Learning Problem Solving Skills from Demonstration: An Architectural Approach

Haris Dindo[1], Antonio Chella[1], Giuseppe La Tona[1], Monica Vitali[1], Eric Nivel[2], Kristinn R. Thórisson[2,3]

[1] Computer Science Engineering, University of Palermo (UNIPA/DINFO)
Viale delle Scienze, Ed. 6, 90100 Palermo, Italy
haris.dindo@unipa.it, chella@unipa.it, latona@dinfo.unipa.it, vitali@dinfo.unipa.it

[2] Center for Analysis & Design of Intelligent Agents and School of Computer Science (CADIA)
Reykjavik University, Menntavegur 1, IS-101 Reykjavik, Iceland
nivel@ru.is

[3] Icelandic Institute for Intelligent Machines
Uranus 2. h., Menntavegur 1, IS-101 Reykjavik, Iceland
thorisson@iiim.is

Abstract. We present an architectural approach to learning problem solving skills from demonstration, using internal models to represent problem-solving operational knowledge. Internal forward and inverse models are initially learned through active interaction with the environment, and then enhanced and finessed by observing expert teachers. While a single internal model is capable of solving a single goal-oriented task, it is their sequence that enables the system to hierarchically solve more complex task. Activation of models is goal-driven, and internal "mental" simulations are used to predict and anticipate future rewards and perils and to make decisions accordingly. In this approach intelligent system behavior emerges as a coordinated activity of internal models over time governed by sound architectural principles. In this paper we report preliminary results using the game of Sokoban, where the aim is to learn goal-oriented patterns of model activations capable of solving the problem in various contexts.

1 Introduction

Complex systems require a significant amount of work to be programmed and maintained. Research on *programming by demonstration*, or *imitation learning*, has recently been seen by the scientific community as a promising approach for simplifying programming efforts. It is inspired by the remarkable ability of humans to learn skills by simply watching others performing them. From the methodological point of view, programming by demonstration is an efficient method for pruning high-dimensional search space and it makes the task of learning problem solving skills computationally feasible. However, these advantages

come at the cost of increased architectural complexity and are still far from being solved in the scientific community.

In this paper we describe an architecture[4] for learning problem solving skills from demonstration and by active interaction with the environment. In contrast to *constructionist* A.I., in which complete specification of knowledge is provided by human programmers, ours is a *constructivist* A.I. approach, where an architecture continuously expands by self-generating code (e.g. models) [1]. We target *goal-level imitation*, where imitation is seen as the process of achieving the intention hidden in the observation of an action[2]. In other words, it is not the means that are imitated but rather the goal of a demonstration. Central to the computational modeling of such intentional actions is the idea that understanding others can be efficiently achieved by reenacting one's own internal models in simulation [3]. It is thought that similar mechanisms could lead to higher cognitive processes like *imagery, anticipation* and *theory of mind*[4, 5]

We are working towards an architecture that can incrementally learn sequences of internal model activations from demonstration. It is based on the idea of coupled forward-inverse *internal models* for representing goal-directed behaviors [6, 7]. A *forward model* is a predictor that, given the state of the system and a command, produces a prediction of the future outcome of the given command. An *inverse model*, known also as controller in control theory, produces command(s) necessary to reach a goal state given the present state. Internal models are powerful constructs able to represent operational knowledge of an agent and to govern its interaction with the environment. While a single internal model is capable of solving a single goal-oriented task, it is their sequence that enables the system to hierarchically solve more complex task.

The architecture is thus model-based and model-driven, and it solves all tasks by exploiting the set of such learned internal models. The system is designed to avoid the use of explicit planning: intelligent behavior emerges from the interaction of incrementally learned skills. Internal models encode various behavior of the system and an ad hoc module is responsible for coordinating the activation of the most significant internal models given the actual state of the system and the goal. In a similar way, learning of complex skills is achieved by composing and coordinating simple ones (as encoded by models).

The article is organized as follows. We first give a high-level overview of our architecture in sec. 2, and then we discuss how is the architecture used to learn skills by imitation in sec. 3. Section 4 explains how the architecture has been used to learn problem solving skills in typical A.I. test-bed, the Sokoban game. Finally, we outline the conclusions and future works in sec. 5.

[4] The architecture is being built by the HUMANOBS consortium as part of the HUMANOBS FP7 project led by CADIA/Reykjavik University; details of this architecture will be presented in future papers.

