



AFRL-RI-RS-TR-2016-156

CURRICULUM DEVELOPMENT FOR TRANSFER LEARNING IN DYNAMIC MULTIAGENT SETTINGS

UNIVERSITY OF TEXAS AT AUSTIN

JUNE 2016

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2016-156 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

EDWARD VERENICH
Work Unit Manager

/ S /

JULIE BRICHACEK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) JUN 2016		2. REPORT TYPE FINAL TECHNICAL REPORT		3. DATES COVERED (From - To) FEB 2014 – FEB 2016	
4. TITLE AND SUBTITLE CURRICULUM DEVELOPMENT FOR TRANSFER LEARNING IN DYNAMIC MULTIAGENT SETTINGS				5a. CONTRACT NUMBER FA8750-14-1-0070	
				5b. GRANT NUMBER N/A	
				5c. PROGRAM ELEMENT NUMBER 62788F	
6. AUTHOR(S) Peter Stone, Jivko Sinapov, Matthew Taylor				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER R19U	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Texas at Austin Washington State University 2317 Speedway P.O. Box 642752 Austin TX 78712-1757 Pulman WA 99164-2752				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RISC 525 Brooks Road Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI	
				11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2016-156	
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Transfer learning in reinforcement learning has been an active area of research over the past decade. In transfer learning, training on a source task is leveraged to speed up or otherwise improve learning on a target task. This project addressed the ambitious problem of curriculum learning in reinforcement learning, in which the goal is to design a sequence of source tasks for an agent to train on, such that final performance or learning speed is improved. We take the position that each stage of such a curriculum should be tailored to the ability of the agent in order to promote learning new behaviors. To tackle the problem of curriculum learning, we addressed three key sub-problems: 1) Learning Transferability, and 2) Automatic Source Task Creation, 3) Curriculum Construction through Crowd Sourcing. This technical report documents the methods, experiments, and results of the proposed frameworks for curriculum construction for reinforcement learning agents.					
15. SUBJECT TERMS reinforcement learning, transfer learning					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 55	19a. NAME OF RESPONSIBLE PERSON EDWARD VERENICH
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) N/A

Contents

1	Summary	1
1.1	Learning Inter-Task Transferability	1
1.2	Automatic Source Task Creation	1
1.3	Crowd-sourcing for Curriculum Learning	1
2	Introduction	2
2.1	Learning Inter-Task Transferability	3
2.1.1	Related Work	3
2.2	Source Task Creation for Curriculum Learning	4
2.2.1	Related Work	5
2.3	Crowd-Sourcing Curriculum Design for RL Agents	5
2.3.1	Related Work	6
3	Methods, Assumptions, and Procedures	7
3.1	Background	7
3.2	Modeling Task Transferability	8
3.2.1	Notation and Problem Formulation	8
3.2.2	Predicting the Benefit of Transfer	9
3.2.3	Evaluation	9
3.3	Source Task Creation For Curriculum Learning	10
3.4	Search Space for Source Tasks	11
3.4.1	Task Dimension Simplification	11
3.4.2	Promising Initializations	12
3.4.3	Mistake-Driven Subtasks	12
3.4.4	Option-based Subgoals	13
3.4.5	Task-based Subgoals	14
3.4.6	Composite Subtasks	15
3.4.7	Summary	16
3.5	Curriculum Construction by non-Expert Humans	16
3.5.1	Language Learning with Reinforcement and Punishment	17
3.5.2	Curriculum Design Problem Formulation	18
3.5.3	User Study	19
4	Results and Discussion	20
4.1	Learning Inter-task Transferability	20
4.1.1	The Ms. Pac Man Domain	21
4.1.2	Experimental Methodology	21
4.1.3	The Transferability Matrix	24
4.1.4	Regression Model Performance	25
4.1.5	Source Task Ranking and Selection	26
4.1.6	Multi-stage Transfer	27
4.1.7	Summary and Discussion	28
4.2	Source Task Creation for Curriculum Learning	29
4.2.1	Ms. Pac-Man	30
4.2.2	Maze Simplification	30

