With the help of this transformation, any classical heuristic can be used to guide the search by delegating to the determinization.

While the determinization-based approach enables the use of a large arsenal of classical planning heuristics, these heuristics are often not very informative, since the uncertainty in the problem is completely ignored. On the other hand, occupation measure heuristics for (constrained) SSPs (Trevizan, Thiébaux, and Haslum 2017) actually make use of the probabilistic information. These LP-based heuristics can be seen as extensions of operator-counting heuristics (Pommerening et al. 2014) to probabilistic planning. Trevizan, Thiébaux, and Haslum experimental evaluations show that these heuristics lead to greatly decreased search effort compared to determinization-based heuristics. As of today, these heuristics are considered state-of-the-art.

In this thesis, I develop novel domain-independent heuristics for probabilistic planning which are not agnostic to the uncertainty of the problem. To this end, I extend several families of abstraction heuristics from classical planning to probabilistic planning, including Pattern Database heuristics (Korf 1997; Haslum et al. 2007; Pommerening, Röger, and Helmert 2013), and Merge-and-Shrink heuristics (Helmert, Haslum, and Hoffmann 2007; Nissim, Hoffmann, and Helmert 2011; Helmert et al. 2014). Apart from an experimental evaluation, I investigate the theoretical relationships with previous heuristics, in particular with classical abstraction heuristics on the determinization as well as occupation measure heuristics.

Preliminaries

I consider probabilistic planning problems with full observability in the context of different optimization objectives. This thesis abstract focuses primarily on the MaxProb objective, which prioritizes goal probability maximization, as well as Stochastic Shortest-Path Problems (SSPs, Bertsekas and Tsitsiklis (1991)). To represent both settings in a uniform manner, the underlying probabilistic model will be kept separate from the considered optimization objective.

MDPs and Optimization Objectives As the baseline model, we define a Markov Decision Process (MDP) as a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, s_\mathcal{I} \rangle$. $\mathcal{S}$ is the finite, non-empty set of *states*, $\mathcal{A}$ is a finite, non-empty set of *actions*, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the *transition probability function* and $s_\mathcal{I} \in \mathcal{S}$ is the *initial state* of the problem. For any state-action pair $\langle s, a \rangle \in \mathcal{S} \times \mathcal{A}$, either $\sum_{t \in \mathcal{S}} \mathcal{T}(s,a,t) = 1$ (a is enabled in s) or $\mathcal{T}(s,a,t) = 0$ for all $t \in \mathcal{S}$ (a is disabled in s). The set of actions enabled in s is denoted $\mathcal{A}(s)$. We assume that $\mathcal{A}(s) \neq \emptyset$ for all states. We can easily introduce artificial self-loops to achieve this. Finally, a policy is a mapping $\pi : \mathcal{S} \to \mathcal{A}$ with $\pi(s) \in \mathcal{A}(s)$ for every state $s \in \mathcal{S}$.

The MaxProb optimization objective is specified by a set of goal states $\mathcal{S}_\mathcal{G} \subseteq \mathcal{S}$. The semantics of a policy in presence of this optimization objective is given by the policy state value function $\mathcal{V}^\pi_{\text{MP}} : \mathcal{S} \to [0,1]$, where $\mathcal{V}^\pi_{\text{MP}}(s)$ represents the probability to reach the goal when starting in the state s and following policy π. It is defined as the (point-wise) *least* solution of the equation system

$$\mathcal{V}^\pi_{\text{MP}}(s) = \begin{cases} 1 & s \in \mathcal{S}_\mathcal{G}, \\ \sum_{t \in \mathcal{S}} \mathcal{T}(s,a,t)\mathcal{V}^\pi_{\text{MP}}(t) & s \notin \mathcal{S}_\mathcal{G}. \end{cases}$$

The optimal state value function $\mathcal{V}^*_{\text{MP}}$ is defined as $\mathcal{V}^*_{\text{MP}}(s) := \max_\pi \mathcal{V}^\pi_{\text{MP}}(s)$. A policy $\pi^\star$ is optimal if $\mathcal{V}^{\pi^\star}_{\text{MP}} = \mathcal{V}^*_{\text{MP}}$. For MaxProb, an optimal policy always exists. Moreover, we say that π is an s-proper policy, if $\mathcal{V}^\pi_{\text{MP}}(s) = 1$. If π is s-proper for all s, then π is proper.

The optimization objective considered for SSPs is given by a set of goal states $\mathcal{S}_\mathcal{G} \subseteq \mathcal{S}$ and an action cost function $c : \mathcal{A} \to \mathbb{R}$. This objective makes two additional assumptions: (i) There exists a proper policy and (ii) Every improper policy eventually accumulates infinite cost[1]. The policy state value function $\mathcal{V}^\pi_{\text{SSP}} : \mathcal{S} \to \mathbb{R}$ is only defined for proper policies π for this objective. $\mathcal{V}^\pi_{\text{SSP}}(s)$ gives the expected accumulated cost until the goal is reached when starting in state s and acting according to π. It is the unique solution of the equation system

$$\mathcal{V}^\pi_{\text{SSP}}(s) = \begin{cases} 0 & s \in \mathcal{S}_\mathcal{G} \\ c(\pi(s)) + \sum_{t \in \mathcal{S}} \mathcal{T}(s,a,t)\mathcal{V}^\pi_{\text{SSP}}(t) & s \notin \mathcal{S}_\mathcal{G} \end{cases}$$

The optimal state value function $\mathcal{V}^*_{\text{SSP}}$ is defined by $\mathcal{V}^*_{\text{SSP}}(s) := \min_{\pi \text{ proper}} \mathcal{V}^\pi_{\text{SSP}}(s)$. Analogously, a policy $\pi^\star$ is optimal if $\mathcal{V}^{\pi^\star}_{\text{SSP}} = \mathcal{V}^*_{\text{SSP}}$ and always exists.

Probabilistic SAS$^+$ Tasks The planning problem is specified as a probabilistic SAS$^+$ task (Trevizan, Thiébaux, and Haslum 2017), except that the cost function is omitted since it is not needed for MaxProb. A probabilistic SAS$^+$ task is a tuple $\langle \mathcal{V}, \mathcal{A}, s_\mathcal{I}, \mathcal{G} \rangle$. $\mathcal{V}$ denotes the *state variables*, where each $v \in \mathcal{V}$ is associated with a finite domain $\mathcal{D}(v)$ of at least two values. A *partial state* is a partial function $s : \mathcal{V} \rightharpoonup \bigcup_{v \in \mathcal{V}} \mathcal{D}(v)$ with $s(v) \in \mathcal{D}(v)$ if defined. We denote the variables on which s is defined by $\mathcal{V}(s)$. s is a *state* if $\mathcal{V}(s) = \mathcal{V}$. The set of states of a probabilistic SAS$^+$ task Π is denoted $\mathcal{S}(\Pi)$. For a set of variables $P \subseteq \mathcal{V}$ and partial state s, we denote by $s[P]$ the *projection of s onto P* and define the set $S[P] := \{s[P] \mid s \in S\}$. We say s *subsumes* t, written $t \subseteq s$, if $\mathcal{V}(s) \subseteq \mathcal{V}(t)$ and $s[\mathcal{V}(s)] = t[\mathcal{V}(s)]$. The *application* of partial state e onto partial state s is defined by $\text{appl}(s,e)(v) = e(v)$ if $v \in \mathcal{V}(e)$ and $s(v)$ otherwise. $\mathcal{A}$ is the set of *actions*. An action a specifies its *precondition* $\text{pre}(a)$, and a probability distribution Pr_a over effects, where an effect is a partial state. The possible effects of a are denoted $\text{Eff}(a) := \{e \mid \text{Pr}_a(e) > 0\}$. Lastly, the *initial state* $s_\mathcal{I}$ is a state and the *goal* $\mathcal{G}$ is a partial state.

A probabilistic SAS$^+$ task $\Pi = \langle \mathcal{V}, \mathcal{A}, s_\mathcal{I}, \mathcal{G} \rangle$ induces the MDP $\langle \mathcal{S}(\Pi), \mathcal{A}, \mathcal{T}, s_\mathcal{I} \rangle$ where $\mathcal{T}(s,a,t)$ is defined as 0 if $\text{pre}(a) \not\subseteq s$ and by

$$\mathcal{T}(s,a,t) := \sum_{\substack{e \in \text{Eff}(a) \text{ s.t.} \\ \text{appl}(s,e) = t}} \text{Pr}_a(e)$$

otherwise. The set of goal states for the MaxProb and SSP objective is given by $\mathcal{S}_\mathcal{G} = \{s \mid s \subseteq \mathcal{G}\}$.

Heuristics A heuristic h returns an estimate $h(s)$ for the optimal state value $\mathcal{V}^*_{\text{MP}}(s)$ or $\mathcal{V}^*_{\text{SSP}}(s)$ of a state s. For MaxProb, a heuristic is *admissible* if $h(s) \geq \mathcal{V}^*_{\text{MP}}(s)$, whereas it is admissible in the SSP setting if $h(s) \leq \mathcal{V}^*_{\text{SSP}}(s)$. For SSPs, a heuristic is *consistent* if the equation

$$h(s) \leq c(a) + \sum_{t \in \mathcal{S}} \mathcal{T}(s,a,t)h(t)$$

is satisfied for every $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and *goal-aware* if $h(s) = 0$ for goal states $s \in \mathcal{S}_\mathcal{G}$. These properties are convenient because a heuristic that is both consistent and goal-aware is admissible. Also, some SSP heuristics search algorithms like iLAO* (Hansen and Zilberstein 2001) can be optimized for consistent, goal-aware heuristics.

Lastly, a finite family of heuristics $(h_i)_{i \in I}$ (where I is some index set) is *additive* if $\sum_{i \in I} h_i(s) \leq \mathcal{V}^*_{\text{SSP}}(s)$ and *multiplicative* if $\prod_{i \in I} h_i(s) \geq \mathcal{V}^*_{\text{MP}}(s)$.

[1]More general SSP definitions exist (Kolobov et al. 2011; Guillot and Stauffer 2020), but I focus on this traditional definition for the sake of brevity.

Abstraction Heuristics

In classical planning, abstractions heuristics are a fairly versatile family of heuristics that has been studied extensively in various forms, for example through Pattern Databases (e.g. Korf (1997); Haslum et al. (2007); Pommerening, Röger, and Helmert (2013)), Cartesian Abstraction (Seipp and Helmert 2013) and Merge-and-Shrink Abstraction (e.g. Helmert, Haslum, and Hoffmann (2007); Nissim, Hoffmann, and Helmert (2011); Helmert et al. (2014)). In classical planning, an abstraction is typically specified by a surjective abstraction mapping $\alpha : \mathcal{S} \to \alpha(\mathcal{S})$, which associates each state with a corresponding abstract state $\alpha(s) \in \alpha(\mathcal{S})$. For a labelled transition system (LTS), the deterministic planning model usually assumed in classical planning, an abstraction induces an abstract LTS which overapproximates the behaviour of the original LTS. This abstract LTS can then be solved to obtain an admissible heuristic for the original problem. To do the same with respect to an MDP, a definition for the abstract MDP induced by an abstraction mapping needs to be proposed.

Projection Heuristics

In recent work (Klößner et al. 2021), we propose a definition for the specific case of *projections*. A projection is an abstraction mapping $s \mapsto s[P]$ which considers a subset of state variables (a *pattern*) $P \subseteq \mathcal{V}$ of the problem. The abstract MDP for a projection with respect to P is defined as $\langle S[P], A, \mathcal{T}_P, s_{\mathcal{I}}[P]\rangle$, where the transition probability $\mathcal{T}_P(\sigma, a, \tau)$ is defined as 0 if $\mathrm{pre}(a)[P] \nsubseteq \sigma$, otherwise

$$\mathcal{T}_P(\sigma, a, \tau) = \sum_{\substack{e \in \mathrm{Eff}(a) \text{ s.t.} \\ \mathrm{appl}(\sigma, e[P]) = \tau}} \mathrm{Pr}_a(e).$$

The probabilistic projection heuristic $h^P(s) := \mathcal{V}^*_{\mathrm{MP}}(s[P])$ is an admissible heuristic for MaxProb, and the analogous heuristic $h^P(s) := \mathcal{V}^*_{\mathrm{SSP}}(s[P])$ is even consistent and goal-aware for SSPs, when the abstract set of goal states for both objectives is defined as $\mathcal{S}_{\mathcal{G}}[P]$ and the cost function for SSPs is unchanged for the abstraction. Most importantly, these heuristics dominate the respective determinization-based projection heuristic on the same pattern.

Pattern Database Heuristics

In classical planning, Pattern Database (PDB) heuristics are a family of abstraction heuristics that use several projections in unison to achieve a more accurate heuristic. Given a collection of patterns $C \subseteq \mathcal{P}(\mathcal{V})$, the corresponding pattern database heuristic is constructed by precomputing a lookup table of heuristic values for each individual projection heuristic h^P for $P \in \mathcal{C}$. When an estimate for a state s is requested, these individual projection heuristics can then be combined by performing the necessary table lookups and taking the highest estimate: $h^{\max}_C(s) = \max_{P \in C}\{h^P(s)\}$. An even better approach is to employ additivity constraints to find sub-collections $D \subseteq C$ such that the heuristics $(h^P)_{P \in D}$ become additive (Haslum et al. 2007). Max'ing over these sub-collections then yields and an even stronger heuristic, called the *canonical PDB heuristic* $h^{\mathrm{can}}_C(s)$.

We published two papers (Klößner et al. 2021; Klößner and Hoffmann 2021) in which we transfer these concepts to probabilistic planning and construct pattern database heuristics which exploit a collection of MDP projections instead. In particular, we show that the well-known additivity constraints considered by Haslum et al. can be adapted and used in a straightforward manner to obtain additivity constraints for SSPs and even multiplicativity constraints for MaxProb.

In more detail, we say that an action *affects* a variable v if there is a possible effect $e \in \mathrm{Eff}(a)$ with $v \in \mathcal{V}(e)$ and $e(v) \neq \mathrm{pre}(a)(v)$. An action a affects a pattern P if any variable $v \in P$ is affected. We show that, for a collection of patterns C, if every action affects at most one pattern $P \in C$, the projection heuristics $(h^P)_{P \in C}$ are additive for the SSP objective, and multiplicative for the MaxProb objective.

This observation leads to a direct generalization of $h^{\mathrm{can}}_C(s)$ for both MaxProb and SSPs. We show that construction of this heuristic is analogous to the construction in classical planning: Finding the maximal additive sub-collections of C can still be accomplished by finding the maximal cliques in the graph where nodes are the pattern $P \in C$ and two patterns are connected if their projections are additive, which is easy to check for only two patterns. Our empirical evaluation shows a substantial improvement over the determinization-based canonical PDB heuristics.

In very recent work (Klößner et al. 2022b), we also deal with the question of how to construct reasonably construct the initial pattern collection C when the problem is no longer deterministic. We consider and extend two approaches that have been studied in classical planning: Pattern construction via Counter-example guided abstraction refinement (CEGAR, Rovner, Sievers, and Helmert (2019)) and pattern construction as a search problem solved using hill-climbing (Haslum et al. 2007). We reformulate both of these frameworks to operate on MDPs, as opposed to using the classical algorithm variants on the determinization. Compared to classical pattern construction techniques on the determinization, both algorithms have a significant advantage in particular problem domains. However, there also exist many domains in which we observe no benefit over determinization-based pattern construction, so these algorithms can by no means be seen as a universal answer to this research question. We might therefore revisit this topic in the future.

Merge-and Shrink Heuristics

The Merge-and-Shrink framework (Dräger, Finkbeiner, and Podelski 2006) is a framework that originates from model checking but has since found use in various forms in classical planning, in particular to compute abstraction heuristics. In a nutshell, the Merge-and-Shrink framework operates on a *factored transition system* which is a tuple of labelled transition systems (LTS) $F = \langle \Theta_1, \dots, \Theta_n \rangle$ with a common set of labels. Each Θ_i is called a *factor*. These factors implicitly represent the LTS that is their synchronous product. If $\Theta_i = \langle \mathcal{S}^i, \mathcal{L}, \mathcal{T}^i, s^i_{\mathcal{I}} \rangle$, the synchronous product is the LTS $\bigotimes F = \langle \bigotimes_{i=1}^n \mathcal{S}^i, \mathcal{L}, \mathcal{T}^{\otimes}, \langle s^1_{\mathcal{I}}, \dots s^n_{\mathcal{I}} \rangle \rangle$ where

$$\mathcal{T}^{\otimes} := \{\langle \langle s_1, \dots, s_n \rangle, a, \langle t_1, \dots, t_n \rangle \rangle \mid \forall i \in \{1, \dots, n\}. \langle s_i, a, t_i \rangle \in \mathcal{T}^i\}.$$

At the start of Merge-and-Shrink, the tuple of atomic projections (to a single variable) of the LTS induced by the planning task yields the initial factored transition system and is an exact implicit representation of the state space. In each iteration, the algorithm applies one of four transformation to the factored transition system:

1. Merge: Select two factors Θ_1 and Θ_2 and replace them by their synchronous product $\Theta_1 \otimes \Theta_2$.
2. Shrink: Select a factor and apply an abstraction on top of it. Replace the old factor with the abstraction.
3. Prune: Select a factor and remove states which are not *alive*, i.e. which are unreachable or cannot reach the goal.
4. Label Reduction: Reduce the number of labels by aggregating multiple labels into a common label.

Depending on how the framework is used, the procedure either stops when there is only one factor left or when a memory or time limit is reached.

The algorithms has several important properties. If we ignore label reduction, each factor represents an abstraction of the original state space at any point in time in the algorithm (modulo non-alive states). The algorithm can therefore be used to construct abstraction heuristics. Furthermore, without label reduction and if bisimulation is used as a shrinking strategy, then each factor represents a bisimulation the factored LTS implicitly represents a bisimulation of the original LTS at any point in time (modulo non-alive states).