2 Architecture

While the architecture we are building will be extensively described in future publications, a cursory overview of its main components relevant to imitation learning will now be given. The basic building blocks of our architecture are internal models, both forward and inverse, which represent an agent's functional knowledge (i.e. they are executable), and are learned by the system through continuous interaction with the environment. The system is able to observe how the environment changes as a consequence of an event (both exogenous and self-produced) and to encode this causal relation between the current context and a proximal goal into an internal model. Complex chains of causal relations, linking the current context to a distal goal, are learned by adopting the paradigm of imitation learning where the agent focuses its computational efforts on interesting parts of the problem search space only.

Models operate on *states* of the world, including those regarding the agent itself. We will formalize the notion of the state shortly. For the discussion that follows, it suffices to stress that states need not to be complete (in a Markovian sense), nor uniquely defined. In this respect, states will be composed of facts that hold in the world.
The agent senses the world through *messages*, where each element of a message holds a true fact in the world as observed through a set of predefined perceptual processes. Each time a change is detected in the world (either as the effect of agent's actions or external phenomena) a message is generated. Messages are given in the form *marker-value*, where *value* can be of any kind (integer, boolean, string, etc.). Having introduced messages, the state can be defined as a collection of messages produced by a set of predefined perceptual processes related to the world (including the agent itself).

Internal models operate on states. An internal model is a structure containing a couple of pattern lists and a production. The pattern lists restrict the applicability of a model: a state can be input to a model only if its messages match the patterns on the list. The same structure is used to encode forward and inverse models. The qualifiers 'forward' and 'inverse' describe a pattern-matching-wise arrangement of said models and their inputs: shall the latter match the right-side, the model operates as an inverse model, a forward model otherwise.

Learning is initially triggered by domain-dependent knowledge stored in a component called *Masterplan*. It stores a set of primitive skills together with an initial ontology, goals and heuristics needed to monitor the learning progress. By directly interacting with the world the system generates hypotheses of new models through a component called *Model proposer*. These will be stored in Masterplan and tested in real situations in order to assess their usefulness.
The system learns new models by observing the world and interacting with it. The components depicted in Fig. 1 continuously analyze the perceptual data in order to acquire new knowledge. Initially the system interacts with the world executing casual actions in order to learn simple causal relations of the entities of the world and of itself - a process common in newborns called *motor babbling*

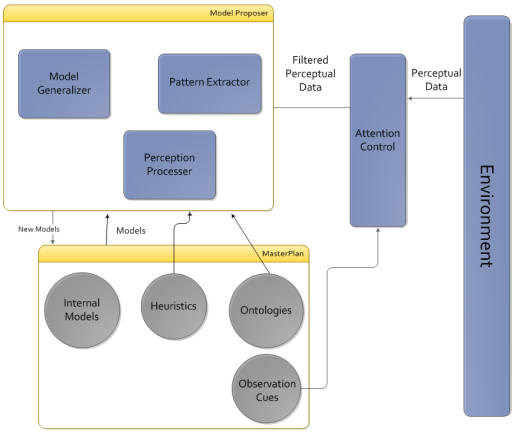


Fig. 1: How does the learning happen in the system? Environment is sensed through a set of existing models, and obtained percepts are analyzed in order to generate novel models by the "Model proposer" component. Models can be as simple as a composition of simple facts of the world, or complex sequences of existing models. This process is bootstrapped by existing knowledge stored in a component called 'Masterplan'. Generated models are themselves stored in the Masterplan for their validation and possible future usage. Interaction with the environment is performed through the Model execution component, which is directly connected with the world.

[8]. More complex models, encoded as chains of simple models activations, are learned by observing other skilled actors performing goal-directed tasks.

In order to exhibit robustness, the system must be able to generalize learned behaviors to novel, previously unseen, situations. To do this the system needs to reason about its own models and propose new, more general, models. In addition, the agent has to *anticipate* its future and to make decisions based not only on the actual and previous states of the system, but also on a prediction of a future state. Our architecture offers a mechanism of anticipation that is based on the knowledge encoded in the internal models.