4.2.3	Avoiding Ghosts	30
4.2.4	Half Field Offense (HFO)	31
4.2.5	2v2 HFO Curriculum	31
4.2.6	Extension to 2v3 HFO	34
4.2.7	Summary and Discussion	36
4.3	Curriculum Construction through Crowd-sourcing	36
4.3.1	Participant Performance	37
4.3.2	Concept Introduction	38
4.3.3	Transition Dynamics	39
4.3.4	Environment Preference	40
5	Conclusion	40
	References	43
	List of Acronyms	49

List of Figures

1	Different subgames in Quick Chess	2
2	An example five-room layout with one virtual dog, one chair, bag, and backpack with the same color in our domain. It is also the target environment (target command: “move the bag to the yellow room”) used in our user study.	16
3	A library of environments provided in a 4×4 grid. They are organized according to the number of rooms and number of objects. There is a command list for each of the 16 environments.	18
4	Screen shots of the game Ms. Pac-Man. In our experiments, the agent played 192 variations of the task, spanning 4 different mazes, shown above. The top-left image shows the configuration at the start of each game.	20
5	An example baseline test for one of the 192 tasks. The dark line indicates the reward averaged after 10 different runs (shown as the lighter lines), each starting with a different random seed. In this example, the policy converged after about 700 episodes.	22
6	An example transfer result for a given target task and two potential source tasks. Task A is clearly the better source task, resulting in a large positive transfer.	24
7	An example transferability matrix computed for each pair of the 192 tasks considered in our experiments. In this matrix, the entry at i, j amounts to the resulting <i>jumpstart</i> (30) measure after transferring the policy learned on task T_i to task T_j . Light values indicate high jump start while dark values indicate low (possibly negative) jump start.	25
8	Histograms of the jump start measures for two randomly chosen target tasks (i.e., a histogram over the values in a given column of the transferability matrix). For the first target task (top histogram), virtually all source task result in positive transfer, while for the second, there are a large number of source tasks that induce negative transfer.	25
9	Source Task Selection <i>loss</i> for three transferability measures. The two regression models were compared with the baseline source task selection model and with random source task selection.	26

10	Evaluation of source task ranking using the learned regression model and the baseline case-based reasoning approach. The ranking was evaluated using the Normalized Discounted Cumulative Gain (DCG_p) and the $jumpstart(m = 5)$ measure (the results were similar for the remaining values of m used in this study). The value for p , the number of elements to be considered in the ranking (starting at position 1) was set to 20.	27
11	Performance on a target task using one and two-stage transfer. The transfer curves are offset to reflect time spent training in their source tasks. In this example, all methods of transfer result in jump start but there is no benefit of two-stage transfer relative to single-stage transfer.	28
12	Examples of tasks in Ms. Pac-Man (a and b) and Half Field Offense (c and d). (a) Maze 1 (b) Maze 4 (c) HFO initial configuration and 2v2 dribble task (d) 2v2 shoot task. In HFO, offensive players are colored yellow, defensive players are blue, and the goalie is pink. The ball is shown by the white circle.	30
13	Results of TASKSIMPLIFICATION applied to the Ms. Pac-Man domain. See Section 4.2.2 for details. Dashed lines indicate standard error.	31
14	Results of MISTAKELEARNING applied to the Ms. Pac-Man domain. See Section 4.2.3 for details. Dashed lines indicate standard error.	32
15	Goal scoring accuracy on 2v2 HFO for agents following different curricula. Standard error (not shown to avoid clutter) ranged from 0.015 to 0.027 over the last 200 episodes for all curves.	39
16	Goal scoring accuracy on 2v3 HFO for agents following different curricula. Standard error (not shown to avoid clutter) ranged from 0.010 to 0.039 over the last 200 episodes for all curves.	35
17	The number of times each of four transitions being followed for each environment in the initial curricula in the two experimental conditions. There are 16 corresponding squares for 16 environments. The blue number represents the total number of times all four transitions being followed in each environment.	39
18	The probability of each environment being included in the initial or final curricula in the two experimental conditions. The purple circle represents the overlap of probability.	41