As of yet, it remains an open question whether the Merge-and-Shrink framework can be formulated for probabilistic planning. The first approach that comes to mind is to model each factor as an MDP and to initialize the factored representation with all atomic (MDP) projections. A reasonable definition of a product between MDPs should again have the property that the product of all atomic projections yield the original state space. Regrettably, this is impossible to accomplish. Consider a planning task with variables $v, w \in \{0, 1\}$ and a single action a with the following effects:

$$\Pr_a(\{v \mapsto 1\}) = 0.25 \quad \Pr_a(\{w \mapsto 1\}) = 0.25$$
$$\Pr_a(\{v \mapsto 1, w \mapsto 1\}) = 0.5$$

Unfortunately, this planning task has exactly the same projections onto v and w as the planning task where the effect probabilities are changed to

$$\Pr_a(\{\}) = \frac{1}{16}$$
$$\Pr_a(\{v \mapsto 1\}) = \frac{3}{16} \quad \Pr_a(\{w \mapsto 1\}) = \frac{3}{16}$$
$$\Pr_a(\{v \mapsto 1, w \mapsto 1\}) = \frac{9}{16}$$

Consequentially, we cannot define a merging operation that reconstructs the original MDP of the planning task from the atomic projections, as there are already multiple MDPs with the same atomic projections. The problem arises because we do no longer remember the individual action outcomes in the atomic projections. Whereas in classical Merge-and-Shrink we must only synchronize those transitions with the same label when building the product, we have an additional layer of synchronization in the probabilistic setting: We must now also synchronize on the outcomes of an action, as every factor must be subject to the same outcome. Therefore, the planning model used to represent a factor must remember the individual outcomes and their probabilities.

Regarding shrinking strategies, previous considerations in classical planning dealt with variants of bisimulation (Nissim, Hoffmann, and Helmert 2011; Katz, Hoffmann, and Helmert 2012). Therefore, a natural candidate to investigate for shrinking strategies in probabilistic planning is probabilistic bisimulation (Larsen and Skou 1991), as well as possibly relaxed variations of this concept. Alternatively, traditional shrinking strategies are still applicable by considering the determinization of a factor. In particular, any bisimulation on the determinization is a probabilistic bisimulation, although not necessarily the coarsest bisimulation.

Finally, label reduction is a transformation that becomes useful when using bisimulation as a shrinking strategy. Often, using bisimulation only leads to a small reduction in the size of a factor, making this shrinking strategy barely effective in isolation. However, collapsing multiple labels into a common label usually has positive effects on bisimulation, as less labels mean less restrictions for the bisimulation relation. If the label reduction is exact, i.e. it does not introduce any spurious transitions in the represented transition system, then this transformation can even be safely applied without changing the cost-to-goal estimates the factors. It is therefore important to also consider label reduction when using a variant of bisimulation in the probabilistic setting.

Other Contributions and Research Ideas

Although this thesis mainly focuses on abstraction heuristics, any topic related to domain-independent heuristic construction for MaxProb and SSPs falls into the broader scope of the thesis. In a recent publication (Klößner et al. 2022a), we propose a theory of cost partitioning (Katz and Domshlak 2010) for SSPs. We found out that Trevizan, Thiébaux, and Haslum's projection occupation measure heuristic h^{pom} essentially computes an optimal cost-partitioning over atomic projections. This has major implications, as it means that optimal cost partitioning over PDB heuristics is theoretically superior to h^{pom}. An obvious candidate for future work is an experimental evaluation of different cost-partitioning techniques and different sets of combined heuristics.

Conclusion

Abstraction heuristics for MaxProb and SSPs are promising candidates to extend the landscape of admissible heuristics for these settings and enable more effective use of MDP heuristic search algorithms. So far, we focused in particular on Pattern Database heuristics, for which we observe a clear advantage over determinization-based heuristics. When combined with cost-partitioning, these heuristics even have the potential to outperform occupation measure heuristics, which are the most powerful heuristics for SSPs at present. In future work, we aim to transfer the Merge-and-Shrink framework to probabilistic planning and take a detailed look at various cost-partitioning techniques for SSPs.

References

Bertsekas, D. P.; and Tsitsiklis, J. N. 1991. An Analysis of Stochastic Shortest Path Problems. *Mathematics of Operations Research*, 16: 580–595.

Bonet, B.; and Geffner, H. 2003. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, 12–21. Trento, Italy: AAAI Press.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed Model Checking with Distance-Preserving Abstractions. In Valmari, A., ed., *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, 19–34. Springer-Verlag.

Guillot, M.; and Stauffer, G. 2020. The Stochastic Shortest Path Problem: A polyhedral combinatorics perspective. *European Journal of Operational Research*, 285(1): 148–158.

Hansen, E. A.; and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2): 35–62.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In Howe, A.; and Holte, R. C., eds., *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, 1007–1012. Vancouver, BC, Canada: AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 176–183. Providence, Rhode Island, USA: Morgan Kaufmann.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the Association for Computing Machinery*, 61(3): 16:1–16:63.

Katz, M.; and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12–13): 767–798.

Katz, M.; Hoffmann, J.; and Helmert, M. 2012. How to Relax a Bisimulation? In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 101–109. AAAI Press.

Klößner, T.; and Hoffmann, J. 2021. Pattern Databases for Stochastic Shortest Path Problems. In *Proceedings of the 14th Annual Symposium on Combinatorial Search (SOCS'21)*, 131–135. AAAI Press.

Klößner, T.; Pommerening, F.; Keller, T.; and Röger, G. 2022a. Cost Partitioning Heuristics for Stochastic Shortest Path Problems. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, 193–202. AAAI Press.

Klößner, T.; Steinmetz, M.; Torralba, À.; and Hoffmann, J. 2022b. Pattern Selection Strategies for Pattern Databases in Probabilistic Planning. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, 184–192. AAAI Press.

Klößner, T.; Torralba, Á.; Steinmetz, M.; and Hoffmann, J. 2021. Pattern Databases for Goal-Probability Maximization in Probabilistic Planning. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21)*, 80–89. AAAI Press.

Kolobov, A.; Mausam; Weld, D. S.; and Geffner, H. 2011. Heuristic Search for Generalized Stochastic Shortest Path MDPs. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*. AAAI Press.

Korf, R. E. 1997. Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. In Kuipers, B. J.; and Webber, B., eds., *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI'97)*, 700–705. Portland, OR: MIT Press.

Larsen, K. G.; and Skou, A. 1991. Bisimulation through probabilistic testing. *Information and Computation*, 94(1): 1–28.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, 1983–1990. AAAI Press/IJCAI.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*. AAAI Press/IJCAI.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-Based Heuristics for Cost-Optimal Planning. In Chien, S.; Do, M.; Fern, A.; and Ruml, W., eds., *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 226–234. AAAI Press.

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*, 362–367. AAAI Press.

Seipp, J.; and Helmert, M. 2013. Counterexample-guided Cartesian Abstraction Refinement. In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 347–351. Rome, Italy: AAAI Press.

Trevizan, F. W.; Thiébaux, S.; and Haslum, P. 2017. Occupation Measure Heuristics for Probabilistic Planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, 306–315. AAAI Press.

Trevizan, F. W.; Thiébaux, S.; Santana, P. H.; and Williams, B. 2017. I-dual: Solving Constrained SSPs via Heuristic Search in the Dual Space. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, 4954–4958. AAAI Press/IJCAI.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 352–359. Providence, Rhode Island, USA: Morgan Kaufmann.

Learning Hierarchical Abstractions for Efficient Taskable Robots – Dissertation Abstract

Naman Shah
Advisor: Prof. Siddharth Srivastava
School of Computing and Augmented Intelligence (SCAI),
Arizona State University,
Tempe, AZ, USA, 85281
shah.naman@asu.edu

Abstract

Although state-of-the-art hierarchical robot planning algorithms allow robots to efficiently compute long-horizon motion plans for achieving user desired tasks, these methods typically rely upon environment-dependent state and action abstractions that need to be hand-designed by experts. On the other hand, non-hierarchical robot planning approaches fail to compute solutions for complex tasks that require reasoning over a long horizon. My research addresses these problems by proposing an approach for learning abstractions and developing hierarchical planners that efficiently use learned abstractions to boost robot planning performance while providing strong guarantees of reliability.

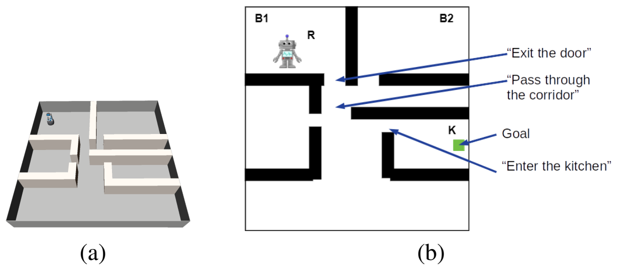


Figure 1: (a) A 3D model of a real life house hold environment. (b) A 2D map of the 3D environment. R shows the current position of the robot. Green rectangle shows the target position of the robot. The arrows show the hypothetical high-level actions that the robot must take to reach its goal from the current position that my work aims to learn automatically.

1 Introduction

Recent years have seen a sharp increase in the usage of robots in various areas such as manufacturing, household chores, and delivery. Such robots interact with their environments by moving their links around. To efficiently interact with their environments, a robot needs to compute a trajectory or a motion plan that takes the robot from its current configuration to its desired configuration. Generally, robots use sampling-based motion planners such as RRT (LaValle 1998) and PRM (Kavraki et al. 1996) to compute these motion plans that excel at computing short-horizon motion plans between pairs of configurations.

Complex tasks such as arranging a room or delivering items to different locations require robots to deal with changing configuration spaces and complex motion plans, requiring robots to reason over a long horizon. Typically, sampling-based motion planners fail to perform well in reasoning over a long horizon to compute such complex motion plans due to the infinite branching factor of the configuration space. This prevents robots from autonomously solving such complex tasks.

On the other hand, humans excel at such tasks which require extended reasoning owing to their capacity of abstracting information about the task. E.g, consider a human that has to arrange a dining table. To accomplish this task, humans would not think which joints they would move and by how much. However, they would think in high-level abstract actions such as "pick up the plate from the counter", "place the plate on the table", "place glass on the table", etc. Similarly, a person that has to reach the kitchen from a room (Fig. 1) in a house would "plan" using high-level actions such as "exit the door", "pass through the corridor", and "enter the kitchen". Such abstract actions allow humans to reason over a long horizon without considering the intricacies of the domain.

Hierarchical planning systems such as combined task and motion planning frameworks (Srivastava et al. 2014; Dantam, Kingston, and Chaudhuriand L. Kavraki 2018; Garrett, Lozano-Pérez, and Kaelbling 2020; Shah et al. 2020) use such high-level actions to guide the low-level (concrete) motion planning. These systems use hand-coded abstractions to generate high-level task specifications for these robot planning problems and use them to perform hierarchical planning. A domain expert is required to write these hand-coded abstractions, which limits the scope of the domains where these approaches can be applied.

Through my work, I aim to answer the following two crucial research questions: 1) Can we automatically learn hierarchical state and action abstractions for new environments and 2) can we efficiently use these abstractions to perform hierarchical robot planning? As part of my thesis, I aim to develop a set of approaches that answer these questions by automatically identifying hierarchical state and action abstractions for new environments and using them to perform hierarchical planning with various robots.

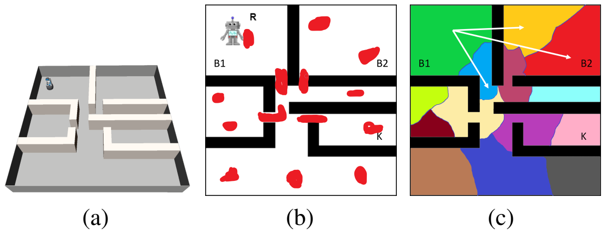


Figure 2: (a) An illustrative environment for a motion planning problem. The robot (R) is tasked to reach the kitchen (K). Red blobs in (b) show a set of candidate critical regions in the environment. Lastly, (c) shows an example of state abstraction. Each colored cell represents an abstract state. White arrows show a few abstract actions that take the robot from one abstract state to another abstract state.

Now in the rest of the paper, I present the approach that I would propose as part of my thesis, show preliminary results using the approach, and discuss a few related approaches.

2 Proposed Approach

To answer the research questions outlined in Sec. 1, I propose to learn hierarchical state and action abstractions by identifying regions in the environment that are critical for solving the given class of motion planning problems. E.g., consider a household environment (Fig. 2(a)).Here, the robot is currently in room B1. If the robot is tasked to bring a bottle of water from the kitchen (K) to the room (B1), then every motion plan accomplishing this task must pass through the doors and the passage. All these motion plans must take the robot to a configuration from which the robot is able to grasp the bottle and also to a configuration where the robot is holding the bottle in its gripper. Fig. 2(b) shows a few candidate critical regions for the given environment. This implies that these regions are loosely similar *landmarks* in the symbolic planning literature. But, contrary to landmarks, these regions are not a necessary condition to reach the goal. Molina, Kumar, and Srivastava (2020) define such regions as *critical regions* as proposes a method for identifying critical regions in an environment using a DNN.

I propose to learn hierarchical state and action abstractions using such automatically identified critical regions. Precisely, abstract states can be identified by constructing regions around these critical regions. Similarly, abstract actions can be automatically identified as transitions between these abstract states (similar to Fig. 2(c)). Once the abstract states and actions are identified, we can use them with a high-level planner to compute a high-level plan that can be refined into a motion plan using hierarchical planning.

One major technical challenge in this approach would be that the heuristic used to perform high-level planning would not be very informative. It would also fail to identify actions that the low-level planner would fail to refine due to obstacles in the environment. To overcome this issue, I propose to use a multi-source algorithm for high-level planning. Now, typically multi-source approaches can not be used with robot planning problems as we do not have any information about what the intermediate states would be. But, as we are identifying high-level abstract states automatically using critical regions, we can use these abstract states as candidate intermediate states for multi-source search.

On the other hand, a probabilistically-complete interleaved approach that tries to search for a high-level plan which has low-level refinements can also be used to perform hierarchical planning with such imprecise and lossy abstractions. This interleaved search approach would search for motion planning refinements for high-level actions while continually updating the high-level abstractions to compute accurate high-level solutions.

Now, I discuss some of the approaches that we have developed using these methods and their preliminary results.

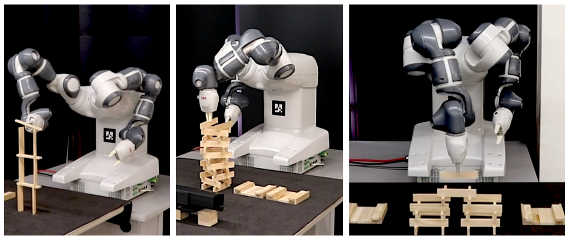


Figure 3: ABB YuMi builds Keva structures using a STAMP policy: 3π (left), twisted 12-level tower (center), and 3-towers (right).

3 Preliminary Results

We developed two algorithms that use the methods discussed in the previous approach for solving robot planning problems. The first method (Shah et al. 2020) uses interleaved search for performing combined task and motion planning in stochastic environments (Sec. 3.1) and the second approach -- ***H****ierarchical* ***A****bstraction-guided* ***R****obot* ***P****lanning (HARP)* (Shah and Srivastava 2022) -- performs hierarchical robot planning using automatically identified abstract states and actions and a multi-source planning algorithm (Sec. 3.2).

3.1 Stochastic Task and Motion Planning

Our work (Shah et al. 2020) presents an anytime *probabilistically complete* framework using the approach outlined in the previous section. It uses entity abstraction and stochastic shortest path (SSP) problems to formulate a *STAMP* problem for robots with stochastic actions. It performs an interleaved search to compute a high-level policy that also has valid low-level motion planning refinements. We use concretization functions called *generators* to refine each abstract action in the high-level solution. Our algorithm also continually updates the abstraction to generate more accurate high-level solutions if any action in the current high-level policy fails to admit a low-level refinement.

We evaluate our work in multiple settings where combined task and motion planning is necessary to compute feasible solutions. Refining each possible outcome in the solution tree can take a substantial amount of time. Here, we reduce the problem of selecting scenarios for refinement to a knapsack problem and use a greedy approach to prioritize

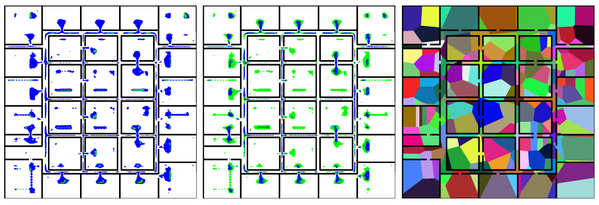


Figure 4: Critical regions and generated abstraction for a $\mathbf{4}$-DOF hinged robot. (a) and (b) shows the CRs predicted by the model. Blue regions in (a) show that model predicted the robot's base link to be horizontal while green regions show that the model predicted the robot's base link to be vertical. Blue regions in (b) show that the network predicted the hinge to be closer to $\mathbf{180}^\circ$ and green regions show that the network predicted it to be closer to $\mathbf{90}^\circ$ or $\mathbf{270}^\circ$. (c) shows 2D projection of the state abstraction generated by our approach though our approach does not need to explicitly generate these abstractions.

more likely outcomes for refinement. Our empirical evaluation shows that doing so allows the robot to start executing an action much earlier. Fig. 3 shows one of the test domains where the YuMi robot uses the task and motion policies computed using our framework to construct geometric structures using Keva planks.