When acting, the agent needs to decide what to do in a particular situation in order to achieve its goals. Fig. 2 depicts the components used to exploit models for acting. A *decision making* module, the Decision maker, is responsible of selecting which internal models to execute given the goal and the current situation. Decisions are made reactively and in parallel by exploiting all the available knowledge at present. Since multiple decisions can be made, we have developed a module that anticipates potentially useless or dangerous choices, and uses this information to decide which decision to execute. The Simulation module predicts the outcome of a decision by chaining the activations of internal models. The Decision maker uses the heuristics defined in the Masterplan to evaluate the desirability of a predicted future state. The Anticipation module analyzes the simulation looking for failures of the system or difficulties encountered. Conse-

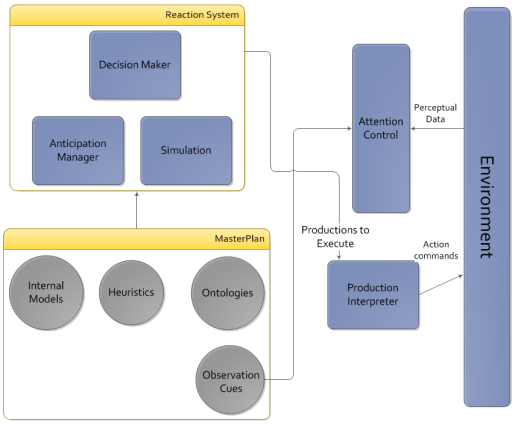


Fig. 2: How does the system act? Interactions with the environment are performed through the "Model execution" module which is controlled by a "Decision maker" component. The former is responsible for the coordination of acquired models of the system. The anticipation module is responsible for simulating the execution of models and their online correction.

quently, it might suggest anticipating a future production in order to avoid such situations.

3 Learning problem solving skills

Models encode problem-solving skills as chains of actions towards a goal, parameterized through patterns expressing states in which models are applicable (i.e. preconditions). For the discussion that follows, it is convenient to introduce a distinction between *low-level* and *high-level* models. In our definition, low-level models encode a *direct* causal relations between events the agent observes. For example, a low-level model could describe how a room illumination changes as a consequence of pushing the switch. On the other hand, high-level models prescribe actions needed to reach distal goals, or to predict the outcome of a present action arbitrarily far in the future. As an example, a high-level model can describe how to switch on the light in a different room from the one the agent is currently in. Such a scenario would require the agent to exit the room, reach the desired room, enter it, look for the switch and turn it on, where each act in the chain could be described by either a low- or high-level model.

The system tries to explain the events it observes in terms of its current repository of internal models by reenacting them in simulation. Whenever it fails, model learning is triggered which proposes new models. The Model proposer component produces a new low-level model by analyzing differences in state before and after an action has been executed, or an external event has been observed. We assume that each state transition can always be conveniently expressed as a combination of *elementary models* stored in the Masterplan. The

module called Pattern Extractor is responsible for generating the patterns on the messages that will be used to restrict the application of a model to situations similar to the observed one. Finally, the Model proposer produces a new forward model that predicts a state transition given an input. By inverting the forward model, the system produces its corresponding inverse model.

High-level models are learned through imitation learning. This process is accomplished by observing a demonstrator carrying out a task and trying to match its behavior with the set of available models.[5] To this end, forward and inverse models are used in couples to *rank* activations of those models that best explain the current observation (see [3] for details). A dedicated process is in charge of analyzing the sequence of activated models in order to detect *key states* encountered during demonstrations which will constitute sub-goals for the learning agent. The Model proposer then produces a sequence of sub-goals and patterns which compose the high-level model (patterns are produced in the same way as in low-level model acquisition).

However, newly acquired low- and high-level models are too specific since they have been learned from a single observation. As an example, suppose the agent learns how to switch on the light in a room. The model created is initially tailored to the particular room where the demonstration has been performed, as the agent has no means to assess whether the model could be applied in similar situations. A module called *Model generalizer* is responsible for the generation of new models that inductively generalize more specific ones. This process is triggered each time a model is created that shares the production section with a previously acquired model. If the only difference between these models is their pattern section, meaning that the same model can possibly be applied to both situations. A set of predefined rewriting procedures are applied that create a single, more general model.

3.1 Bootstrapping the learning process: Masterplan

Masterplan stores the domain-dependent knowledge that all the domain-independent components of the system use to produce new models and hence new knowledge. The Masterplan is not a fixed entity: it expands as the agent acquires novel knowledge through its own experience and learning.

In our system, prior knowledge in the Masterplan includes:

- a set of a-priori defined forward and inverse models; these models are augmented at runtime through the processes of motor babbling and imitation learning (the initial cannot be empty);
- a set of innate goals/subgoals and a monitoring process which provides the currently active goals/subgoals;
- facts about salient aspects in the world;
- an ontology which describes relations between entities in the world;

[5] Demonstrations should be performed in a bottom-up way: whether a task includes complex subtask, these should be thought first.