List of Tables

1	The Reward Structure of the Ms. Pac-Man Domain	21
2	The features that describe each task	23
3	Regression Model Performance measured by Correlation Coefficient	26
4	Reward structure in HFO	32
5	Feature space for the player with the ball in HFO. We index offensive players by their distance to the ball. Thus, the player with the ball is O_1 and its teammates are $O_2, O_3, \dots O_m$	33
6	Half Field Offense degrees of freedom	34
7	Summary of percentage of participants for different command selections in the gradually simple condition	37
8	Summary of percentage of participants for different command selections in the gradually complex condition	37

1 Summary

Transfer learning in reinforcement learning has been an active area of research over the past decade. In transfer learning, training on a source task is leveraged to speed up or otherwise improve learning on a target task. This project addressed the ambitious problem of *curriculum learning* in reinforcement learning, in which the goal is to design a *sequence* of source tasks for an agent to train on, such that final performance or learning speed is improved. We take the position that each stage of such a curriculum should be tailored to the current ability of the agent in order to promote learning new behaviors.

To tackle the problem of curriculum learning, we addressed three key sub-problems: 1) Learning Transferability, and 3) Automatic Source Task Creation, 3) Curriculum Construction through Crowd-Sourcing.

1.1 Learning Inter-Task Transferability

In a reinforcement learning setting, the goal of transfer learning is to improve performance on a target task by re-using knowledge from one or more source tasks. A key problem is choosing appropriate source tasks for a given target task. Current approaches typically require that the agent has some experience in the target domain, or that the target task is specified by a model (e.g., a Markov Decision Process) with known parameters. To address these limitations, we propose a framework for selecting source tasks in the absence of a known model or target task samples. Instead, our approach uses meta-data (e.g., attribute-value pairs) associated with each task to learn the expected benefit of transfer given a source-target task pair. To test the method, we conducted a large-scale experiment in the Ms. Pac-Man domain in which an agent played over 170 million games spanning 192 variations of the task. The agent used vast amounts of experience about transfer learning in the domain to model the benefit (or detriment) of transferring knowledge from one task to another. Subsequently, the agent successfully selected appropriate source tasks for previously unseen target tasks.

1.2 Automatic Source Task Creation

In many realistic scenarios, source tasks may not be available by default and instead, must be created on the fly. In such situations, the trainer must be able to create novel, agent-specific source tasks that can serve as a curriculum tailored towards an agent trying to learn a particularly difficult target task. We explore how such a space of useful tasks can be created using a parameterized model of the domain and observed trajectories on the target task. We experimentally show that these methods can be used to form components of a curriculum and that such a curriculum can be used successfully for transfer learning in 2 challenging multiagent reinforcement learning domains.

1.3 Crowd-sourcing for Curriculum Learning

Most existing work in curriculum learning focuses on developing automatic methods to iteratively select training examples with increasing difficulty tailored to the current ability of the learner, neglecting how non-expert humans may design curricula. In this project we introduce a curriculum-design problem in the context of reinforcement learning and conduct a user study to explicitly

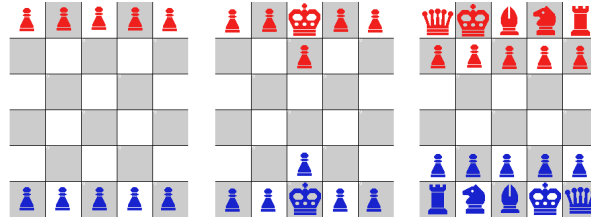


Figure 1: Different subgames in Quick Chess

explore how non-expert humans go about assembling curricula. We present results from 80 participants on Amazon Mechanical Turk that show 1) humans can successfully design curricula that gradually introduce more complex concepts to the agent within each curriculum, and even across different curricula, and 2) users choose to add task complexity in different ways and follow salient principles when selecting tasks into the curriculum. This work serves as an important first step towards better integration of non-expert humans into the reinforcement learning process and the development of new machine learning algorithms to accommodate human teaching strategies.

2 Introduction

As autonomous agents are called upon to perform increasingly difficult tasks, new techniques will be needed to make learning such tasks tractable. Transfer learning [24, 58] is a recent area of research that has been shown to speed up learning on a complex task by transferring knowledge from one or more easier source tasks. However, most transfer learning methods assume the set of source tasks is provided, and treat the transfer of knowledge as a one-step process. Paradigms such as multi-task reinforcement learning and lifelong learning consider learning multiple tasks, but typically focus on optimizing performance over all tasks, and/or still require the set of tasks to be provided.