3.2 Learning and Using Abstractions for Robot Planning

In this work (Shah and Srivastava 2022), we automatically learn hierarchical state and action abstractions and use them with a novel probabilistically-complete hierarchical planner to solve motion planning problems. These state and action abstractions are generated using automatically identified critical regions as described in Sec. 2. We use a custom robot-specific deep neural network to learn to identify critical regions in unseen environments for each degree of freedom of the robot and location of the robot's end-effector (base link for navigational problem) in the workspace. Our approach automatically generates this custom architecture using robot geometry and the number of degrees of freedom and using UNet (Ronneberger, Fischer, and Brox 2015) as the base architecture.

Once a set of critical regions is identified for a given configuration space and a robot, our approach generates abstract states by growing Voronoi cells around these critical regions. We call this structure a *region-based Voronoi diagram (RBVD)*. Each cell in an RBVD is an abstract state and transitions between these cells are abstract actions. We show that the abstractions generated using this approach are sound and fulfill the downward refinement property for holonomic robots.

We extend Beam search (Lowerre 1976) to develop a novel multi-source bidirectional high-level search algorithm to compute a set of high-level plans using the generated abstract states and actions. Our approach computes a set of high-level plans using this multi-source bidirectional Beam search and refines them simultaneously using a multi-source multi-directional Learn and Link Planner (LLP) (Molina, Kumar, and Srivastava 2020) while continually updating the heuristic for high-level search. We also prove that the overall planning algorithm is probabilistically complete.

We extensively evaluate our approach in a total of twenty different settings with four different robots that included holonomic as well as non-holonomic robots. Fig. 4 shows critical regions predicted by our learned model for a hinged robot with 4 degrees of freedom and abstract states generated by our approach. We empirically evaluate our approach against sampling-based motion planning algorithms such as RRT (LaValle 1998), PRM (Kavraki et al. 1996), and BiRRT (Kuffner and LaValle 2000) and learning-based motion planner (Molina, Kumar, and Srivastava 2020). Our exhaustive empirical evaluation shows that our approach that combines learning with hierarchical planning not only significantly outperforms state-of-the-art sampling-based motion planners but also outperforms learning-based LLP which does generate hierarchical abstractions and does not perform hierarchical planning.

This work learns state and action abstractions for motion planning problems and robots with deterministic actions. In the future, we plan to extend this work for robots with stochastic action dynamics and learn high-level states and actions for combined task and motion planning problems. Lastly, I discuss a few approaches related to my existing work.

4 Related Work

Combined Task and Motion Planning Hierarchical planning with abstractions has been used in multiple ways with automated planning to improve the efficiency of planners in order to compute plans that achieve complex goals. Some of the earliest planning systems such as ABSTRIPS (Sacerdoti 1974) and ALPINE (Knoblock 1990) used abstractions with symbolic planning problems defined in STRIPS representations to perform hierarchical planning. Approaches such as FF (Hoffmann 2001), HSP2.0 (Bonet and Geffner 2001), and GraphPlan (Blum and Furst 1997) uses abstractions to solve a relaxed problem in order to automatically synthesize heuristics for planning.

A significant number of approaches have been developed to solve task and motion planning (TAMP) problems in recent years. Major works on TAMP can be categorized into three categories: *1)* approaches that use symbolic planners to guide motion planning (Cambon, Alami, and Gravot 2009), *2)* approaches that extend high-level representations to simultaneously search high-level plans along with continuous parameters (Garrett, Lozano-Pérez, and Kaelbling 2020), and *3)* approaches that use interleaved search for valid high-level plans with low-level refinements for its actions (Srivastava et al. 2014; Dantam, Kingston, and Chaudhuriand L. Kavraki 2018). These approaches focus on solving TAMP problems in deterministic environments where robots are expected to carry out their tasks accurately, while our approach provides a method to consider stochastic actions while computing task and motion policies to allow the robot to efficiently perform its tasks in the real world.

Learning for Motion Planning Multiple approaches use statistical learning to guide motion planning. Ichter, Harrison, and Pavone (2018) and Kumar et al. (2019) use *CVAE* to learn sampling distributions for motion planning. Molina, Kumar, and Srivastava (2020) use an image-based approach to learn sampling distributions for path-planning problems. These approaches use learning to bias the sampling distribution for sampling-based motion planning, while our approach focuses on learning state and action abstractions and using them efficiently to perform hierarchical planning.

Learning Abstractions While a lot of approaches (Blum and Furst 1997; Bonet and Geffner 2001; Knoblock 1990; Cambon, Alami, and Gravot 2009; Garrett, Lozano-Pérez, and Kaelbling 2020; Shah et al. 2020) have tried using abstractions to perform automated planning, not a lot of them have tried learning it. Konidaris, Kaelbling, and Lozano-Perez (2018) propose an approach that learns high-level action descriptions of low-level behaviors by computing sets of regions reachable by those behaviors. Contrary to their approach that requires a detailed description of reachability for low-level composite actions which are extremely difficult to obtain for complex AI agents, my thesis aims to develop an approach that solely requires low-level observations.

Chitnis et al. (2020) use CNNs to learn context-specific regions of the state space. Silver et al. (2020) uses GNNs to predict importance for each object in the environment for classical planning problems. These approaches use abstraction to reduce the planning space, but they still need handcrafted action descriptions in order to perform planning. Silver et al. (2021) propose an approach that uses low-level transitions along with high-level vocabulary to learn the symbolic representation of these actions for combined task and motion planning. Their approach requires an abstraction that is sufficient to represent low-level actions that the proposed work aims to learn.

References

Blum, A. L.; and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1-2): 281–300.

Bonet, B.; and Geffner, H. 2001. Heuristic search planner 2.0. *AI Magazine*, 22(3): 77–77.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *IJRR*, 28: 104–126.

Chitnis, R.; Silver, T.; Kim, B.; Kaelbling, L. P.; and Lozano-Perez, T. 2020. CAMPs: Learning Context-Specific Abstractions for Efficient Planning in Factored MDPs. *arXiv preprint arXiv:2007.13202*.

Dantam, N.; Kingston, Z.; and Chaudhuriand L. Kavraki, S. 2018. An incremental constraint-based framework for task and motion planning. *IJRR*, 37(10): 1134–1151.

Garrett, C.; Lozano-Pérez, T.; and Kaelbling, L. 2020. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proc. ICAPS*.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine*, 22(3): 57–57.

Ichter, B.; Harrison, J.; and Pavone, M. 2018. Learning sampling distributions for robot motion planning. In *Proc. ICRA*.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE transactions on Robotics and Automation*, 12(4): 566–580.

Knoblock, C. A. 1990. Learning Abstraction Hierarchies for Problem Solving. In *AAAI*, 923–928.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Kuffner, J. J.; and LaValle, S. M. 2000. RRT-connect: An Efficient Approach to Single-Query Path Planning. In *Proc. ICRA, 2000.*

Kumar, R.; Mandalika, A.; Choudhury, S.; and Srinivasa, S. 2019. LEGO: Leveraging Experience in Roadmap Generation for Sampling-Based Planning. In *Proc. IROS*.

LaValle, S. M. 1998. Rapidly-Exploring Random Trees: A New Tool for Path Planning.

Lowerre, B. T. 1976. *The Harpy Speech Recognition System.* Carnegie Mellon University.

Molina, D.; Kumar, K.; and Srivastava, S. 2020. Identifying Critical Regions for Motion Planning using Auto-Generated Saliency Labels with Convolutional Neural Networks. In *Proc. ICRA*.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Proc. MICCAI, 2015*.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2): 115–135.

Shah, N.; Kala Vasudevan, D.; Kumar, K.; Kamojjhala, P.; and Srivastava, S. 2020. Anytime Integrated Task and Motion Policies for Stochastic Environments. In *Proc. ICRA*.

Shah, N.; and Srivastava, S. 2022. Using Deep Learning to Bootstrap Abstractions for Hierarchical Robot Planning. *arXiv preprint arXiv:2202.00907*.

Silver, T.; Chitnis, R.; Curtis, A.; Tenenbaum, J. B.; Lozano-Pérez, T.; and Kaelbling, L. P. 2020. Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks. *CoRR*, abs/2009.05613.

Silver, T.; Chitnis, R.; Tenenbaum, J.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Learning Symbolic Operators for Task and Motion Planning. *arXiv preprint arXiv:2103.00589*.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. A Modular Approach to Task and Motion Planning with an Extensible Planner-Independent Interface Layer. In *Proc. ICRA*.

Modeling Assistance for AI Planning From the Perspective of Model Reconciliation – Dissertation Abstract

[1]Songtuan Lin
Supervisors: [1]Pascal Bercher, [2]Gregor Behnke, [1]Alban Grastien
[1]School of Computing, The Australian National University, Canberra, Australia
[2]Institute for Logic, Language and Computation, University of Amsterdam, Amsterdam, Netherlands
[1]firstname.secondname@anu.edu.au
[2]behnkeg@informatik.uni-freiburg.de

Abstract

Providing modeling assistance to domain modelers is a prominent challenge in incorporating humans into planning processes. Many efforts have been devoted to this direction in classical planning, however, only few works have been done in hierarchical planning. In this thesis, we will study a methodology for providing modeling assistance in HTN planning, which is the most commonly used hierarchical planning framework. Particularly, we will address two bottleneck problems for this purpose, namely domain model validation and domain model refinements. For the former one, we propose an approach based on plan verification, and for the latter, we view it as a model reconciliation problem and will study a novel approach for solving it.

Introduction

Human-AI interaction has evolved as the frontier of the research on automated planning in the last decades for its capability of solving complex problems. One prominent challenge in this direction faced by the community is providing *modeling assistance* from which a domain engineer can benefit. Designing a planning domain, also known as knowledge engineering in planning and scheduling (KEPS) (McCluskey, Vaquero, and Vallati 2017) has been shown to be a difficult task, as evidenced by the establishment of the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS), whereas it is also a mandatory process in automated planning. Thus, providing modeling assistance to domain engineers is also in deep need.

Two bottleneck problems in engineering planning domains are domain validations and domain refinements, that is, deciding whether a domain is functioning correctly and how to refine the domain if that is not the case. A significant number of efforts have been made attempting to address these two problems in order to provide modeling assistance in classical planning, for example by Lindsay et al. (2020). However, only few have been done in hierarchical planning, e.g., by Olz et al. (2021). Consequently, in this thesis, we intend to deal with those two problems in the context of hierarchical planning for the purpose of providing modeling assistance for hierarchical domain modeling. The hierarchical planning framework we are concerned with is Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1996; Geier and Bercher 2011; Bercher, Alford, and Höller 2019), as it is the most widely used one in recent years. Additionally, although our focus is modeling assistance for hierarchical planning, the approach we study here is also compatible with classical planning, which thus generalizes the scope of our work.

Our approach for domain model validations is via plan verification (Behnke, Höller, and Biundo 2015). More concretely, we provide a plan to a planning problem in the domain we want to validate which is supposed to be a solution, and then we verify whether this is the case. This stems from how a software is validated in practice, i.e., via providing test cases to see whether the software's outputs are correct. In our context, a provided plan serves as a test case, and the failed verification indicates that there are some flaws within the domain model.

For domain refinements, we are essentially investigating the scenario where a given plan, which serves as a test case, turns out not to be a solution to a planning problem in a domain, and we want to refine the domain so that it can be. This can be viewed as a model reconciliation problem (Chakraborti et al. 2017; Chakraborti, Sreedharan, and Kambhampati 2020; Sreedharan, Chakraborti, and Kambhampati 2021) where we intend to reconcile a flawed domain model (engineered by a domain modeler) to the ground truth one (which is unfortunately unknown). However, the underlying assumption of the existing model reconciliation approaches (Chakraborti et al. 2017; Sreedharan et al. 2019; Sreedharan, Chakraborti, and Kambhampati 2021) is that there are two domain models given (one is a human's mental model, and the other is the real one used by a planner/robot). This is however not the case in our scenario where we only have one flawed domain model. Consequently, we will study a new model reconciliation approach in the thesis for serving our scenario, i.e., changing the domain model so that the given plan will be a solution.

Beyond the scope of modeling assistance, the idea of changing a domain model to make a plan be a solution can also serve as an approach for providing a contrastive explanation (Miller 2019) about why a given plan is not a solution and can henceforth be employed in building Explainable AI Planning (XAIP) systems.

The objective of this paper is to outline the contents of the thesis and give an introduction to our methodology for providing modeling assistance. For this purpose, we will first present the HTN formalism which we based upon and, on top of this, give an example to illustrate how our approach can be used to provide modeling assistance. Afterwards, we will present theoretical foundations for our methodology and abstractly describe how our approach will be implemented. We start by presenting the HTN formalism employed.

HTN Formalism

Thus far there exist various HTN formalisms. Those of major importance are the ones by Erol, Hendler, and Nau (1996), Geier and Bercher (2011), and Bercher, Alford, and Höller (2019), where the last two only differ in syntax, whereas the first one features more constraints over task networks (which we will introduce later on). In the thesis, we adhere to the one by Bercher, Alford, and Höller (2019), as it is the one which most complexity investigations are based on in the last decade.

One foundation for the HTN formalism is the concept of task networks, which is a set of *labeled* task names with a partial order defined over them. The task names in a task network are further categorized as being primitive or compound. A primitive task name, also called an action, is associated with its preconditions, add, and delete list, each of which consists of a set of propositions. The add list together with the delete list of an action is called the effects of the action. On the other hand, a compound task name can be refined (decomposed) into a task network by some method.

Another fundamental concept in the HTN formalism is the decomposition of a task network. Informally, a task network tn is said to be decomposed into another one tn' if tn' is obtained from tn by substituting a compound task in tn with a task network into which this compound task is decomposed by a method.

An HTN planning problem is constituted of three components: a domain, an initial task network, and an initial state. The domain consists of a finite set of propositions, a finite set of actions, a finite set of compound tasks, a finite set of methods, and a function mapping each action in the domain to its preconditions and effects. A state in an HTN planning problem is a set of propositions which describes the world.

A plan is a solution to an HTN planning problem if it is a refinement of the initial task network and consists of solely actions, meaning that it is a primitive task network obtained from the initial task network by a sequence of decompositions. Further, this refined task network must possess a linearisation (i.e., an action sequence) which is executable in the initial state, i.e., the plan is executable in the initial state. In this paper, unless otherwise specified, a plan is referred to a *partially ordered primitive task network*. For formal definitions, we refer to the work by Bercher, Alford, and Höller (2019).

Domain Validation via Verification

Having presented the HTN planning formalism used in the thesis, now we would like to introduce more technical details about the thesis in the following sections, including the theoretical foundations and implementation techniques. We begin with domain model validation.

As mentioned earlier, our way to validate a domain model is by verifying whether a plan, which serves as a test case, is a solution to a planning problem in the domain. The core challenge in this step is clearly solving a plan verification problem (Behnke, Höller, and Biundo 2015). Hence, we will also exploit a plan verifier in our implementation.

There are currently two main approaches for plan verification in HTN planning. One is by transforming a plan verification problem into a SAT problem and exploiting a SAT solver to solve the problem, i.e., the SAT-based approach (Behnke, Höller, and Biundo 2017). The other one exploits the similarities between an HTN planning problem and an attribute grammar and regards a plan verification problem as a parsing problem, i.e., the parsing-based approach (Barták, Maillard, and Cardoso 2018; Barták et al. 2020). Thus far, the empirical evaluations show that the parsing-based approach outperforms the SAT-based approach. However, the SAT-based approach is implemented in JAVA in a deprecated version of the planning system called PANDA$_3$, and recently, a new version called PANDA$_{\text{pi}}$ was developed in C++. Hence, despite the underperformance of the SAT-based approach compared with the parsing-based one, we are still planning to reimplement it in C++, integrate it into the new version of PANDA, and again compare its performance with the parsing-based approach. Further, we also plan to develop a new SAT approach for plan verification based upon solution order graphs (SOGs) proposed by Behnke, Höller, and Biundo (2019), which are used in solving an HTN planning problem via encoding it as a SAT formula and is proved that such a graph can significantly reduce the number of SAT clauses and variables required. We will also compare the performance of this new SAT approach with the reimplemented one and the parsing-based one.

Further, we will also pay extra attentions to plan verification for totally ordered (TO) HTN planning problems. In contrast to plan verifications for partially ordered (PO) HTN planning problems, which have been shown to be NP-complete (Behnke, Höller, and Biundo 2015), a TOHTN plan verification problem is computationally easy. This is because any TOHTN planning problem can be captured by a context-free grammar (Höller et al. 2014), and hence, a TOHTN plan verification problem can essentially be viewed as a membership decision problem for context-free grammars. Nevertheless, the CYK algorithm for the membership decision problem for context-free grammars cannot be directly employed in TOHTN, due to additional constraints a TOHTN planning problem might have, e.g., method preconditions. Barták et al. (2021) extended the parsing-based approach to adapt to totally ordered planning problems, whereas the extended approach still relies on blind search. Consequently, we are going to improve the approach by taking advantage of the CYK algorithm and conduct empirical evaluations to compare the performance of those two.

Domain Validations via Model Checking

Apart from providing test cases, another possible way to validate a domain model is by model checking (Baier and Katoen 2008) which has already been successfully used in verifying whether a system, e.g., an Information and Communication Technology (ICT) system, preserves certain properties. Hence, we can also employ this approach to check, e.g., whether the plans produced in a domain model satisfy certain constraints.

In model checking, properties in need of verification are normally captured by an LTL formula. It is thus natural to think of incorporating LTL directly into the HTN formalism, e.g., using an LTL formula to serve as the goal description of an HTN planning problem and using such a fusion to catch those domains which fail to produce plans that satisfy the LTL formula. As a starting point, we have investigated the expressiveness power of LTL in conjunction with the HTN formalism as well as the STRIPS (classical) formalism (Lin and Bercher 2022). We believe those results can serve as the theoretical foundations for applying model checking to domain model validation.