– a heuristic which evaluates the goodness of a state given a set of subgoals;
– a list of primitive actions the agent can perform: more complex behaviors will be hierarchically built starting from the same set of elementary ones.

Starting from these "innate" principles, the agent will be able to acquire knowledge by direct experience and to learn strategies through observation of other expert teachers. During interaction, only *relevant* aspects of the world are taken into consideration through a set of predefined attention mechanisms in the Masterplan. The Masterplan also holds predicates for assessing the similarity between two sets of messages, used to guide the learning phase.

New models are learned by combining innate *primitive* models provided in the Masterplan. In our architecture, these *primitive models* are defined by programmers as a set of *elementary functions* describing known facts about state transitions; a primitive model can e.g. relate position and velocity through known physical laws which combine elementary functions of multiplication and addition, and a set of *ontological relations* which describes how these functions can be applied to a given state (e.g. multiplication is not well defined for string variables).

4 Case study: Sokoban

In order to test the ideas described we chose a simplified version of the well-known Sokoban game as a case study, which presents a handy subset of our ultimate target application field(s) of the architecture. Sokoban is classified as a motion planning PSPACE-complete problem and as an NP-HARD space problem [9].

In our version of the game the agent moves a given number of blocks randomly placed in a grid; the goal is to place each block in a given final position. The number of blocks is a free parameter and can be set by the user. In our experiments we decided to use three blocks. An example of an initial grid configuration is shown in Fig.3 (left).

Whenever the agent performs an action, its perceptual sensors produce messages related to its position in the environment, and that of blocks. Messages indicating whether a block is next to another, or whether a block is next to the border are also given. The Masterplan holds a set of a-priori facts and models about the game. We defined *elementary functions* of the *primitive models* as mathematical and logical functions that compose any state transition as a consequence to an agent's act. These functions are *increment* and *decrement* for the numerical values, *negate* for logical values and the *identity* function that can be applied to any value. We have also defined ontological relations that specify how the elementary functions can be applied to the elements of a state (e.g. the increment and decrement functions can be applied to the coordinate of the blocks and of the agent). The Model proposer module analyzes the perceived transitions of state as a composition of elementary functions; the ontological relations are the constraints for this analysis.

The Masterplan also holds a heuristic to evaluate a state with respect to the desired goals. This heuristic is based on the Manhattan distance of blocks from

their desired position and on the measure of the degrees of freedom of both the agent and the blocks (in order to avoid deadlock configurations).

Instead of focusing on the computational costs of our approach and on comparing it to other Sokoban solvers, we have performed experiments aiming to assess the validity of our architecture as a general architecture for learning problem solving skills by imitation. We present results for various aspects of the architecture.

To test the results we consider how the agent predicts the outcome of an action in a set of defined states. This set of states was chosen to represent some of the most commonly encountered situations in the Sokoban game, together with few particular and rare configurations.

By analyzing the results of the motor babbling we provide an evaluation of the performances of the model acquisition and generalization processes. The parameters used in the motor babbling phase are: a) the number of actions to execute for each trial and b) the total number of trials. We have performed several tests by varying these parameters.

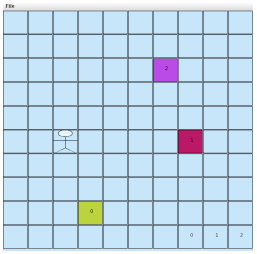
To evaluate models learned by imitation we have collected feedback from a group of randomly chosen subjects who have been asked to demonstrate a particular problem solving behavior to the system.[6] After the learning phase, each subject has been asked to grade the system's ability to solve similar tasks in a range of situations: a) whether the system was able to successfully complete the task and b) give a score from 1 to 10 related to the quality of the observed behavior, 10 being best and 0 meaning "no ability to perform".

4.1 Results: Motor babbling

For each run of the motor babbling, we store the predictions of the system for each of the encountered states. These predictions are then compared to the real outcome of the actions in corresponding states.

Through the motor babbling phase the system acquires its low-level internal models and learns how to interact with the environment. As shown in the experimental data(see table 3 (right)), the accuracy of the predictions grows with the number of trials played and actions taken. This could at first sound obvious: by increasing the number of trials we increase the amount of available data to analyze. However, it is worth noticing that the number of actions executed in a trial plays a marginal role compared to the total number of trials played. This can be explained by the fact that the motor babbling in a specific game tends to remain in states that are similar to the initial one. In order for the system to experience a wider range of situations, we need to increase the number of trials played.