The goal of this research was to extend transfer learning to the problem of *curriculum learning*. As a motivating example, consider the game of Quick Chess¹ (Figure 1). Quick Chess is a game designed to introduce players to the full game of chess, by using a sequence of progressively more difficult “subgames.” For example, the first subgame is a 5x5 board with only pawns, where the player learns how pawns move and about promotions. The second subgame is a small board with pawns and a king, which introduces a new objective: keeping the king alive. In each successive subgame, new elements are introduced (such as new pieces, a larger board, or different configurations) that require learning new skills and building upon knowledge learned in previous games. The final game is the full game of chess.

The question that motivates us is: can we find an optimal sequence of subgames (i.e. a curriculum) for an agent to play that will make it possible to learn the target task of chess fastest, or at a performance level better than learning from scratch?

We postulate that the effectiveness of such a curriculum depends crucially on the quality of the source tasks that compose it *and* the current learning abilities of the agent. As in Quick Chess, tasks should be designed to build upon existing knowledge and promote learning new skills. However, unlike Quick Chess, the tasks need not be the same for all agents.

¹http://www.intplay.com/uploadedFiles/Game_Rules/P20051-QuickChess-Rules.pdf

We tackled the problem of curriculum learning along several fronts. First, we propose a solution to the problem of *source-task selection* in which the agent learns to predict the benefit of transferring knowledge from one task to another. Second, we propose a set of methods for *source-task construction* in which a *trainer* creates source-tasks specifically adapted to an agent learning a difficult target task. Finally, we explored how non-expert humans create curricula for RL agents in a crowd-sourcing setting.

2.1 Learning Inter-Task Transferability

In a typical transfer learning scenario, the agent solves a *target* task by leveraging knowledge from 1 or more *source* tasks. An agent may transfer individual samples [55, 26, 25], a learned action-value function [57, 10], a policy [15, 14], or a model [36, 13]. Most current research in TL assumes that a good source task has already been identified [58]. The main limitation of the few existing approaches to source task selection is that they typically require the agent to already have some experience (e.g., training samples) in the target domain or for the target domain to be specified using a model (e.g., a Markov Decision Process) with known parameters.

To address these limitations, we propose a framework for selecting appropriate source tasks that uses meta-data – more specifically, attribute-value pairs – associated with each task. Our main hypothesis is that given parameters or attributes that describe two tasks, an agent may learn the benefits (or lack thereof) of transferring knowledge from one of them to the other. To test this hypothesis, we conducted a large-scale experiment in the Ms. Pac-Man domain in which the agent played over 170 million games spanning 192 variations of the task. For each source-target task pair, the agent measured the jump start in performance on the target task as a result of applying value-function transfer [59]. Subsequently, the agent learned a regression model of the transferability for any given pair and successfully used it to select appropriate source tasks for a new set of target tasks. To our knowledge, this is the largest computational experiment in transfer learning conducted to date.

2.1.1 Related Work

In recent years, research in transfer learning (TL) has improved the performance of reinforcement learning (RL) methods by enabling them to re-use knowledge from one task to another (see [58] and [24] for a review). For example, an agent may transfer individual samples [55, 26, 25], a learned action-value function [57, 10], a policy [15, 39, 14], or a model [36, 13]. In situations where the state and/or action spaces differ across tasks, an agent can learn an inter-task mapping [4] or use a hard-coded one provided by a human teacher [61]. Most TL methods assume that the source task has already been selected and that it is indeed a good source task for the target task [58]. In a more general case, however, an agent may have to choose appropriate source tasks on its own when faced with learning a complex novel task. This problem is referred to as *source task selection* and while it has been addressed to some degree in standard supervised machine learning domains (see [12, 41]), it has received relatively little attention in RL settings [58]. This problem is particularly important when some of the potential source tasks may be irrelevant to the target task, in which case the agent can suffer from negative transfer.