Domain Refinements via Model Reconciliation

The major concern of the thesis is domain refinements, provided that a test case, i.e., a plan, given to an HTN planning problem built upon a domain model fails. Our goal is to refine the domain model so that the plan can become a solution. As mentioned in the introduction, although this is essentially a model reconciliation problem, existing model reconciliation approaches (Chakraborti et al. 2017; Sreedharan et al. 2019; Sreedharan, Chakraborti, and Kambhampati 2021) are not applicable here, due to the inconsistency between the underlying assumption of those approaches and our scenario where we only have a flawed domain model in hand, but the existing approaches demand two models, i.e., a human's mental model and the actual model employed by a planner/robot. Consequently, in the thesis, we will propose a novel method for solving our model reconciliation problem.

Framework

As our ultimate goal is to change a domain model so that a given plan will be a solution in the updated model, we shall first specify what changes are allowed to be imposed to the domain. We first observe that, according to the solution criteria of HTN planning problems, there are mainly two reasons for why a plan is not a solution to a planning problem:

1) The plan cannot be obtained from the initial task network via decompositions, and
2) the plan is not executable in the initial state.

We can fix the first problem via refining methods in a domain model, and for the second one, we can change actions' preconditions and effects. One might further notice that those two classes of changes are independent of each other, because changing methods will not affect the executability of a plan and changing actions will not affect the decomposition hierarchy as well from which a plan is obtained.

More concretely, we will define four change operations targeted at refining methods in a domain model, namely,

1) adding an action to a method,
2) removing an action from a method,
3) adding an ordering constraint (between two actions) to a method, and
4) removing an ordering constraint (between two actions) from a method.

The reason for restricting ourselves to those four operations is that we can implement other high-level changes targeted at turning a non-solution plan to a solution, e.g., adding a compound task to a method, in terms of those defined.

For changing actions' preconditions and effects, we are concerned with three operations:

1) adding a proposition to an action's add list,
2) removing a preposition from an action's preconditions,
3) and removing a preposition from an action's delete list.

We do not consider any other change here, e.g., adding a proposition to an action's preconditions, because it can only increase the possibility that a plan is not executable.

Theoretical Foundations – Complexity

Before developing a methodology for accomplishing domain refinements via the operations described above, we are interested in finding out how hard that might be, that is, we are going to study the computational complexity of turning a non-solution plan into a solution via domain refinements. For the purpose of complexity investigation, we will use *decision languages* to describe the scenario, that is, we want to decide, given an HTN planning problem and a plan, whether there exist a way to refine the domain model of the planning problem via the operations described above such that the given plan can be a solution to the planning problem.

Note that the decision problem we are studying here is *not* exactly the same as the actual domain refinement (model reconciliation) problem. The former one only demands a *yes* or *no* answer, whereas the actual domain refinement problem we want to solve demands an exact sequence of refinement operations that can turn the plan into a solution as an output. In spite of that, the complexity results for the decision problem quantify the computational efficiency of the actual domain refinement problem. Further, the complexity investigation (on the decision problem) is also a way to identify the sources that make the actual refinement problem hard, which can thus point out the direction of developing an actual methodology for accomplishing domain refinements.

Since changing methods and changing actions' preconditions and effects are orthogonal, it is safe for us to study these two classes of changes independently. Concretely, we will first study the complexity of deciding the existence of *method* refinement operations that turn the plan into a solution by assuming that the plan is already executable in the initial state. Under this assumption, the decision problem can be rephrased as: Given an HTN planning problem and a plan, is there a way to change the methods in the domain of the planning problem such that the given plan can be obtained from the initial task network via decompositions.

Our earlier work has shown that deciding whether such changes exist is already $\mathbf{NP}$-complete in TOHTN (Lin and Bercher 2021a), independent of what method refinement operations are allowed, and later on we extended the results from a TO setting to a PO setting (Lin and Bercher 2021b). Hence, unless $\mathbf{P} = \mathbf{NP}$, there exist no polynomial algorithms that can solve the refinement problem in poly-time.

Thus far we restrict an input test case to one single plan. In practice, a domain modeler may supply a plan together with the decomposition hierarchy that is supposed to lead to the plan as a test case. The decision problem asking for the existence of method refinement operations with regard to this scenario is similar to the previous one except that now we have an additional decomposition hierarchy as the input. Though the problem is still NP-complete in general, we identified several special cases in $\mathbf{P}$ where an input decomposition hierarchy satisfies certain constraints, e.g., every method in the hierarchy decomposes a unique compound task (Lin and Bercher 2021a), and by exploiting those special cases, we were able to find all hardness sources of the decision problem where a decomposition hierarchy is not given (Lin and Bercher 2021b).

We have also studied the complexity of the problem of deciding whether there exists a way to turn a plan into a refinement of the initial task of a planning problem by applying at most k method refinement operations, where $k \in \mathbb{N}$ is a given number. This decision problem corresponds to the domain refinement problem demanding the minimum number of change operations that turn a plan into a solution. Given the previous results, this k-existence decision problem is unsurprisingly $\mathbf{NP}$-complete as well, independent of whether a decomposition hierarchy is given as an additional input.

Next we introduce the complexity of deciding the existence of action refinement operations which turn a plan into a solution, by assuming that a given plan is an action sequence and is a refinement of the initial task network of a planning problem. Clearly, the answer to this decision problem is always *yes*, because we can at least empty the preconditions of each action in the given action sequence. Therefore, what we are really interested in here is the complexity of the k-existence decision problem asking whether the plan can become executable by applying at most k refinement operations. The complexity varies in what operations are allowed. The problem is $\mathbf{NP}$-complete if adding propositions to actions' add lists is allowed, otherwise it is in $\mathbf{P}$.

Implementation – A SAT-Baseed Approach

Lastly, we will develop a methodology for actually solving the domain refinement problem. Our first attempt toward this will be to encode the refinement problem as the SAT problem. In our previous works (Lin and Bercher 2021b), we have shown that two main sources that make the refinement problem $\mathbf{NP}$-hard are: 1) the non-deterministic choices of the decomposition hierarchy that leads to the given plan, i.e., the plan verification problem, and 2) the non-deterministic choices of the methods that are in need of changing. The natural properties of these hardness sources make SAT approaches the best tool to encode the non-determinism. As an example, we can use one SAT variable $x_{a \to m}$ to indicate whether the action a is added to the method m. Additionally, we can also exploit the procedure by Behnke, Höller, and Biundo (2017) that transforms the plan verification problem into the SAT problem. Further, since both the refinement problem and the SAT problem are both $\mathbf{NP}$-complete, it is reasonable to assume that transforming one into the other will not add too many overheads.

Our another concern is how to do the empirical evaluation. We are current considering two approaches. One is to use the existing *invalid* plans. We will use our implementation to alter the respective domains to make those plans become valid. The metric for this evaluation approach will thus be the number of successful instances in a certain time frame, i.e., how many instances can be successfully solved (refined) in a given time.

Alternatively, we are also considering using the existing *valid* plans and randomly changing the respective HTN domains to make those plans become invalid. Afterward, we will employ our implementation to change the domains and make those plan valid again. The metric for this evaluation approach is to estimate, for each domain, the similarity between the original unmodified version (the ground truth) and the version refined by our implementation.

Implementation – A Planning-Based Approach

Another implementation approach we are concerned with is by encoding the domain refinement problem as an HTN planning problem. The idea stems from the similarity between the plan verification problem and the domain refinement problem. Höller et al. (2022) have proposed the approach of transforming a plan verification problem into an HTN planning problem. The empirical evaluation shown that this approach significantly outperforms others, i.e., the SAT-based and parsing based approach. Hence, we are also planning to extend this approach to support model refinements and compare its performance with the SAT approach.

Future Work

In the thesis, we only concern grounded HTN planning models, i.e., models without variables. One straightforward extension of the thesis is generalizing our approach to lifted models, i.e., models with variables. In fact, such an extension is of great importance because most domain description languages for HTN planning, e.g., HDDL (Höller et al. 2020), are designed for lifted models.

Another direction of extending our work is to take into account *negative* test cases by assuming that a domain modeler provides a plan which is *not* supposed to be a solution, and we want to refine the domain to ensure this.

Conclusion

The thesis aims at addressing two major problems in providing modeling assistance in AI planning, namely, domain model validation and domain model refinements. The approach for domain validation is by plan verification. We assume that a plan which serves as a test case is provided to a planning problem built on a domain model that need to be validated, and a failed verification indicates that domain model is flawed. For domain refinements, we view it as a model reconciliation problem where given a plan and an HTN planning problem, we want to refine the domain model such that the plan can be a solution to the planning problem, and we will propose a novel approach for solving this reconciliation problem,

References

Baier, C.; and Katoen, J. 2008. *Principles of model checking*. MIT.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2018. Validation of Hierarchical Plans via Parsing of Attribute Grammars. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling, ICAPS 2018*, 11–19. AAAI.

Barták, R.; Ondrcková, S.; Behnke, G.; and Bercher, P. 2021. On the Verification of Totally-Ordered HTN Plans. In *Proceedings of the 33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021*, 263–267. IEEE.

Barták, R.; Ondrcková, S.; Maillard, A.; Behnke, G.; and Bercher, P. 2020. A Novel Parsing-based Approach for Verification of Hierarchical Plans. In *Proceedings of the 32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020*, 118–125. IEEE.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the Complexity of HTN Plan Verification and Its Implications for Plan Recognition. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling, ICAPS 2015*, 25–33. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This Is a Solution! (... But Is It Though?) - Verifying Solutions of Hierarchical Planning Problems. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling, ICAPS 2017*, 20–28. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2019. Bringing Order to Chaos - A Compact Representation of Partial Order in SAT-Based HTN Planning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, 7520–7529. AAAI.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, 6267–6275. IJCAI.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2020. The Emerging Landscape of Explainable Automated Planning & Decision Making. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, 4803–4811. IJCAI.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI 2017*, 156–163. IJCAI.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Annals of Mathematics and Artificial Intelligence*, 18(1): 69–93.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, 1955–1961. IJCAI.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *Proceedings of the 21st European Conference on Artificial Intelligence, ECAI 2014*, 447–452. IOS.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, 9883–9891. AAAI.

Höller, D.; Wichlacz, J.; Bercher, P.; and Behnke, G. 2022. Compiling HTN Plan Verification Problems into HTN Planning Problems. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI.

Lin, S.; and Bercher, P. 2021a. Change the World - How Hard Can that Be? On the Computational Complexity of Fixing Planning Models. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, 4152–4159. IJCAI.

Lin, S.; and Bercher, P. 2021b. On the Computational Complexity of Correcting HTN Domain Models. In *Proceedings of the 4th ICAPS Workshop on Hierarchical Planning, HPlan 2021*, 35–43.

Lin, S.; and Bercher, P. 2022. On the Expressive Power of Planning Formalisms in Conjunction with LTL. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI.

Lindsay, A.; Franco, S.; Reba, R.; and McCluskey, T. L. 2020. Refining Process Descriptions from Execution Data in Hybrid Planning Domain Models. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling, ICAPS 2020*, 469–477. AAAI.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *Proceedings of the 9th Knowledge Capture Conference, K-CAP 2017*, 14:1–14:8. ACM.

Miller, T. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267: 1–38.

Olz, C.; Wierzba, E.; Bercher, P.; and Lindner, F. 2021. Towards Improving the Comprehension of HTN Planning Domains by Means of Preconditions and Effects of Compound Tasks. In *Proceedings of the 10th Workshop on Knowledge Engineering for Planning and Scheduling, KEPS 2021*.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2021. Foundations of explanations as model reconciliation. *Artificial Intelligence*, 301: 103558.

Sreedharan, S.; Hernandez, A. O.; Mishra, A. P.; and Kambhampati, S. 2019. Model-Free Model Reconciliation. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, 587–594. IJCAI.

Neural Network Action Policy Verification via Predicate Abstraction – Dissertation Abstract

Marcel Vinzent
Supervisor: Jörg Hoffmann
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
{vinzent, hoffmann}@cs.uni-saarland.de

Abstract

Neural networks (NN) are an increasingly important representation of action policies. With their application for real-time decision-making in safety critical areas, like, e.g., autonomous driving, it arises the need to gain trust in the applied policies. The ultimate goal to gain this trust is through formal verification of the policy-induced behavior. This is a challenging endeavor as it compounds the state space explosion with the difficulty of analyzing even single NN decision episodes.

In our work, we make a contribution to cope with this challenge. We approach safety verification through (over-approximating) abstract reachability analysis. We compute predicate abstractions of the policy-restricted state space; expressing the abstract transition computation as a satisfiability modulo theories (SMT) problem, and devise a range of algorithmic enhancements to avoid costly calls to SMT.

First empirical results show that our approach can outperform competing approaches. Future work will further enhance the technique and extend it to support probabilistic settings.

Introduction

Neural networks (NN) are an increasingly important representation of action policies; in particular for real-time decision making in dynamic environments. The vision is elegant and simple: The NN policy can be learned in advance and at run-time a single (computationally efficient) call to the NN suffices to select an action. But how to verify that such a policy is safe? Leading such a proof is potentially very hard as it compounds the state space explosion with the difficulty of analyzing even single NN decision episodes.

In our work, we contribute to cope with this challenge. We tackle non-deterministic state spaces over bounded-integer state variables. Given an NN action policy π, a **start condition** ϕ_0, and an **unsafety condition** ϕ_U, we verify whether a state $s_U \models \phi_U$ is reachable from a state $s_0 \models \phi_0$ under π.

We approach safety verification via the extension of **predicate abstraction (PA)** (Graf and Saïdi 1997; Ball et al. 2001) to deal with NN action policies. PA is defined through a set $\mathcal{P}$ of **predicates**, where each $p \in \mathcal{P}$ is a linear constraint over the state variables (e.g. $x \leq 5$ or $y \geq z$). Abstract states are characterized by truth value assignments over $\mathcal{P}$, grouping together all concrete states that induce the same truth values. Transitions are over-approximated to preserve all possible behaviors. For our purpose of policy safety verification, we are interested in the predicate abstraction of the **policy-restricted** state space Θ^π, i.e., the subgraph of Θ induced by π. We refer to the predicate abstraction of Θ^π as **policy predicate abstraction (PPA)** $\Theta^\pi_\mathcal{P}$. We build the fragment of $\Theta^\pi_\mathcal{P}$ reachable from ϕ_0. If ϕ_U is not reached then policy π is proven to be safe.

To compute PA, one repeatedly needs to decide whether there is a transition from abstract state A to abstract state A' under some action a: *does there exist a state $s \in A$ s.t. executing a in s results in $s' \in A'$?* This transition problem is routinely addressed using SMT solvers such as $Z3$ (de Moura and Bjørner 2008). Now, to compute the PPA $\Theta^\pi_\mathcal{P}$, one additionally needs to check whether $\pi(s) = a$, i.e., whether the policy selects a in s. Solving this over and over again via calls to SMT quickly becomes infeasible due to the complex structure of neural networks.

In our current work, we hence have devised a range of algorithmic enhancements, leveraging relaxed tests to avoid costly calls to SMT. Most importantly, continuous relaxation of the discrete state variables enables to plug in state-of-the-art SMT solvers tailored to NN (Katz et al. 2017, 2019). We also have devised a method using branch-and-bound around relaxed tests to avoid exact calls to SMT altogether, as well as a method that simplifies SMT calls via information obtained through NN analysis. Empirical results so far show that our approach can outperform competing approaches and that our algorithmic enhancements are required for practicality (Vinzent, Steinmetz, and Hoffmann 2022).

Future research will investigate techniques for automatic abstraction refinement, specifically via counter example guided abstraction refinement (Clarke et al. 2000). Here, we will develop refinement approaches to rule out spuriousness related to the policy. Additionally, we will leverage further NN analysis techniques to enhance the efficacy of our approach, e.g., symbolic propagation (Li et al. 2019) and adversarial attack methods (Goodfellow, Shlens, and Szegedy 2015). We also plan to compute quantitative safety results via value iteration (Givan, Leach, and Dean 1997) in the abstract state space of probabilistic systems.

Related Work

There has been remarkable progress on analyzing individual **NN decision episodes**. In our work so far, we query *Marabou* (Katz et al. 2019), which extends Simplex by a *lazy* case splitting approach to handle piecewise-linear activation functions. *Marabou* utilizes a symbolic interval propagation approach (Wang et al. 2018b) which leverages interval arithmetic to propagate bounds on the NN input through the network. Other related work (Ehlers 2017) leverages non-symbolic bound propagation for linear relaxation of NN activation nodes. Follow-up work (Wang et al. 2018a) then combines symbolic bound propagation with linear relaxation. Recent work (Li et al. 2019) extends the symbolic propagation approach to general abstract domains; e.g., also zonotopes rather than simple interval bounds. In the context of our work, such bound propagation techniques can be utilized to over-approximate the possible NN policy behavior. Additionally, adversarial attack methods (e.g. (Goodfellow, Shlens, and Szegedy 2015)), can in principle be adapted to certify transition existence.

The verification of **NN decision sequences** – NN policies executed in an environment – is in its infancy. For software verification, there is initial work on abstract interpretation of programs with calls to NN sub-procedures (Christakis et al. 2021). Gros et al. (2020b) apply statistical model checking to statistically verify NN action policies, but this approach is limited to small numbers of start states as these need to be explicitly enumerated. Tran et al. (2019) use star sets to exactly compute respectively over-approximate reachable sets of a system controlled by a neural network; focusing on linear systems however.

The verification of NN controllers through polynomial approximation has been studied in several works (e.g. (Huang et al. 2019; Ivanov et al. 2021)). These approaches are conceptually very different to our work. Specifically, Ivanov et al. (2021) focus on NN with sigmoid/tanh activation functions[1] which allows to compile the NN behavior into a hybrid system – whose composition with the controlled system is amenable to known verification techniques. On the one hand, this immediately enables to perform verification for a broad range of, possibly complex, systems. On the other hand, such verification is bound to approximation. In contrast, our approach is tailored to piecewise-linear activation functions (e.g. ReLU) and leverages NN-specific techniques which enable for exact analysis.