[6] Participants were 6 male and 3 female PhD students from our lab; each participant played twelve trials on average.



(a) Sokoban game

	50	100	150	200
50	0.375	0.375	0.375	0.375
100	0.375	0.5	0.5	0.5
150	0.7	0.7	0.7	0.8
200	0.8	0.88	0.88	0.88
250	0.9	0.9	0.9	0.9
300	0.9	1	1	1

(b) Motor babbling results

Fig. 3: (left) A possible state in the Sokoban game; (right) Performance of our architecture in motor babbling: columns represent the number of steps in a trial, while rows represent the total number of trials; each cell contains the success ratio in predicting the correct outcome.

4.2 Results: Imitation

The results of the human demonstrator evaluation of subsequent system performance, after demonstrations, show that the system is able to learn new skills from the demonstration and to apply them in novel situations. Satisfaction analysis shows that more than 80% of participants judged the system's performances "more than sufficient" (a vote greater than 6). In particular, when the system was able to anticipate a production the evaluation was greater or equal than 8. This confirms that the anticipation ability is subjectively considered a necessary skill for any intelligent behavior.

5 Conclusions

In this paper we described new principles for learning complex problem solving skills through imitation. Our approach is based on *constructivist A.I.* principles, which proposes pervasive architectural self-modification as prerequisites for holistic system learning and self-expansion [1]. Our architecture allows self-expansion through a set of modules and a "Masterplan" that encodes initial bootstrapping knowledge to guide it. Before acting in the real world a system based on this approach runs actions in "simulation mode" using internal models, for the purposes of anticipation. These same set of models also enable our system to reach its goals, provided real-world experiences; our architecture allows the system to learn its internal models by observing other skilled actors.

As the whole architecture is model-based, learning is devoted to constantly acquiring new forward and inverse models. However, learning does not occur from scratch but it is rather bootstrapped by domain-dependent knowledge contained in the Masterplan; it holds the so called "first principles" that enables the system to learn more complex goal-directed behaviors. This approach has an obvious advantage over more traditional (i.e. hand-coded) architectures, as it

allows "goal-level" imitation, in which what is learned is the goal of the demonstration, rather than a particular sequence of acts to imitate. This is the most powerful way of learning, as the system acquires what amounts to an "understanding" of a set of actions - that is, the knowledge that the system acquires lends itself to explaining, which in turn (in our approach) allows the system to evaluate alternative explanations and choose between them based on available evidence.

Future work will focus on making the principles presented here more robust, expanding the architecture to be able to learn not only goals but also actively choosing which level of detail is appropriate to imitate, e.g. surface-level (morphology), goal-level (intention), or some combination thereof. Such a system should be applicable to a wide range of task learning scenarios, as many human-level tasks are in principle a complex mixture of the two. By having addressed the more difficult of these, namely goal-level imitation, we are optimistic about creating such a system in the near future.

Acknowledgments

This work has been supported in part by the EU funded project HUMANOBS: Humanoids That Learn Socio-Communicative Skills Through Observation, contract no. FP7-STREP-231453 (www.humanobs.org). The authors would like to thank the HUMANOBS Consortium for valuable discussions and ideas, which have greatly benefited this work.

References

1. K.R. Thórisson. From constructionist to constructivist AI. *AAAI Fall Symposium Series: Biologically Inspired Cognitive Architectures*, AAAI Tech Report FS-09-01:175–183, 2009.
2. A. Chella, H. Dindo, and I. Infantino. A cognitive framework for imitation learning. *Robotics and Autonomous Systems*, 54(5):403–408, 2006. doi: 10.1016/j.robot.2006.01.008.
3. D.M. Wolpert, K. Doya, and M. Kawato. A unifying computational framework for motor control and social interaction. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 358(1431):593, 2003.
4. R. Grush. The emulation theory of representation: Motor control, imagery, and perception. *Behavioral and brain sciences*, 27(03):377–396, 2004.
5. Y. Demiris. Prediction of intent in robotics and multi-agent systems. *Cognitive Processing*, 8(3):151–158, 2007.
6. M. Kawato. Internal models for motor control and trajectory planning. *Current opinion in neurobiology*, 9(6):718–727, 1999.
7. D.M. Wolpert and Z. Ghahramani. Computational motor control. *Science*, (269):718–727, 2004.
8. A.N. Meltzoff and M.K. Moore. Explaining facial imitation: A theoretical model. *Early development and parenting*, 6(34):179–192, 1997.
9. D. Dor and U. Zwick. SOKOBAN and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.