The few existing methods for source task selection typically assume that the agent has some experience (e.g., training samples) in the target task or that a model of the target task is available to the agent. For example, the method described by Lazaric *et al.* [26] enables an agent to select relevant samples from known source tasks by comparing them to samples collected in the target

task. In their experiments, the agent was able to effectively choose samples from two source tasks to speed up learning on the target task. A different approach to the problem is that of Nguyen *et al.* [36] where instead of transferring samples, the method transfers learned expectation models of how the environment changes as a result of the agent's actions. In that framework, the agent learns expectation models from a set of known source tasks and then dynamically identifies which of these models are useful when learning the new target task. Similarly, Perkins and Precup [37] describe an approach in which the agent learns reinforcement learning options on a set of source tasks and then uses them on a target task. While learning the target task, the agent estimated the value of known options by maintaining a belief about the target task's identity with respect to the known tasks.

Another model-based approach to the problem is described by Ammar *et al.* [5]. The authors propose a novel similarity measure for Markov Decision Processes which is shown to be effective at selecting good source tasks for a target task. Like other model-based approaches for transfer learning, the method proposed by Ammar *et al.* requires that the agent has access to a good estimate of the target task's MDP, which in practice may not always be available.

In contrast to existing methods for source task selection, we address the problem under the assumption that no samples from the target task are available. Instead, we consider the case where tasks are described using a fixed-length feature vector and thus, the agent is tasked with learning transferability across tasks using such meta-data. While most existing methods are evaluated only on a single target task, our empirical evaluation is conducted using a large-scale experiment in which the agent learns pairwise task transferability for a large number of tasks.

Another area of research that is relevant to this study is that of case based reasoning (CBR) [1, 29]. When faced with a new problem, CBR methods typically find similar problems (i.e., cases) that have already been solved in the past and re-use their solution. This is typically done by the use of a similarity function that can be used to identify relevant cases. The major limitation of applying CBR for source task selection is that it requires the similarity function to be indicative of whether or not transferring from one task to another will result in positive or negative transfer. Therefore, while we evaluate the approach of using task descriptors to compute task similarity and select sources accordingly, the work here proposes that an agent can learn to directly predict the outcome of transfer from the task descriptors.

2.2 Source Task Creation for Curriculum Learning

In many realistic scenarios, source tasks may not be available by default and instead, must be created on the fly. In such situations, the trainer must be able to create novel, agent-specific source tasks that can serve as a curriculum tailored towards an agent trying to learn a particularly difficult target task.

Thus, as a first step towards curriculum development, we focus on how to automatically construct a space of useful subtasks. Our approach uses knowledge of the problem encoded via a parameterized model of the domain, and observes the agent's performance on the target task and each prior task in the curriculum, in order to suggest new source tasks tailored to the abilities of the agent.

Our three contributions are as follows. First, we introduce the problem of curriculum learning in the context of reinforcement learning. Second, we propose a set of methods that can produce a space of agent-specific subtasks suitable for use in a curriculum. Third, we experimentally show that training using a curriculum has a strong impact on the learning speed or performance of an

agent, and that the sequence of tasks in the curriculum does matter. Furthermore, we demonstrate that the methods proposed can create such a curriculum, and be used successfully for transfer learning.

2.2.1 Related Work

Learning via a curriculum is an idea pervasive throughout human and animal training [46]. Recently, curriculum learning has also started to be explored in the context of supervised learning [6, 22], where the order in which individual samples are presented to an online learner was shown to considerably affect learning speed and generalization. Several related paradigms, such as multi-task learning [9] and lifelong learning [42], consider learning *groups* of *prediction* tasks. These methods assume tasks are related, and knowledge gained from solving one task can transfer to help learn another. In particular, Ruvolo and Eaton [41] show how a learner can actively select tasks to improve learning speed for all tasks, or for a specific target task. However, all of these works apply to supervised prediction tasks and assume the set of tasks to be learned is already given.

Subsequently, many of these ideas have been studied in the reinforcement learning paradigm. For example, Wilson et al. [68] explored multi-task reinforcement learning while Ammar et al. [3] consider lifelong learning applied to sequential decision making tasks. In both cases, a sequence of RL tasks is presented to a learner, and the goal is to optimize over all tasks. In contrast, our source tasks are designed solely to improve performance on a target task. We aren't concerned with optimizing performance in a source. In addition, neither work considers task *generation*, and thus are dependent on the quality of source tasks given.