In a context closer to AI sequential decision making, recent work (Akintunde et al. 2018, 2019) explores the use of MIP encodings for bounded-length verification of NN controlled systems. The approach is technically rather different to ours. While we compute individual (abstract) transitions, in bounded-length verification one checks fixed-size path existence, i.e., transition sequences, via a monolithic MIP encoding; iteratively for paths of increasing sizes. This requires to solve encodings of stepwise increasing cost. Empirical evaluations show that for our purposes the practicability of bounded-length verification is rather limited (Vinzent, Steinmetz, and Hoffmann 2022).

Besides verification, there are also **other techniques to gain trust** in an NN action policy, e.g., safe reinforcement learning (see (García and Fernández 2015) for an overview), especially shielding (e.g. (Alshiekh et al. 2017)); or testing (e.g. (Steinmetz et al. 2022)); or any manner of explainable AI that may help to elucidate the NN's action decisions, in particular visualization (e.g. (Gros et al. 2020a)). In fact, beyond verification, our policy predicate abstraction technique might also turn out to be useful for policy visualization purposes; enabling zooming in the policy-restricted state space based on abstraction predicates.

Background

State Space Representation. A state space is a tuple $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ of **state variables** $\mathcal{V}$, **action labels** $\mathcal{L}$, and **operators** $\mathcal{O}$. The domain D_v of each variable $v \in \mathcal{V}$ is a non-empty bounded integer interval. Exp denotes the set of **linear integer expressions** over $\mathcal{V}$ (i.e., of the form $d_1 \cdot v_1 + \cdots + d_r \cdot v_r + c$ with $d_1, \ldots, d_r, c \in \mathbb{Z}$). C denotes the set of **linear integer constraints** over $\mathcal{V}$, (i.e., of the form $e_1 \bowtie e_2$ with $\bowtie \in \{\leq, =, \geq\}$ and $e_1, e_2 \in Exp$), and all Boolean combinations thereof. An **operator** $o \in \mathcal{O}$ is a tuple (g, l, u) with **label** $l \in \mathcal{L}$, **guard** $g \in C$, and (partial) **update** $u \colon \mathcal{V} \to Exp$.

A (partial) **variable assignment** s over $\mathcal{V}$ is a function with domain $dom(s) \subseteq \mathcal{V}$ and $s(v) \in D_v$ for all $v \in dom(s)$. By $s_1[s_2]$ we denote the update of s_1 by s_2, i.e., $dom(s_1[s_2]) = dom(s_1) \cup dom(s_2)$, where $s_1[s_2](v) = s_2(v)$ if $v \in dom(s_2)$ and $s_1[s_2](v) = s_1(v)$ otherwise. By $e(s)$, respectively $\phi \in C$, we denote the evaluation of $e \in Exp$, respectively $\phi \in C$, over s. We write $s \models \phi$, if $\phi(s)$ evaluates to true.

The **state space** of $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ is a labeled transition system (LTS) $\Theta = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$. The **states** $\mathcal{S}$ are the complete variable assignments over $\mathcal{V}$. For the **transitions** $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ it holds $(s, l, s') \in \mathcal{T}$ iff there exists an operator $o = (g, l, u)$ such that $s \models g$ (the guard is satisfied in the source state s) and $s' = s[u(s)]$ with $u(s) = \{v \mapsto u(v)(s) \mid v \in dom(u)\}$ (the successor state s' is the update of s by u evaluated over s). We also write $s \models o$ for $s \models g$ and $s[\![o]\!]$ for $s[u(s)]$.

Observe that the separation between action labels and operators allows both, state-dependent effects (different operators with the same label l applicable in different states); as well as action outcome non-determinism (different operators with the same label l applicable in the same state).

NN Action Policies. An **action policy** π is a function $\mathcal{S} \to \mathcal{L}$. The **policy-restricted state space** Θ^π is the subgraph $\langle \mathcal{S}, \mathcal{L}, \mathcal{T}^\pi \rangle$ of Θ with $\mathcal{T}^\pi = \{(s, l, s') \in \mathcal{T} \mid \pi(s) = l\}$.

We consider action policies represented by **neural networks** (NN). Specifically, we focus on fully connected feed-forward NN with piecewise-linear activation functions. In the input layer there is an input for each state variable; and in the output layer there is an output for each action label. The policy selects an action label by applying argmax to the output layer.

Policy Safety. A **safety property** is a pair $\rho = (\phi_0, \phi_U)$, where $\phi_0, \phi_U \in C$. Here, ϕ_U identifies the set of unsafe states that should be unreachable from the set of possible start states represented by ϕ_0. That is, π is **unsafe** with respect to ρ iff there exist states $s_0, s_U \in \mathcal{S}$ such that $s_0 \models \phi_0$,

[1]Arguably, piecewise-linear activation functions can be smoothly approximated (e.g., ReLU via Swish (Ramachandran, Zoph, and Le 2018)) and vice versa (e.g., (Dutta et al. 2018)).

$s_U \models \phi_U$, and s_U is reachable from s_0 in policy-restricted Θ^π. Otherwise π is **safe**.

Policy Predicate Abstraction

In general, it is not feasible to perform explicit reachability analysis of ϕ_U (from ϕ_0) in Θ^π. Instead, our approach is to perform reachability analysis in the abstract state space obtained through **predicate abstraction** (Graf and Saïdi 1997). Given a set of predicates $\mathcal{P} \subseteq C$, an **abstract state** $s_\mathcal{P}$ is a (complete) truth value assignment over $\mathcal{P}$. The abstraction of a (concrete) state $s \in \mathcal{S}$ is the abstract state $s|_\mathcal{P}$ with $s|_\mathcal{P}(p) = p(s)$ for each $p \in \mathcal{P}$. Conversely, $[s_\mathcal{P}] = \{s' \in \mathcal{S} \mid s'|_\mathcal{P} = s_\mathcal{P}\}$ denotes the concretization of $s_\mathcal{P}$, i.e., the set of all concrete state represented by $s_\mathcal{P}$. Accordingly, we say that $s_\mathcal{P}$ satisfies a constraint $\phi \in C$, written $s_\mathcal{P} \models \phi$, iff there exists $s \in [s_\mathcal{P}]$ such that $s \models \phi$.

The abstract state space is then defined in a transition-preserving manner:

Definition 1 (Predicate Abstraction of Θ^π)**.** The **predicate abstraction** of Θ^π over $\mathcal{P}$ is the LTS $\Theta^\pi_\mathcal{P} = \langle \mathcal{S}_\mathcal{P}, \mathcal{L}, \mathcal{T}^\pi_\mathcal{P} \rangle$, where $\mathcal{S}_\mathcal{P}$ is the set of all predicates states over $\mathcal{P}$, and $\mathcal{T}^\pi_\mathcal{P} = \{(s|_\mathcal{P}, l, s'|_\mathcal{P}) \mid (s, l, s') \in \mathcal{T}^\pi\}$.

Due to the underlying policy π, we refer to $\Theta^\pi_\mathcal{P}$ as **policy predicate abstraction**. Due to its over-approximating nature, safety of π can be proven via safety in $\Theta^\pi_\mathcal{P}$:

Proposition 2 (Safety in $\Theta^\pi_\mathcal{P}$)**.** Let $\rho = (\phi_0, \phi_U)$ be a safety property. If there do **not** exist $s_\mathcal{P}, s'_\mathcal{P} \in \mathcal{S}_\mathcal{P}$ with $s_\mathcal{P} \models \phi_0$, $s'_\mathcal{P} \models \phi_U$ such that $s'_\mathcal{P}$ is reachable from $s_\mathcal{P}$ in $\Theta^\pi_\mathcal{P}$, then π is **safe** with respect to ρ.

The computation of $\Theta^\pi_\mathcal{P}$ necessitates to solve the **transition problem** for every possible abstract state transition: $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}^\pi_\mathcal{P}$ iff there exists an operator $o \in \mathcal{O}$ with label l and a concrete state $s \in [s_\mathcal{P}]$ such that $s \models o$, $s[\![o]\!] \in s'_\mathcal{P}$, and $\pi(s) = l$. We can check this via individual tests for each l-labeled operator:

Definition 3 (Transition Test of $\Theta^\pi_\mathcal{P}$)**.** Let $s_\mathcal{P}, s'_\mathcal{P} \in \mathcal{S}_\mathcal{P}$, and let $o = (g, l, u)$ be an operator. The **transition test** of $\Theta^\pi_\mathcal{P}$, denoted **TSat**$^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$, is fulfilled iff there exists $s \in [s_\mathcal{P}]$ such that $s \models o$, $s[\![o]\!] \in [s'_\mathcal{P}]$ and $\pi(s) = l$.

Transition tests are routinely addressed as satisfiability modulo theories (SMT) (Barrett et al. 1994) problems. Predicate abstraction is applicable in principle so long as any method for solving these is available. Compared to standard predicate abstraction approaches, the dominating source of complexity in computing policy predicate abstraction is that, in addition to the standard transition condition, one needs to check whether the policy π actually selects l in $s \in [s_\mathcal{P}]$.

Algorithmic Enhancements

An exact SMT solution of TSat$^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ is computationally very expensive – specifically due to the large number of disjunctions encoding every (piecewise-linear) activation function in the NN representation of π. In our current work, we address this via a range of algorithmic enhancements, leveraging relaxed tests that over-approximate TSat$^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$. If such a relaxed test is violated, then TSat$^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ is violated as well.

Necessary Conditions. One relaxation technique is through tests on necessary conditions of TSat$^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$. Necessary condition that we check are: label selection ($\exists s \in [s_\mathcal{P}]\colon \pi(s) = l$; short $l \in \pi(s_\mathcal{P})$), operator applicability ($s_\mathcal{P} \models o$), respectively their combination ($\exists s \in [s_\mathcal{P}]\colon \pi(s) = l \wedge s \models o$), and the non-policy-restricted transition condition ($\exists s \in [s_\mathcal{P}]\colon s \models o \wedge s[\![o]\!] \in [s'_\mathcal{P}]$).

These conditions essentially check different parts of TSat$^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ in isolation. Tests on these conditions have the decisive advantage that, if one such test is violated, one can skip all corresponding transition tests. For instance, if we find $l \notin \pi(s_\mathcal{P})$, then TSat$^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ is violated and can thus be skipped for all l-labeled operators o and all abstract successor states $s'_\mathcal{P}$. Additionally, tests ignoring the NN selection condition are usually much cheaper to answer.

Continuous Relaxation. Another relaxation technique is through continuous relaxation. Each test (TSat$^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ as well as tests on necessary conditions) can be relaxed by interpreting the integer state variables as continuous variables (with real-valued interval domains). The advantage of this relaxation is the applicability of existing SMT solvers specialized to NN analysis. Specifically, in our work so far, we query *Marabou* (Katz et al. 2019), an SMT solver tailored to NN with piecewise-linear activation functions.

If a relaxed test is violated, the corresponding exact test is violated too (and can be skipped). However, continuously-relaxed tests can be utilized even further: If the solution found to a relaxed test happens to be integer, the corresponding exact test is derived to be fulfilled as well. While this will in general not be the case, we can iterate relaxed tests in a **branch & bound (B&B)** search for such a solution. In each iteration, if there exists a state variable v assigned to a non-integer value α in the relaxed solution found by the solver, we pick one such v and create two search branches, restricting v to be less equal $\lfloor \alpha \rfloor$ respectively greater equal $\lceil \alpha \rceil$. A branch is terminated once the relaxed tests is found to be violated, or when an integer solution is found. If no integer solution is found during the search, the exact tests is violated. The advantage of this approach is the applicability of existing SMT solvers dedicated to NN analysis to answer not only relaxed but also exact tests.

Fixing Activation Cases. Additionally to the enhancement trough relaxed tests, there are further techniques from NN analysis that can be utilized. One such option is **fixing activation cases** in the NN. That is, given value bounds on the neurons in the network one can potentially prune some or even fix one of the (piecewise-linear) activation cases of the respective neuron. For instance, for the prominent ReLU activation function $\text{ReLU}(x) := \max(x, 0)$ the idea works as follows: If the activation-function input x is known to be less equal 0, then one can fix $\text{ReLU}(x) = 0$; if x is known to be greater equal 0, one can fix $\text{ReLU}(x) = x$.

There is a broad range of NN analysis techniques that can be utilized to compute such bounds (e.g. (Wang et al. 2018b; Li et al. 2019)) given constraints on the input of an NN (in our case $s_\mathcal{P}$). Towards transition tests – relaxed or exact – activation case fixing as well as the derived bounds themselves can be used to simplify the resulting SMT encodings and to prune the SMT search space. This idea has been deployed in other work before (e.g. (Mohammadi et al. 2020; Katz et al. 2019)). Specifically, *Marabou* (Katz et al. 2019), which we query for relaxed tests, fixes activation cases based on bounds implied by individual constraints, respectively derived through symbolic interval propagation on the network topology (Wang et al. 2018b). In our work, we then also extract these bounds towards activation case fixing in exact tests.

Experimental Results

In our experiments so far (Vinzent, Steinmetz, and Hoffmann 2022), we evaluated our approach on a collection of benchmarks that involve action outcome non-determinism. As competing approaches, we implemented an explicit-search approach enumerating all states the policy can reach, as well as a bounded-length verification approach following the ideas of Akintunde et al. (2018; 2019). The results show that our algorithmic enhancements – in particular relaxed tests answered by leveraging dedicated NN analysis techniques – are required for practicality, and that our approach can outperform its competitors.

Future Work

We distinguish three research lines that we plan to cover in our future work: technical enhancements, adaption to other settings, and automatic abstraction refinement. We also note that there are many more approaches from formal methods that (when adapted to the the NN policy setting) can in principle serve as competing verification approaches (e.g. (Tonetta 2009; Cimatti et al. 2016)).

Technical Enhancements. There remains a broad range of NN analysis techniques that can be utilized to improve performance. For instance, techniques to derive tight neuron value bounds (e.g. (Li et al. 2019)) can be leveraged to enhance activation case fixing. In principle, our approach can profit from any progress in the analysis of single NN decision episodes.

Furthermore, we also plan to leverage **adversarial attack** methods for under-approximation purposes (e.g. (Goodfellow, Shlens, and Szegedy 2015)). In essence, if an attack finds a solution to a transition test, the call to SMT can be skipped. If not, there still may exist a solution and we call SMT. That said, robustness guarantees on adversarial attacks (e.g. (Hein and Andriushchenko 2017)) might even be used to prove that a solution does not exist.

On the technical level there is also a vast potential for **parallelization** – running several transition tests at once (different (solver) techniques, different transitions); as well as **incremental solving** – e.g., preserving the solver-internal state between selection and transition tests.

Adaption to Other Settings. In our current setting, we consider non-probabilistic state spaces with non-deterministic action outcomes. A natural extension to make our approach amenable to **probabilistic** systems is by performing value iteration (e.g. (Givan, Leach, and Dean 1997)) in the abstract state space; obtaining quantitative safety results. The abstraction computation itself remains unchanged.

Currently, we allow π to select inapplicable actions, i.e., there may exist $s \in \mathcal{S}$ such that $\pi(s)$ does not label any outgoing transition and the policy execution stalls. Here, as potential future work, we plan to extend our approach to enable stalling detection. Alternatively, a prominent option is also to super-impose applicability on π, restricting its selection to the applicable actions. Again, we plan to adapt our approach to such settings as part of future work. In principle, both adaptions are straight-forward; resulting in significantly more expensive SMT problems tough.

Abstraction Refinement. In our work so far, we assume the predicate set $\mathcal{P}$ to be provided as input. Yet, the converse of Proposition 2 does not hold, i.e., unsafety in $\Theta^{\pi}_{\mathcal{P}}$ does not imply unsafety in Θ^{π}. Depending on $\mathcal{P}$, the abstraction may contain *spurious* paths without correspondence in the concrete state space. Hence, automatic abstraction generation (respectively refinement) remains key future work towards a complete verification procedure.

Here, **counter example guided abstraction refinement** (CEGAR) (e.g. (Clarke et al. 2000)) is a common procedure to refine $\mathcal{P}$, iteratively removing spurious (unsafe) paths until either the abstraction is proven safe, or a *non-spurious* unsafe path is found – proving unsafety of the concrete system. The adaption of CEGAR to NN policy verification is non-trivial since refinement predicates must reflect NN selection behavior; a path $s^1_{\mathcal{P}}, l^1, \ldots, s^i_{\mathcal{P}}, l^i, \ldots, s^n_{\mathcal{P}}$ may be spurious due to $l^i \in \pi(s^i_{\mathcal{P}})$ while $\pi(s^i) \neq l^i$ in the path concretizations. Our future research will address this challenge. One idea is to approximate (state-dependent) "selection guards" via constraints that split $s^i_{\mathcal{P}}$ based on NN behavior, e.g., with respect to s_i in a path concretization. The selection guards can then be fed into standard refinement procedures based on weakest precondition computation.

In the context of CEGAR, we also plan to investigate the potential of incremental abstraction computation, i.e., reusing (transition) information of coarser abstractions when computing the refined abstract state space. Most importantly such information can be used to prune the potential transition space as well as to witness existing transitions. Another technique of interest is lazy abstraction, i.e., refining the abstraction only locally respectively checking (abstract) policy-restriction only once an abstract unsafe path has been found. Moreover, the search for counter examples may also be sped up using heuristics, e.g., again based on information from coarser (non-policy-restricted) abstractions.

Conclusion

In our work, we provide policy predicate abstraction as a new method to address NN policy verification. Our experiments so far have shown that it can outperform competing approaches and that our algorithmic enhancements are required for practicality. An important future task is to address the automatic selection of the abstraction predicates.

Overall, we believe that NN policy verification is important, and we hope that our work provides one basic building block for this huge endeavor.