In fact, as far as we know, we are the first to propose general methods for creating source tasks for transfer learning. Past work has typically relied solely on domain knowledge to supply suitable source tasks. For example, 3v2 keepaway serving as a source for 4v3 keepaway [57], 2D mountain car as a source for 3D mountain car [56], or varying parameters of physical systems [3]. Sinapov et al. [44] use a set of task descriptors, which are similar to our degrees of freedom, to specify a set of source tasks. This work goes significantly beyond all of those by creating agent-specific tasks from a dynamic analysis of an agent's performance, and shows how these tasks can be used in a multistage curriculum.

Finally, the problem of source task *selection*, which is different from task *generation* has been considered in single step transfer learning as well [23, 44]. As before, they assume the set of tasks is already prespecified, and the goal is to select the best ones. Again, these tasks are not individualized for each agent, and thus depend on the quality of tasks present.

2.3 Crowd-Sourcing Curriculum Design for RL Agents

Humans acquire knowledge efficiently through a highly organized education system, starting from simple concepts, and then gradually generalizing to more complex ones using previously learned information. Similar ideas are exploited in animal training [46]—animals can learn much better through progressive task shaping. Recent work [6, 22, 27] has shown that machine learning algorithms can benefit from a similar training strategy, called *curriculum learning*. Rather than considering all training examples at once, the training data can be introduced in a meaningful order based on their apparent simplicity to the learner, such that the learner can build up a more complex model step by step. The agent will be able to learn faster on more difficult examples after it has learned on simpler examples. This training strategy was shown to drastically affect learning speed and generalization [6, 22].

While most existing work on curriculum learning (in the context of machine learning) focuses on developing an automatic method to iteratively select training examples with increasing difficulty tailored to the current ability of the learner [22, 27], how *humans* design curricula is one neglected topic. A better understanding of the curriculum design strategies used by humans may lead to the development of new machine learning algorithms that accommodate human teaching strategies. Another motivation for this work is the increasing need for non-expert humans to teach autonomous agents new skills without programming. A number of published works in Interactive Reinforcement Learning [62, 20, 17] has shown that reinforcement learning (RL) [50] agents can successfully speed up learning using human feedback, demonstrating the significant role humans play in teaching an agent to learn a (near-) optimal policy. [53] first proposed that curricula should be automatically designed in an RL context, and that we should try to leverage human knowledge to design more efficient curricula [53]. As more robots and virtual agents become deployed, the majority of teachers will be non-experts. This work focuses on understanding non-expert human teachers—future work will investigate how to adapt machine learning algorithms to better take advantage of this type of non-expert curricula. We believe this work is the first to explicitly study how non-expert humans approach designing curricula for RL domains.

We are interested in studying whether humans can identify the concepts an agent needs to learn in the curriculum to complete a given target task. We hypothesize that humans gradually introduce more complex concepts to the agent within each curriculum. It is interesting to explore how humans increase task complexity and general principles regarding efficient curricula by analyzing the humans' design processes. If we can discover salient patterns within the curricula, we may be able to automate the *active selection* of suitable tasks in a curriculum or design new RL algorithms with inductive biases that favor the types of curricula non-expert human teachers use more frequently.

In this work, we task non-expert humans with designing a curriculum for an RL agent and evaluate the different curricula designs they produced. Specifically, we consider an RL domain in which an agent needs to learn to complete different tasks that are specified with textual commands in a variety of simulated home environments using reinforcement and/or punishment feedback. Human participants are told the target environment on which the agent will be tested on, and their goal is to select a sequence of training tasks that will result in the agent learning the target task as quickly as possible. Our results show that 1) most users successfully identified the two most important concepts the agent needed to learn to complete the target task when designing curricula, 2) users tended to gradually introduce more complex concepts to the agent within each curriculum, and even across different curricula, and 3) different users chose to increase task complexity in different ways and it was significantly affected by the ordering of the presentation of the source tasks. We also find some interesting salient patterns followed by most users when selecting tasks into the curriculum, which could be highly useful for the design of new RL algorithms that accommodate human teaching strategies.

2.3.1 Related Work

The concept of curriculum learning was proposed by [6] to solve the non-convex optimization task in machine learning more efficiently. Motivated by their work, considering the case where it is hard to measure the easiness of examples, [22] [22] developed a self-paced learning algorithm to select a set of easy examples in each iteration, to learn the parameters of latent variable models in machine learning tasks. Similarly, [27] proposed a self-spaced approach to solve the visual category discovery problem by self-selecting easier instances to discover first, and then gradually discovering new models of increasing complexity.

Although previous work has shown that machine learning algorithms can benefit from curriculum strategies [6, 22, 27], there is limited work on curriculum learning in the context of RL. However, there are several areas related to curriculum learning for RL. Wilson *et al.* [68] explored the problem of multi-task RL, where the agent needed to solve a number of Markov Decision Processes drawn from the same distribution to find the optimal policy. Sutton *et al.* [51] extended the idea of lifelong learning [63] to the RL setting, considering the future sequence of tasks the agent could encounter. Both cases assume a sequence of RL tasks is presented to a learner, and the goal is to optimize over all tasks rather than only the target task. The idea of active learning [11] was also exploited in RL domains [34, 64] to actively maximize the rate at which an agent learns its environment’s dynamics.

Of existing RL paradigms, transfer learning [58] is the most similar to curriculum learning. The main insight behind transfer learning is that knowledge learned in one or more source tasks can be used to improve learning in one or more related target tasks. However, in most transfer learning methods: 1) the set of source tasks is assumed to be provided, 2) the agent knows nothing about the target tasks when learning source tasks, and 3) the transfer of knowledge is a single-step process and can be applied in different similar domains. In contrast, the goal of curriculum learning is to design a sequence of source tasks for an agent to learn such that it can develop progressively more complex skills and improve performance on a pre-specified target task.

Taylor *et al.* [60] first showed that curricula work in RL via transfer learning by gradually increasing the complexity of tasks. Narvekar *et al.* [35] developed a number of different methods to automatically generate novel source tasks for a curriculum, and showed that such curricula could be successfully used for transfer learning in multiagent RL domains. However, none of their work explicitly investigates curriculum design from the perspective of human teachers. We think it is natural to consider what humans do when designing curricula since it might be easier for them to capture some examples that are “too easy” (*e.g.*, does not help to improve the current model) or “too hard” (*e.g.*, long training times are needed before the current model could capture this example) for the agent to learn. Such an idea has been studied in the context of teaching humans (*i.e.*, the *zone of proximal development* [65]) but not in agent learning.

By presenting a human-subjects experiment to explore how non-expert humans design curricula, this work represents an important step towards designing better machine learning algorithms that can accommodate human teaching strategies.

3 Methods, Assumptions, and Procedures

3.1 Background

A Markov Decision Process (MDP) $\mathcal{M}$ is defined by a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \Pi(\mathcal{S})$ is a transition function that maps the probability of moving to a new state given an action and the current state, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a reward function that gives the immediate reward of taking an action in a state, and $\gamma \in [0, 1)$ is the discount factor.

At each step, the agent is able to observe its current state, and must choose an action according to its *policy* $\pi : \mathcal{S} \mapsto \mathcal{A}$. The goal of an RL agent is to learn an *optimal policy* π^* that maximizes the long-term expected sum of discounted rewards. One way to learn the optimal policy is to learn the optimal action-value function $Q^*(s, a)$, which gives the expected reward for taking action a in state s , and following policy π^* after:

$$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{P}(s'|s, a) \max_{a'} Q^*(s', a')$$

Common temporal difference methods for learning the action-value function include Q-learning [49, 67] and Sarsa [45]. The optimal policy is then to choose $\arg \max_a Q^*(s, a)$ in each state. These temporal-difference methods are especially useful for problems with large and continuous state spaces which are challenging for approaches that directly try to learn the MDP. In this work, most of our experiments were conducted using the Sarsa algorithm [45]. We used Sarsa as a simple representative base learning algorithm, though in principle our methodology is equally applicable to any RL algorithm that learns a value-function.

Since the policy consists of taking the action with the highest action-value, transferring a policy is equivalent to transferring the action-value function. For example, if the function $Q^*(s, a)$ is represented using a parameterized function approximator, then value function transfer is achieved by using the parameters learned in a source task to initialize the function’s parameters in the target task. In other words, the agent starts learning the target task while acting under the policy learned in the source task. When a good source task is available, value function transfer has been shown to speed up learning by initializing the policy to something better than random exploration [57].