References

Akintunde, M.; Lomuscio, A.; Maganti, L.; and Pirovano, E. 2018. Reachability Analysis for Neural Agent-Environment Systems. In *KR*.

Akintunde, M. E.; Kevorchian, A.; Lomuscio, A.; and Pirovano, E. 2019. Verification of RNN-Based Neural Agent-Environment Systems. In *AAAI*.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2017. Safe Reinforcement Learning via Shielding. *CoRR*, abs/1708.08611.

Ball, T.; Majumdar, R.; Millstein, T. D.; and Rajamani, S. K. 2001. Automatic Predicate Abstraction of C Programs. In *Prog. Lang. Design and Implementation (PLDI)*.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 1994. Satisfiability modulo theories. *In Handbook of Satisfiability*, 825–885.

Christakis, M.; Eniser, H. F.; Hermanns, H.; Hoffmann, J.; Kothari, Y.; Li, J.; Navas, J.; and Wüstholz, V. 2021. Automated Safety Verification of Programs Invoking Neural Networks. In *CAV*.

Cimatti, A.; Griggio, A.; Mover, S.; and Tonetta, S. 2016. Infinite-state invariant checking with IC3 and predicate abstraction. *Formal Methods Syst. Des.*, 49(3): 190–218.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-Guided Abstraction Refinement. In *CAV*.

de Moura, L.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In *TACAS*.

Dutta, S.; Jha, S.; Sankaranarayanan, S.; and Tiwari, A. 2018. Output Range Analysis for Deep Feedforward Neural Networks. In Dutle, A.; Muñoz, C. A.; and Narkawicz, A., eds., *NFM*.

Ehlers, R. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In D'Souza, D.; and Kumar, K. N., eds., *Automated Technology for Verification and Analysis*.

García, J.; and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *JMLR*, 16: 1437–1480.

Givan, R.; Leach, S. M.; and Dean, T. L. 1997. Bounded Parameter Markov Decision Processes. In Steel, S.; and Alami, R., eds., *ECP*.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *Learning Representations (ICLR)*.

Graf, S.; and Saïdi, H. 1997. Construction of Abstract State Graphs with PVS. In *CAV*.

Gros, T. P.; Groß, D.; Gumhold, S.; Hoffmann, J.; Klauck, M.; and Steinmetz, M. 2020a. TraceVis: Towards Visualization for Deep Statistical Model Checking. In *Proceedings of the 9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'20)*.

Gros, T. P.; Hermanns, H.; Hoffmann, J.; Klauck, M.; and Steinmetz, M. 2020b. Deep Statistical Model Checking. In *Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*.

Hein, M.; and Andriushchenko, M. 2017. Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation. In *NIPS*.

Huang, S.; Fan, J.; Li, W.; Chen, X.; and Zhu, Q. 2019. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Trans. Emb. Comp. Sys.*, 18: 1–22.

Ivanov, R.; Carpenter, T. J.; Weimer, J.; Alur, R.; Pappas, G. J.; and Lee, I. 2021. Verifying the Safety of Autonomous Systems with Neural Network Controllers. *ACM Trans. Emb. Comp. Sys.*, 20: 7:1–7:26.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV*.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D. L.; Kochenderfer, M.; and Barrett, C. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV*.

Li, J.; Liu, J.; Yang, P.; Chen, L.; Huang, X.; and Zhang, L. 2019. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In *Static Analysis (SAS)*.

Mohammadi, K.; Karimi, A.; Barthe, G.; and Valera, I. 2020. Scaling Guarantees for Nearest Counterfactual Explanations. *CoRR*, abs/2010.04965.

Ramachandran, P.; Zoph, B.; and Le, Q. V. 2018. Searching for Activation Functions. In *ICLR Workshop Track Proceedings*.

Steinmetz, M.; Fiser, D.; Eniser, H.; Ferber, P.; Gros, T.; Heim, P.; Höller, D.; Schuler, X.; Wüstholz, V.; Christakis, M.; and Hoffmann, J. 2022. Debugging a Policy: Automatic Action-Policy Testing in AI Planning. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Tonetta, S. 2009. Abstract Model Checking without Computing the Abstraction. In Cavalcanti, A.; and Dams, D., eds., *Formal Methods*.

Tran, H.; Cai, F.; Lopez, D. M.; Musau, P.; Johnson, T. T.; and Koutsoukos, X. D. 2019. Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control. *ACM Trans. Embed. Comput. Syst.*, 18(5s): 105:1–105:22.

Vinzent, M.; Steinmetz, M.; and Hoffmann, J. 2022. Neural Network Action Policy Verification via Predicate Abstraction. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018a. Efficient Formal Safety Analysis of Neural Networks. In Bengio, S.; Wallach, H. M.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *NeurIPS*.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018b. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *USENIX Security Symposium*.

Plan Recognition – Dissertation Abstract

Kristýna Pantůčková

Supervisor: Roman Barták
Charles University, Faculty of Mathematics and Physics
Ke Karlovu 3, 121 16 Praha 2, Czech Republic
pantuckova@ktiml.mff.cuni.cz

Abstract

The topic of the dissertation is plan recognition. Plan recognition is the task of recognizing the goal of an agent based on the observed actions. The aim of the current research is to develop an efficient approach to plan recognition in hierarchical task networks (HTN). We intend to improve the performance of existing parsing-based approach by heuristics based on landmarks.

Introduction

Plan recognition is relevant to many fields of artificial intelligence. For instance, plan recognition is related to behaviour recognition, which can be used to recognize suspicious behaviour in public space (Niu et al. 2004). In the field of computer security, plan recognition can be used to predict cybernetic attacks (Li et al. 2020). Other applications include multi-agent systems (Kaminka, Pynadath, and Tambe 2002), or artificial intelligence in computer games (Ha et al. 2011).

We focus on plan recognition in hierarchical task networks (HTN), which allow to express a natural hierarchy of tasks. A domain model of HTN planning consists of a set of abstract tasks, actions and methods. Abstract tasks can be decomposed into subtasks via methods. The aim of hierarchical planning is to decompose the given goal task into a sequence of actions (indecomposable tasks). In hierarchical plan recognition, we intend to find the goal task whose decomposition covers all of the observed actions. In contrast to plan verification, we do not expect that the sequence of observed actions given on input is a complete plan; the goal task will be decomposed into a sequence of actions which contains the set of observed actions as a subset.

Currently there appear to be only two approaches to recognition of hierarchical plans. The first of these approaches is based on compilation to HTN planning (Höller et al. 2018). The second approach (Barták, Maillard, and Cardoso 2020) was inspired by parsing of grammars. As the approach of (Barták, Maillard, and Cardoso 2020) performs worse than the approach of (Höller et al. 2018) on instances with a high number of missing (unobserved) actions, the aim of our current research is to improve the performance of the approach of (Barták, Maillard, and Cardoso 2020) by using landmarks. Our algorithm is based on composing tasks from subtasks until a goal task is found, and we intend to use method landmarks to guide the search. A fact landmark of a decomposition method m is a fact that must be true at some point in all plans created by decomposing the root task of m via m; a task landmark of m is an abstract task or an action which must be contained in all such plans.

Background on HTN plan recognition

Hierarchical planning focuses on planning problems where goals (tasks) can be hierarchically decomposed into subgoals (subtasks). Indecomposable (primitive) tasks are called actions. A planning problem can be described by a hierarchical task network (HTN).

An HTN is described by a pair $w = (T, C)$, where T is a set of tasks and C is a set of constraints over tasks. There are four types of constraints: $t_1 \prec t_2$ is a precedence constraint over tasks t_1 and t_2, $before(T', p)$ indicates that the proposition p must be true in the state before executing tasks in the set of tasks T', $after(T', p)$ indicates that p must be true in the state after executing tasks in T' and $between(T', T'', p)$ indicates that p must be true in all states between the sets of tasks T' and T''. Task decomposition is described by methods, where a method $m = (t, w)$ decomposes a task t to a hierarchical task network w.

A planning problem can be defined as $P = (F, C, A, M, s_0, w_0)$, where F is a set of fluents describing states, C is a set of compound (decomposable) tasks, A is a set of actions (primitive tasks), M is a set of decomposition methods, s_0 is an initial state and w_0 is the initial task network which represents the goal. Actions in A are defined by preconditions and positive and negative effects. Precondition of an action a is a proposition that must be true in order to execute a, positive effect is a proposition that will be true after executing a and negative effect is a proposition that will be false after executing a. The task of a planner is to decompose the tasks in the initial network to primitive tasks. If $w = (T, C)$ is a task network obtained from w_0 using methods from M, all abstract tasks in w are decomposed, and $\pi =< a_1, ..., a_k >$ are all actions in w, where the ordering of actions in π corresponds to the ordering of nodes in w and a_1 is executable in the state s_0, then π is a solution to the HTN planning problem P.

An HTN plan recognition problem is defined as $R = (F, C, A, M, s_0, O, G)$, where $O =< o_1, ..., o_k >$ is an observed plan prefix. The aim of plan recognition is to decide whether there is a goal $g \in G$ and a sequence of actions $< o_{k+1}, ..., o_n >$ such that $< o_1, ..., o_n >$ is a valid plan for the goal g applicable in s_0.

Related work

(Höller et al. 2018) developed an HTN plan recognition algorithm inspired by the "plan recognition as planning" approach of Ramírez and Geffner (Ramírez and Geffner 2009), who leveraged compilation to planning to recognize classical sequential plans. This compilation-based hierarchical plan recognition algorithm requires only one run of a hierarchical planner in a modified hierarchical task network. For an instance of an HTN plan recognition problem, (Höller et al. 2018) define a new goal, which can be decomposed into the initial network of one of the candidate goals, and introduce new constraints to ensure that the resulting plan will contain all observed actions.

(Barták, Maillard, and Cardoso 2020) proposed a different approach, which was inspired by parsing of grammars. Their algorithm firstly tries to find a goal task whose decomposition tree can cover all observations. If the task is not found, the algorithm guesses missing observations by adding all possible actions after the observed action sequence. Plan length is iteratively extended until a goal task is found. In this paper, the authors extended their older algorithm for HTN plan verification (Barták, Maillard, and Cardoso 2018), which described decomposition rules of an HTN planning domain by rewriting rules of attribute grammars.

(Barták, Maillard, and Cardoso 2020) also presented an empirical comparison of the two HTN plan recognition approaches. In contrast to (Höller et al. 2018), their algorithm does not require the initial state to be specified as part of the input as it can be computed during plan recognition. According to the empirical comparison presented in (Barták, Maillard, and Cardoso 2020), the parsing-based algorithm (Barták, Maillard, and Cardoso 2020) was faster than the compilation-based algorithm (Höller et al. 2018) on problem instances with only few missing observations. However, the authors of (Barták, Maillard, and Cardoso 2020) go on to observe that as the number of missing observations grows, the solving time of the parsing-based algorithm grows exponentially, while the compilation-based algorithm (Höller et al. 2018) performs significantly better.

Other hierarchical plan recognition approaches work with models weaker than HTN. For instance, there are approaches based on manipulations with tree-based structures, parsing or rewriting of strings (e.g. (Avrahami-Zilberbrand and Kaminka 2005), (Mirsky, Gal, and Shieber 2017), or (Geib, Maraist, and Goldman 2008)).

Currently, we aim to develop a landmark-based HTN plan recognition approach. Landmarks have already been used for classical plan recognition. The algorithm of (Pereira, Oren, and Meneguzzi 2017) finds the most likely goal by observing the landmarks that were achieved in the plan and comparing them with known landmarks of candidate goals. In comparison to an older approach based on compilation to planning (Ramírez and Geffner 2009), this landmark-based approach is significantly faster with a similar accuracy. (Pereira, Oren, and Meneguzzi 2017) proposed two heuristics for comparing candidate goals based on achieved landmarks. The basic heuristic computes the proportion of all landmarks and achieved landmarks, while the second heuristic, which leads to a better efficiency, takes into account "uniqueness" of landmarks among all goals.

(Vered et al. 2018) used landmarks combined with compilation to planning to develop an algorithm for on-line classical plan recognition, where time efficiency is crucial. Nevertheless, the authors utilize landmarks differently than (Pereira, Oren, and Meneguzzi 2017). After each new observation arriving in an on-line setting, they recompute optimal plans consistent with the observations. Compilation to planning is used to compute the probability distribution of candidate goals; landmarks are used to rule out improbable goals, for which this expensive computation is not necessary.

Current research

Previously we focused on survey of work related to the topic of the thesis – classical and hierarchical plan recognition. Currently we aim to develop an efficient algorithm for HTN plan recognition. Our approach is based on the algorithm of (Barták, Maillard, and Cardoso 2020). Instead of systematically generating all possible plans and composing abstract tasks from the available subtasks, we try to guess suitable abstract tasks which can be decomposed into some of the available tasks. We generate partial plans, which consist of actions, abstract tasks whose decomposition covers some of the observed actions, and extra tasks ordered after the observation sequence, which were generated by the abstract tasks. However, the implementation has not been finished yet. The idea of the algorithm is shown in Figure 1.

The procedure is described in Algorithm 1. The set S contains all generated partial plans. A partial plan contains the sequence of the observed actions and some abstract tasks that decompose into tasks in the partial plan. Additionally, abstract tasks may add some extra tasks which are not mapped to tasks in the plan. These tasks are ordered after the observed actions. The root task of a partial plan is a goal task if all observed actions are covered (all observed actions are contained in decomposition trees of some abstract tasks from the plan) and the new tasks after the plan can be ordered and decomposed to create a valid plan. For the validation of the latter condition, we need to call an HTN planner to decompose the new abstract tasks.

For a partial plan P, app_P denotes the set of methods that are applicable to P. A method m is applicable to a plan P if one of the potential first subtasks of m (one of the tasks that can be ordered as the first subtask in decomposition) is contained in the set of available (uncovered) tasks in P. Application of m to P is mapping of some of the subtasks of m to some of the available tasks in P. For each possible application, there will be one new partial plan. Each new partial plan will add the root task of m into the set of its available tasks. Some of the tasks available in P will become unavailable and m may add some new tasks which will not be mapped to tasks in P.

We intend to select the pair (P, m) based on a heuristic value. The heuristic function will depend on the proportion of landmarks of m achieved in P and the total number of

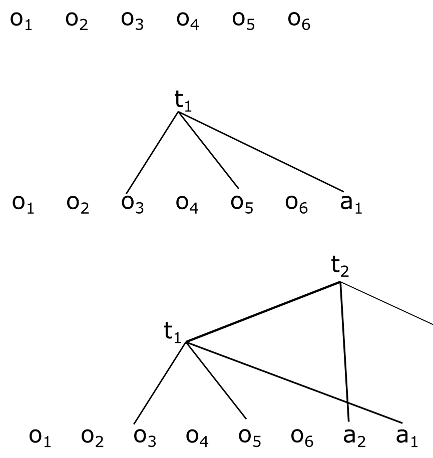


Figure 1: This figure describes the idea of our algorithm. In the first picture, there is an initial partial plan containing only the observed actions $o_1, ...o_6$. The second partial plan is the result of application of a decomposition method with the root task t_1. This method covers observations o_3 and o_5 and adds a new action a_1 after the plan. Application of the next method with the root task t_2 creates a partial plan with a new action a_2 and a new abstract task t_3. At this point, the order of a_1, a_2 and t_3 is not decided and t_3 is not decomposed; we try to resolve these problems only after a potential goal task covering all observations is found.

landmarks of m. For extracting landmarks of methods, we use the algorithm proposed by (Höller and Bercher 2021). In our settings, we expect a set of possible goal tasks as part of the input. For these tasks, we create AND/OR graphs and find landmarks of tasks, methods, actions and facts. Nevertheless, we will need only method landmarks in our heuristic function.

After creating a new partial plan, we use the procedure described in (Barták, Maillard, and Cardoso 2020) to check if all conditions of m are satisfied in the new plan. Our algorithm is clearly sound as if it returns a goal task t, t can be decomposed such that all observed actions are covered and the resulting plan is valid. However, the algorithm is not complete. We may move towards completeness for example by interleaving heuristic selections and random selections of plan-method pairs. Moreover, the solution will not be optimal (with respect to plan length). Quality of solutions may be improved by introducing a more complex heuristic function, which could depend for instance on length of a plan, number of uncovered actions, and number of new abstract tasks.

Future directions

Currently, we are working on implementation of our approach. We will compare our approach with the existing algorithms for HTN plan recognition ((Barták, Maillard, and Cardoso 2020) and (Höller et al. 2018)). Based on the results, we will focus on the heuristic function to improve the performance of the algorithm. In the future, we plan to deal with missing or incorrect observation, as our current approach requires a complete and correct plan prefix as an input.

Algorithm 1: Landmark-based HTN plan recognition

Input: a sequence of observed actions
Output: a corresponding goal task
Variables: S – a set of partial plans, app_P for each partial plan P – a set of all methods applicable to P

```
1: $P_0$ = initial partial plan containing observed actions
2: $S = \{P_0\}$
3: while true do
4:     $P = argmax_{P \in S} max\{h(m, P) | m \in app_P\}$
5:     $m = argmax_m\{h(m, P) | m \in app_P\}$
6:     for all possible applications of m to P do
7:         $P_1$ = apply $m$ to $P$
8:         if $P_1$ is consistent with all conditions then
9:             if $P_1$ covers all observations and a valid plan can
               be generated from $P_1$ then
10:                return the root task of $P_1$
11:            else
12:                add $P_1$ to $S$
13:            end if
14:        end if
15:    end for
16: end while
```

Acknowledgements

Research is supported by the Charles University, project GA UK number 156121.

References

Avrahami-Zilberbrand, D.; and Kaminka, G. A. 2005. Fast and Complete Symbolic Plan Recognition. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 653–658.

Barták, R.; Maillard, A.; and Cardoso, R. 2018. Validation of hierarchical plans via parsing of attribute grammars. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2020. Parsing-based Approaches for Verification and Recognition of Hierarchical Plans. In *Plan, activity and intent recognition workshop at the thirty-fourth AAAI Conference on Artificial Intelligence*.

Geib, C. W.; Maraist, J.; and Goldman, R. P. 2008. A New Probabilistic Plan Recognition Algorithm Based on String Rewriting. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, 91–98.

Ha, E.; Rowe, J.; Mott, B.; and Lester, J. 2011. Goal recognition with Markov logic networks for player-adaptive games. In *Proceedings of the seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 6.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2018. Plan and goal recognition as HTN planning. In *2018 IEEE*

Thirtieth International Conference on Tools with Artificial Intelligence, 466–473.

Höller, D.; and Bercher, P. 2021. Landmark Generation in HTN Planning. In *Proceedings of the thirty-fifth AAAI Conference on Artificial Intelligence (AAAI)*, 11826–11834.

Kaminka, G. A.; Pynadath, D. V.; and Tambe, M. 2002. Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of Artificial Intelligence Research*, 17: 83–135.

Li, T.; Liu, Y.; Liu, Y.; Xiao, Y.; and Nguyen, N. A. 2020. Attack plan recognition using hidden Markov and probabilistic inference. *Computers & Security*, 97: 101974.

Mirsky, R.; Gal, Y.; and Shieber, S. M. 2017. CRADLE: an online plan recognition algorithm for exploratory domains. *ACM Transactions on Intelligent Systems and Technology*, 8(3): 1–22.

Niu, W.; Long, J.; Han, D.; and Wang, Y.-F. 2004. Human activity detection and recognition for video surveillance. In *Proceedings of the 2004 IEEE International Conference on Multimedia and Expo (ICME)*, volume 1, 719–722.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 3622–3628.

Ramírez, M.; and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the twenty-first International Joint Conference on Artifical Intelligence*, 1778–1783.

Vered, M.; Pereira, R. F.; Kaminka, G.; and Meneguzzi, F. R. 2018. Towards online goal recognition combining goal mirroring and landmarks. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, 2112–2114.

Probabilistic Replanning with Guarantees – Dissertation Abstract

Johannes Schmalz
School of Computing, Australian National University
johannes.schmalz@anu.edu.au
Supervisor: Felipe Trevizan

Abstract

State-of-the-art probabilistic replanners are solvers for probabilistic planning problems that offer a very efficient means to generate a suboptimal solution quickly, and in an anytime fashion improve on it. Unfortunately, current approaches do so at the cost of guarantees, i.e. the solution may not be optimal, and it can not guarantee its solution will lead to the goal with certainty. To address this issue we introduce CoGNeRe, a novel probabilistic replanner that uses techniques from operations research to provide guarantees and flexibility that previous replanners can not offer.

Overview

Probabilistic replanners are a category of planners for Stochastic Shortest Path Problems (SSPs) (Geffner and Bonet 2013, 79, 81), which iteratively work on improving a candidate policy, and are able to return this candidate as a potentially partial policy when prompted for a solution in an anytime manner. Most replanners use the following method: relax the probabilistic effects of the problem (determinisation), solve the ensuing subproblem with strong deterministic solvers, append this plan as part of the candidate policy, and iterate these steps on undefined states to fill in the policy. FF-Replan (Yoon, Fern, and Givan 2007) and its extension Robust-FF (Teichteil-Königsbuch, Infantes, and Kuter 2008) have demonstrated the effectiveness of this approach winning the International Probabilistic Planning Competitions in 2004 and 2008 respectively. In terms of generating solutions quickly they are still considered state-of-the-art. However, these algorithms are not able to provide guarantees of solution quality, in particular, they can not guarantee that their policy has the highest possible probability of reaching a goal, and if they do manage to find such a policy, they can not guarantee the minimal expected cost.

In response, we present our column generation network flow replanner (CoGNeRe), a novel algorithm that combines this replanning approach with the column generation technique from operations research (Desrosiers and Lübbecke 2005). The aim is to exploit the speed of replanning and the mathematical guarantees of column generation. CoGNeRe can indeed provide guarantees of optimality, now we are filling in remaining theoretical gaps and refining the implementation to obtain an efficient planning algorithm. Preliminary results suggest that usually CoGNeRe is slower to output its first useful policy than Robust-FF, but is quickly able to overtake Robust-FF in terms of solution quality. We expect that CoGNeRe will generally converge to the optimal policy slower than state-of-the-art optimal planners like LRTDP (Bonet and Geffner 2003); CoGNeRe sacrifices fast convergence in favour of strong anytime performance.

$$
\begin{aligned}
\min_{\textbf{flow}} \quad & \sum_{X \in \mathcal{Q} \cup \mathcal{W}} \text{flow}_X \cdot \text{cost}(X) && \text{(LP 1)} \\
\text{s.t.} \quad & \text{flow}_X \geq 0 \qquad \forall X \in \mathcal{Q} \cup \mathcal{W} && \text{(C1)} \\
& \sum_{Q \in \mathcal{Q}} \text{flow}_Q = 1 && \text{(C2)} \\
& \text{regrouping constraints} && \text{(C3)}
\end{aligned}
$$

To describe CoGNeRe, first consider the linear program (LP) that computes the optimal policy for an SSP by minimising a network flow problem (d'Epenoux 1963). A theorem from network flow (Ahuja, Magnanti, and Orlin 1993, 80–81) lets us decompose flow across a deterministic-planning-problem graph into a combination of paths and cycles; we generalised this to probabilistic-problem graphs. With this result we can find an SSP's cheapest flow and therefore its optimal policy as a combination of plans and cycles across the SSP's all-outcomes determinisation, that is, a class of planning problem that relaxes the SSP by mapping each probabilistic effect to a single deterministic action (Yoon, Fern, and Givan 2007). So, we can compute an SSP's optimal policy with LP 1. In LP 1 we consider the set of plans and cycles over the all-outcomes determinisation, $\mathcal{Q}$ and $\mathcal{W}$ respectively; and we introduce variables flow_X for each $X \in \mathcal{Q} \cup \mathcal{W}$ to denote the amount of flow being pumped through the plan or cycle X. Constraint C1 forces flow to be non-negative; the convexity constraint C2 forces a flow of 1 to pass from the initial state to goals, which can be interpreted as the requirement that the corresponding policy reaches goal with probability 1; and the regrouping constraints C3 force the flow to respect the probability distribution of action effects in the SSP.

In LP 1 there is a variable for each plan and cycle in the all-outcomes determinisation, which is intractable to enumerate explicitly for any interesting problem, so solving this LP directly is impractical. This is where column generation can help us: given the specification of an LP with intractably many variables, referred to as the master problem, column generation works with a smaller LP, the reduced master problem (RMP), which contains a subset of the master problem's variables. The algorithm solves the reduced master problem, and iteratively adds variables as needed to converge to a solution that is optimal for the master problem. The intuition for this approach is that not all variables — and in fact most variables — are not relevant to the optimal solution, so we focus on reduced master problems which are sufficiently small to solve, but still lead us to the optimal solution of the master problem. Column generation is able to determine which variables need to be added to the RMP to converge to the master problem's optimal solution by constructing a series of pricing problems, which are relaxations of the original problem. A pricing problem's solution yields the new variables we need to add, and an absence of a solution informs us that column generation has converged.

CoGNeRe applies column generation to the LP for finding the optimal policy as a combination of plans and cycles on the all-outcomes determinisation as follows:

1. we start with a small set of plans and cycles,
2. solve the reduced master problem,
3. construct and solve the corresponding pricing problem to find a plan or cycle whose addition to the RMP will improve the RMP's objective in the next iteration,
4. if we find such a plan or cycle: add it, and repeat from step 2; if not, we have an optimal solution and can terminate.

In CoGNeRe, the pricing problem turns out to be a deterministic shortest path problem; so, as with other replanners, we repeatedly solve deterministic planning problems to obtain a policy for the original problem.

Unfortunately, the devil is in the details: the pricing problem is given by the all-outcomes determinisation with potentially negative action costs determined by the column generation framework with respect to each state. A solution to the pricing problem is a negative cost plan, or negative cycle, or confirmation that neither exist. So, we are dealing with a deterministic shortest path problem with state-dependent costs, negative costs, and crucially, potentially with negative cycles.

Note that the dynamics of pricing problems remain identical across iterations except for action costs, which get updated at every step, i.e. the states, action, effects, etc. remain unchanged, only the costs of some actions are updated in a state-dependent manner.

Current Work

Dealing With Conditional Negative Costs And Negative Cycles

Recall that to solve the pricing problem we must return a negative plan or cycle if it exists with the updated costs, otherwise we terminate column generation. We were able to get relatively strong performance on small problems using a variant of Bellman-Ford with an early-stop mechanic, i.e. once a negative plan or cycle has been detected it returns it straight away. However, this approach and others described by a survey of similar problems (Cherkassky and Goldberg 1999) are uninformed and polynomial with respect to the state space, so they do not scale to larger problems. So, the challenge is to find an informed algorithm that can cope with negative, state-dependent costs, and can detect negative cycles. In fact A* with minor modifications can solve such problems, as long as the heuristic is admissible; so the issue now is that getting an informative and admissible heuristic with negative state-dependent costs is difficult, and even undefined in the presence of negative cycles.

Negative cycles are particularly difficult to deal with, since they can appear anywhere in the state space, without any indication of their presence from surrounding states or transitions — as long as the negative cycle is reachable, it is an optimal solution. For now we avoid the issue by focusing on acyclic problems. This is a limitation, but still leaves us with a large class of problems, notably any SSPs with the notion of monotonically increasing or decreasing timesteps or resources, e.g. finite-horizon SSPs and vehicle routing with fuel consumption.

Even in the absence of negative cycles, negative costs make it difficult to use state-of-the-art heuristics. As a case study, consider disjoint action landmarks (in the deterministic setting), i.e. a set of actions $\mathcal{L}$ such that any plan must use at least one action from $\mathcal{L}$. With non-negative cost actions we can give an admissible heuristic by $\min_{l \in \mathcal{L}} \mathrm{cost}(l)$ since the cheapest possible way to pass through the landmark is by applying the cheapest action once. This argument falls apart with negative costs, since the cheapest way to pass through a landmark may collect multiple negative cost actions in the landmark. This issue makes it non-trivial to adapt landmark-based heuristics to pricing problems e.g. LM-cut (Helmert and Domshlak 2009).

Generating heuristics for state-dependent problems is an actively studied research question. One approach uses edge-valued multi-valued decision diagrams to compactly encode cost functions (Geißer, Keller, and Mattmüller 2015, 2016). Unfortunately this doesn't work for us, since we do not have neat algebraic expression for expressing costs, but rather different costs on a per-state basis, as determined by the pricing problem.

We are still exploring which heuristics are most suitable to be adapted to our use-case.

Exploiting Similarity Between Pricing Problems

Each pricing problem is identical in terms of dynamics, i.e. each pricing problem is similar to the original problem's all-outcomes determinisation, except action costs, which are determined in a state-dependent manner by the column generation algorithm. More formally, the transition systems induced by each pricing problem is identical, up to cost and labels of particular transitions. Our current approach for exploiting this is a variant of the Lifelong Planning A* algorithm (Koenig, Likhachev, and Furcy 2004). The idea is that we run A*, but upon returning a solution we do not discard the frontier and current best costs, but store it, and for the next pricing problem we analyse what cost changes have been made:

- if an edge cost has been decreased then reinsert the affected vertex into the frontier with the new cost, to ensure that the change is propagated to the optimal path if relevant;
- if an edge cost has been increased, then all paths that relied on that edge must be re-evaluated, i.e. descendants of that edge have their current best cost reset to ∞, and they are re-inserted into the frontier.

In the worst case an edge close to the initial state has its cost increased, and we have to recompute the entire search graph, and thus we are running regular A* with the overhead of processing edge updates and the frontier. In practice however, we have found that this approach reduced computation time substantially.

Lower Bound For Policy Cost

The relationship between primal and dual LPs allows us to provide upper and lower bounds for the problem. A novel feature of our approach is that we can leverage the column generation framework to provide a lower bound for the optimal policy cost which becomes tighter as CoGNeRe progresses.

Consider the objective of the optimal solution for the master problem z^*_{MP}. A well-known theorem from column generation (Desrosiers and Lübbecke 2005, 8–9) gives us the bound $\bar{z} + \kappa\bar{c}^* \leq z^*_{\text{MP}} \leq \bar{z}$ where

- $\bar{z}$ is the objective value of the optimal solution for our current RMP;
- $\bar{c}^*$ is the smallest reduced cost for our current RMP, i.e. the cost of the most negative plan or cycle in our pricing problem;
- κ is an upper bound for the sum of all variable assignments in the master problem's optimal solution. In an acyclic problem, thanks to the constraint that ensures a flow of one through the network (convexity constraint C2), we can set $\kappa = 1$.

The value of $\bar{z}$ corresponds to the cost of the current policy, which clearly gives an upper bound to the cost of the optimal policy, hence $z^*_{\text{MP}} \leq \bar{z}$. In the lower bound for z^*_{MP}, $\bar{c}^*$ denotes the most we can possibly decrease the current solution's objective by adding the variable corresponding to the most negative plan or cycle. In the acyclic case we can assign the new variable a value of at most 1 (due to convexity constraint C2), and so the most we can reduce the objective is indeed $\bar{c}^*$. In the presence of cycles it gets a bit more complicated since a cycle's variable is not bounded by the convexity constraint. Note that $\bar{z}$ is monotonically decreasing since solutions can only improve with the introduction of new columns, so the upper bound is only getting tighter; the lower bound has no such guarantee and may fluctuate, but we can take the maximum across all RMPs and thereby get a monotonically increasing lower bound as well.

Lower bounds are not by themselves novel, since heuristics can provide lower bounds; but especially in a probabilistic setting these tend to be very loose, and the lower bounds from column generation can be tighter. This technique enables our anytime solver to give a more precise optimality gap.

Future Work

Here we outline some potential directions of future research that CoGNeRe and more generally, operations research in planning can take.

Dealing with Cyclic Problems

CoGNeRe is able to solve cyclic problems by running an algorithm that can detect negative cycles e.g. Bellman-Ford on the pricing problems, and then adding any found negative cycles as columns, just like with plans. As discussed before, the issue is that we need admissible heuristics to scale up to larger problems. First, we need to redefine what it means for a heuristic to be admissible in the presence of negative cycles, and there are two options: (1) an admissible heuristic must still underestimate the actual cost, so if a negative cycle is reachable from state s then $h(s) = -\infty$; (2) we only require the heuristic to reason about plans, so an admissible heuristic must underestimate the cost of the cheapest plan, and may ignore negative cycles.

Approach (1) has the potential of destroying a lot of information about negative plans, and in a sense prioritises negative cycles. Approach (2) suggests that cycle and plan search should be separated, e.g. we run a plan finding algorithm with some cycle detection mechanisms, as in (Cherkassky and Goldberg 1999), which can prioritise plans over negative cycles. In both cases the issue is that CoGNeRe is sensitive to the order in which columns are added so prioritising cycles or plans tends to perform well on some problems, but poorly on others. A strong solution needs to balance these issues, either by intelligently deciding whether a cycle or plan are more useful, or by adding both.

Solving Pricing Problems as Diverse Plans Problems

In column generation, for some problems, it is very efficient to extract multiple solutions from a single pricing problem, and add all of them as columns to the reduced master problem at once. The idea is that a column that is guaranteed to improve the RMPs solution at one step, is likely to be useful later as well; it is also a way to deal with the property of column generation that the column with most negative reduced cost may not correspond to the column that lets us converge most quickly. So by adding multiple columns we increase the chances of adding columns that lead to a solution quicker, and potentially allows us to reuse the results of the pricing problem in future iterations. Often this approach relies on the columns being sufficiently diverse. As motivation, in CoGNeRe, if we add multiple plans that share an action with an undesirable probabilistic effect, then that action's determinisation will receive a high cost in future iterations, which indicates to us that the columns are not useful.

These requirements can be expressed as a bounded quality diverse planning problem (Katz and Sohrabi 2020), where a solution is a set of plans, where the plans are sufficiently diverse, and each plan's cost is bounded by some constant value. For CoGNeRe, the bound is 0 so that we only consider negative-cost plans in the pricing problem; diversity can be defined in terms of how many actions are shared. Katz and Sohrabi (2020) propose a flexible method for this style of problem which works by forbidding certain plans at the level of the planning task, which is not amenable to our method for exploiting similarity between pricing problems. So the challenge becomes how to combine these concepts.

Generalising CoGNeRe To More Problems

The LP approach endows CoGNeRe with a lot of flexibility with respect to objective functions and additional constraints. For instance, we can search for a policy that maximises the probability of reaching a goal; or, we can even stop CoGNeRe once it has reached some measurement of quality, e.g. return the partial policy as soon as the probability of reaching goal is > 0.9.

SSPs can be extended to constrained shortest path problems (CSSPs) which allows us to bound the expectation of different cost functions. More complex constraints are also possible, for instance using probabilistic linear temporal logic (PLTL) formulae. Such constraints can be compiled into constraints over expectations (Baumgartner, Thiébaux, and Trevizan 2018), i.e. of the form $\mathbb{E}[\text{cost}(\phi)] \geq \rho$ where ϕ is a linear temporal logic formula, $\rho \in [0, 1]$, and the cost function is very similar to the reward function for maximising probability: it is zero everywhere except for a subset of the goal states. It is difficult to generate informative heuristics for this kind of problem. The upshot is that there are currently no strong heuristics, and so informed CSSP planners suffer.

This motivates that CoGNeRe may be a strong candidate for this type of problem, since it does not require heuristics for probabilistic problems, only heuristics for the deterministic pricing problem.

Another complex variant of probabilistic problems requires the solver to take into consideration the variance of trajectories as defined by the candidate policy. This is difficult for current solvers because they are designed around the construction and improvement of policies, and information about the possible trajectories needs to be extracted afterwards. CoGNeRe on the other hand, natively reasons about all possible trajectories in a compact, finite way, so it should offer a clean solution.

Heuristics For Planning Under Uncertainty Models

Rather than directly solving extensions of SSPs and CSSPs like MDPIPs (Shirota Filho et al. 2007), PLTL-constrained CSSPs, etc. we can explore the idea of solving relaxations with CoGNeRe in order to obtain heuristics for the original problem. We have already discussed CoGNeRe's ability to generate lower bounds; this combined with different pricing problem heuristics and techniques for reusing partial solutions may set CoGNeRe up as a good method for obtaining heuristics. CoGNeRe with its guarantees of optimality is likely to be too slow for this, so we can explore how to generate potentially non-admissible, informative heuristics using relaxation techniques from operations research, based on approaches like Dantzig-Wolfe decomposition and Benders decomposition.

Acknowledgments

Thanks to J. Christopher Beck for his feedback.

References

Ahuja, R.; Magnanti, T.; and Orlin, J. 1993. *Network flows : Theory, Algorithms, and Applications*. Englewood Cliffs, N.J: Prentice Hall.

Baumgartner, P.; Thiébaux, S.; and Trevizan, F. 2018. Heuristic Search Planning With Multi-Objective Probabilistic LTL Constraints. In *Proc. of 16th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*.

Bonet, B.; and Geffner, H. 2003. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *Proc. of 13th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Cherkassky, B. V.; and Goldberg, A. V. 1999. Negative-cycle detection algorithms. *Mathematical Programming*.

d'Epenoux, F. 1963. A probabilistic production and inventory problem. *Management Science*.

Desrosiers, J.; and Lübbecke, M. E. 2005. *A Primer in Column Generation*, 1–32. Boston, MA: Springer US.

Geffner, H.; and Bonet, B. 2013. A Concise Introduction to Models and Methods for Automated Planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*.

Geißer, F.; Keller, T.; and Mattmüller, R. 2015. Delete Relaxations for Planning with State-Dependent Action Costs. *Proc. of the Int. Symposium on Combinatorial Search*.

Geißer, F.; Keller, T.; and Mattmüller, R. 2016. Abstractions for Planning with State-Dependent Action Costs. *Proc. of 26th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? *Proc. of 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Katz, M.; and Sohrabi, S. 2020. Reshaping Diverse Planning. *Proc. of the AAAI Conference on Artificial Intelligence*.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong planning A*. *Artificial Intelligence*.

Shirota Filho, R.; Cozman, F. G.; Trevizan, F.; de Campos, C. P.; and Barros, L. N. 2007. Multilinear and Integer Programming for Markov Decision Processes with Imprecise Probabilities. In *Proc. of 5th Int. Symposium On Imprecise Probability: Theories And Applications*.

Teichteil-Königsbuch, F.; Infantes, G.; and Kuter, U. 2008. RFF: A robust, FF-based MDP planning algorithm for generating policies with low probability of failure. *Sixth International Planning Competition at ICAPS*.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *Proc. of 17th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

la VIDA: A System for Value and Identity Driven Autonomous Agent Behavior in Virtual World Scenarios – Dissertation Abstract

Ursula Addison
supervisor: John-Thones Amenyo
Graduate Center
City University of New York

Abstract

There are a great variety of systems that control agents using models of human behavior. However, often times agent action is still a reflection of the system designer's expectations and desired outcome. But, how would agents behave if they had an identity similar to a human? What goals would be formed and how would those goals be realized as actions? We would like to produce agent behavior using these questions as guidance for our work. To this end we investigate how long-term autonomy is influenced by an agent's identity and how these findings can be used to direct the behavior of artificial agents. For our system *la VIDA*, first we will create a model for human identity and ultimately integrate it with a Goal-Driven Autonomy (GDA) system at the drive level.

Introduction

There are a wide range of approaches and frameworks used to generate behavior for autonomous agents. One such approach is called Goal-Driven Autonomy (GDA). GDA systems use drives, which are more abstract than goals and encapsulate a general purpose or direction for the agent's actions (Muñoz-Avila 2018). Many GDA systems are developed to address highly specialized problems and thus simply formulate and manage goals from pre-assigned drives. We are interested in GDA systems from the perspective of agents that are autonomous and develop their drives internally from experiences, memories, impressions, and so forth. In short, our interests lie with agents that are intrinsically motivated to fulfill goals according to an inner sense of purpose. For this research we will design and build an identity profile, i.e. a representation of a mental identity taking inspiration from the human psyche. We will also investigate the connection between long-term autonomy and identity in humans. Our findings will be implemented in the **Value and Identity Driven Autonomy** system, i.e. *la VIDA*, to generate the behavior of artificial agents.

Motivation

Our work has a wide application scope. To test our system, *la VIDA*, we have developed the use-case which involves an interactive virtual reality (VR) scenario between a non-player character (NPC) and human agent. The NPC's behavior should be consistent with its identity profile and aid in the scenario's immersive experience. However, it is not difficult to imagine other situations in which our research could be useful. It might be used for generating agent behavior in simulations for video games, space exploration mission critical training, development and preparations in medicine, and education and virtual workplaces. Other applications include modeling the goals and actions of agents with a particular identity profile.

Current Behavior Approaches

There are a myriad of techniques for generating agent behavior, much more than could be covered in this short review. This summary will only produce a non-exhaustive list of broad categories in which many techniques fall into. These categories include: goal-directed, behavior trees, automaton, domain-independent planners, and hybrid systems.

An agent whose behavior is goal-directed is one who determines the goals it would like to achieve and the state of the world where those goals will be true (Muñoz-Avila 2018). The motivation to bring the world to a specific state is the drive of the agent. An agent may have a certain overarching goal that can be decomposed into to many simpler goals. These simpler goals may be fixed, but in a dynamic world new goals will need to be generated and achieved over time. Examples of this system type include the Goal Generation Management framework (GGM) (Hanheide 2010) and Goal oriented action planning (GOAP) (Orkin 2004).

A behavior tree (BT) is a data structure which arose from the gaming community's need for structures that were intuitive and robust (Iovino et al. 2020). BTs provide a direct and simple way of describing actions that a agent is capable of doing. The structure is a graph with compound tasks that decompose into smaller task subtrees (Winter, Hayes, and Colvin 2010). Task execution is processed bottom-up, completing component tasks to achieve the compound task. The Component Reasoner (Dill 2011) and an evolutionary behavior tree system (Nicolau et al. 2017) fall into this category.

Finite State Machines (FSM) are one of the earlier techniques used to control agent behavior and are ubiquitous in video game development. To execute an action and transition to a new state, an FSM must know its current state, this is accomplished by continually monitoring the environment. FSM are very useful for their highly reactive nature and the degree of control they give designers over the action and state space (Antimirov 1996). An environment may potentially have an infinite number of states, but a finite subset of those states are encoded as nodes of the FSM. When one of those states is recognized, a finite set of actions may be performed when in that state. Examples are classic deterministic FSM and a system using behavioral programming with probabilistic automata and personality by Chittaro et al. (Chittaro and Serra 2004).

Planning is a popular technique for controlling agents, particularly characters in video games. Most techniques used by game developers are some variation on search planners. The planners will normally be paired with a heuristic function that can incorporate some important consideration, e.g. plan construction time, as a way of ordering plans from least to most desirable (Wilkins 1984). A wide variety of STRIPS style and HTN planners have successfully been applied to agent control.

The last category is reserved for techniques that don't fit neatly into the above groups. One such system is the hierarchical task network and behavior tree hybrid planner developed by Neufeld et al. (Neufeld, Mostaghim, and Brand 2018). This planner combines the reactivity of behavior trees (BT) with the long-term strategizing of hierarchical task networks (HTN).

Research Goals

This paper describes early conceptual work, and our thesis is to implement preliminary work for *la VIDA*. Following is a discussion of our current research goals.

1. Can *la VIDA* use its input to formulate non-trivial, decomposable, relevant, and achievable goals?

This research goal describes the requirement that the goals generated are both non-trivial and relevant within context and a sequence of such goals can lead to the scenario being completed. For example, if one of the agent's drives is to "stay alive", it could be a correct but trivial goal to remain standing in a safe starting location. The goals should be achievable in that it is possible to plan for and execute them in the given environment. Finally, goals should be abstracted i.e. they shouldn't be terminal actions such as those sequenced by the planner.

2. If *la VIDA* generates a behavior sequence from a *la VIDA* formulated goal, can an agent autonomously execute the behavior sequence to achieve the goal?

This research goal is related to the first one, but emphasizes that goals should not be strictly theoretical; there must be a problem definition and environment for which the goals can actually be executed action by action. In addition, this action sequence should be achievable by the agent without any external interference.

3. Is the behavior sequence when executed by an agent realistic and consistent with its identity profile?

The final research question relates to the quality of the behavior sequence. Assuming it is valid and achievable, is it also believable? Meaning, is it consistent with norms and expectations that humans have regarding behavior resulting from certain beliefs, roles, and personalities? This research goal is not considering the correctness of the action sequence, but instead how it might be perceived by other agents and if those perceptions are in line with the agent's identity profile.

System Overview

la VIDA is made up of a collection of sub-systems, its chief functionality is to produce behavior sequences for a single agent via goal reasoning and planning in response to its identity profile and environment. *la VIDA* departs from other autonomy frameworks in that it focuses on the agent's mental and personality profile opposed to arbitrary drives specified by a system operator.

The first phase of processing involves the goal formulation module where drives and core goals are identified. The next phase involves interleaved planning and goal management. This planning includes strategic planning and also planning for each of the abstracted goals. The final phase determines how the terminal goals will be performed in the environment, executes those goals, and returns state information to *la VIDA*. Processing is not necessarily sequential and phases can be revisited as necessary. Following is a brief explanation of each system component. Figure 1 shows the basic components and how they interact with each other.

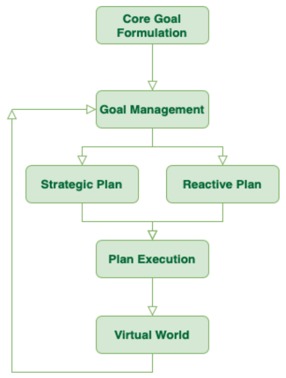


Figure 1: *la VIDA* Overview

Core Goal Formulation From a structured scenario and agent description *la VIDA* will formulate one or more core goals. Where a core goal is related to fulfilling an *integral drive* of the agent. As an example, if an agent is working for an airline in the new flight reservations department as a booking agent, a core goal could be to help each customer reserve new flights while behaving in a professional manner. Using the description which will include a role for the agent with *la VIDA*'s commonsense knowledge base, one or more core goals will be identified.

Goal Management While core goals should change very little, because they are integrally linked to the agent's mental and personality profile, subgoals may change often. Subgoals may change as a result of feedback from the environment or replanning. It is the goal management component's job to decompose core goals into subgoals and order those subgoals for the planner. Subgoals may be eliminated, added, deactivated or activated; each subgoal is given a priority value that also is subject to modification. The subgoals are eventually sent to the planner to be assigned a solution sequence. The solution sequence is an ordered list of actions that can transform the initial state into the goal state.

Strategic Plan The strategic plan also called a long-term plan is responsible for achieving goals over a sequence of actions. It is capable of making plans for subgoals that will be achieved in the somewhat distant future. Typically strategic plans aren't concerned with minute details and instead are are more abstracted than a reactive plan.

Reactive Plan The reactive plan is made of low-level actions assigned by the planner as the agent interacts in the world. By low-level actions we mean actions that can not be further decomposed and may directly be executed. These actions exist within the context of the strategic plan.

Augmenting Plans with Personality Traits and Emotions The agent has a mental and personality profile which is taken into account in the formulation of core goals and the reactive plan. Actions of the reactive plan may be augmented with tags that impact which animation, voice, or facial expression is ultimately selected when the action is being executed. We classify emotions according to the Ekman model of emotions which include: happy, sad, angry, fearful, disgusted, and surprised (Ekman 1993).

Common Sense Knowledge Base *la VIDA* will have access to a knowledge base similar to the commonsense knowledge graph (CSKG) ATOMIC-LIGHT knowledge base (Ammanabrolu et al. 2021). This knowledge base is a vast collection of common sense information and non-specialty knowledge that many humans may have. This will allow *la VIDA* to put the structured scenario and agent description into context. For instance, if the knowledge for a role is absent from the knowledge base, then *la VIDA* may not be able to formulate a core goal, and ultimately will not be able to generate any agent behavior. We may consider integrating functionality to add to the knowledge base or otherwise extending *la VIDA* dependence on a CSKG for this research or future work.

Virtual World Feedback The scenario is unfolding in real-time, as are the actions of the agents within it. To take actions that are effective and realistic, the feedback from the scenario must be used to update the world state information. This information is used by the strategic planner as it's plan is executed. For instance, if at some point the world state information deviates too much from the expected state, replanning may be necessary. The reactive planner is online and will constantly use scenario feedback in combination with the strategic plan to decide its own action sequence.

Agent The agent acts independently in the scenario, i.e. it's behavior sequence will be created via single agent planning. The agent will be able to interact with and respond to the environment and agents within it, but its actions will not require coordination. The agent will have one or more goals that can be decomposed into two or more top level goals. The agent's top level subgoals should be independent; those subgoals will need to be further decomposed so they can be planned for.

Identity Profile An identity profile is a data structure implemented as a container holding elements of type value, role, and personality. It is a summary of the most salient components of an agent's identity that are relevant to goal formulation. We will go into further detail about the identity profile in a system paper.

Usage

To use *la VIDA*, some assumptions, its input, and its output should be understood. The subsequent sections touch on each of these topics.

Preconditions

1. The structured scenario and agent description input has sufficient information for *la VIDA* to formulate a core goal.

2. A parent framework or environment manager creates and maintains the scenario returning accurate world state information to *la VIDA*.

3. Input has agent identity profile, scenario keywords, and planning problem definition.

Input We will determine the input structure and representation as our system is developed. This representation should minimally include partitioned elements for the *Agent* and *Scenario*, figure 2 is an example of one approach for structuring the input and the type of data it should hold. Some element types should only have a single instance such as *Agent*, *Scenario*, and *Role*; in future work we may increase the possible number of roles an agent may have. The other elements can have multiple occurrences, each with their row number within the parent element appended to their type. The agent may have multiple personality traits that constitute it's psychological profile, as well as multiple values that make up its mental behavior paradigm. Similarly, Scenario is expected to require multiple elements of each child type to sufficiently capture its theme and context.

```xml
<?xml version="1.0"?>
<Input>
    <Agent>
        <Role>data</Role>
        <Value0>data</Value0>
        <Value1>data</Value1>
        <Personality0>data</Personality0>
        <Personality1>data</Personality1>
    </Agent>
    <Scenario>
        <Theme0>data</Theme0>
        <Theme1>data</Theme1>
        <Context0>data</Context0>
        <Context0>data</Context1>
    </Scenario>
</Input>
```

Figure 2: Sample *la VIDA* XML input file

Output *la VIDA* output is a plan that consists of two subplans, one is the strategic plan and the other is the reactive plan. Both plans are a sequence of actions, where any action may be augmented with realism tags. The strategic plan is at a much higher level of abstraction than the reactive plan, and each action must be further decomposed before it can be executed. Each element of the reactive plan is a terminal action and may be executed directly. The strategic plan is typically generated offline, and modified online in the case of replanning. The reactive plan is generated fully online as the agent exists and interacts within the scenario.

References

Ammanabrolu, P.; Urbanek, J.; Li, M.; Szlam, A.; Rocktäschel, T.; and Weston, J. 2021. How to Motivate Your Dragon: Teaching Goal-Driven Agents to Speak and Act in Fantasy Worlds. *Proceedings of the 2021 Conference of the Association for Computational Linguistics: Human Language Technologies*, 807–833.

Antimirov, V. 1996. Partial derivatives of regular expressions and finite automaton, constructions. *Theoretical Computer Science*, 155(2): 291–319.

Chittaro, L.; and Serra, M. 2004. Behavioral programming of autonomous characters based on probabilistic automata and personality. *Computer Animation And Virtual Worlds Comp. Anim. Virtual Worlds*, 15: 319–326.

Dill, K. 2011. A Game AI Approach to Autonomous Control of Virtual Characters. *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*.

Ekman, P. 1993. Facial expression of emotion. *American Psychologist*, 48: 384–392.

Hanheide, M. 2010. A Framework for Goal Generation and Management. *Proceedings of the AAAI Workshop on Goal-Directed Autonomy*.

Iovino, M.; Scukins, E.; Styrud, J.; Ögren, P.; and Smith, C. 2020. A Survey of Behavior Trees in Robotics and AI. *Elsevier*.

Muñoz-Avila, H. 2018. Adaptive Goal Driven Autonomy. *Case-Based Reasoning Research and Development. ICCBR*, 11156: 3–12.

Neufeld, X.; Mostaghim, S.; and Brand, S. 2018. A Hybrid Approach to Planning and Execution in Dynamic Environments Through Hierarchical Task Networks and Behavior Trees. *Proceedings of the Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 14(1): 201–207.

Nicolau, M.; Perez-Liebana, D.; O'Neill, M.; and Brabazon, A. 2017. Evolutionary Behavior Tree Approaches for Navigating Platform Games. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(3): 227–238.

Orkin, J. 2004. Symbolic Representation of Game World State: Toward Real-Time Planning in Games. *https://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-006.pdf*.

Wilkins, D. 1984. Domain-independent planning Representation and plan generation. *Artificial Intelligence*, 22(3): 269–301.

Winter, K.; Hayes, I.; and Colvin, R. 2010. Integrating Requirements: The Behavior Tree Philosophy. *2010 8th IEEE International Conference on Software Engineering and Formal Methods*, 41–50.