Common measures used to evaluate the result of transfer typically compare the learning trajectory on the target task after transfer with the trajectory that was produced by learning the target task from scratch [58]. In this work, we used the *jumpstart* measure to quantify transferability. This measure looks at the difference between the initial performance after transfer and the initial performance without transfer. Let $R^{baseline} \in \mathbb{R}^K$ be the reward curve after learning the target task for K episodes such that $r_k^{baseline} \in \mathbb{R}$ is the expected reward after learning for k episodes. Similarly, let $R^{transfer} \in \mathbb{R}^K$ be the reward curve for learning the target task after transferring a policy from the source task. The jump start metric can then be defined by:

$$jumpstart(m) = \frac{\sum_{k=1}^m (r_k^{transfer} - r_k^{baseline})}{m}$$

The parameter m determines the size of the temporal window which is used to compute the jump start after the onset of training on the target task. Other measures of transferability include asymptotic performance improvement (or detriment) as well as time-to-threshold [58].

3.2 Modeling Task Transferability

Following, we introduce the proposed data-driven framework for modeling inter-task transferability. The proposed framework described here is independent of the RL and TL methods that were described in the previous section.

3.2.1 Notation and Problem Formulation

Let $\mathcal{T}$ be the set of possible tasks. Let $\mathcal{T}_{source} \subset \mathcal{T}$ be a set of tasks for which the agent has learned a policy and let $\mathcal{T}_{target} \subset \mathcal{T}$ be another set of tasks that represents the set of target tasks to be learned by the agent. For each task $T_i \in \mathcal{T}$, let $F_i \in \mathbb{R}^n$ be a feature descriptor for the task that is known to the agent.

Given a target task $T_j \in \mathcal{T}_{target}$, the goal of the agent is to select a task $T_i \in \mathcal{T}_{source}$ such that T_i serves as an effective source for learning T_j . Thus, given a task pair, T_i and T_j , let $B(T_i, T_j) \in \mathbb{R}$

denote the benefit of transferring the policy learned in T_i to the task T_j , where $B(T_i, T_j) > 0$ indicates positive transfer, while $B(T_i, T_j) < 0$ indicates negative transfer. In this work, the transfer benefit is estimated using the jump-start measure defined in Section 3.1, though in principle, other measures can be appropriate as well.

We assume that for each pair of source tasks (T_i, T_j) such that $T_i, T_j \in \mathcal{T}_{source}$, the agent has a reliable estimate for $B(T_i, T_j)$. Next, we describe how the agent can use these estimates to predict the expected transfer benefit between tasks in $\mathcal{T}_{source}$ and tasks in $\mathcal{T}_{target}$.

3.2.2 Predicting the Benefit of Transfer

Here, the task of the agent is to learn a function which, given two arbitrary tasks T_i and T_j from $\mathcal{T}$, can predict whether T_i is a good source task for T_j . More specifically, the function should produce the estimate $\hat{B}(T_i, T_j)$, i.e., the expected benefit of transferring from T_i to T_j . Since $B(T_i, T_j) \in \mathbb{R}$, a natural solution for modeling the transferability between two tasks is to train a regression model.

Let $F_i = [f_{i,1}, f_{i,2}, \dots, f_{i,n}]$ and $F_j = [f_{j,1}, f_{j,2}, \dots, f_{j,n}]$ be the features for a pair of tasks (T_i, T_j) . To train a regression model on task pairs, a third feature vector is computed, X^{ij} , such that it captures some aspects of how the two feature vectors F_i and F_j are related. The feature vector X^{ij} was computed such that each element x_k^{ij} is defined by:

$$x_k^{ij} = \frac{f_{i,k} - f_{j,k}}{\max(f_{i,k}, \epsilon)}$$

where ϵ is a very small number to avoid divisions by 0. In other words, the vector represents the change along the n -dimensional features space relative to the feature values of the first task in the pair.²The function that computes how two tasks are related was designed to be sensitive to the order of the tasks in the pair since preliminary experiments suggested that task transferability is not always symmetric.

Given this representation and a dataset $\{X^{ij}\}_{T_i, T_j \in \mathcal{T}_{source}}$, a regression model M is trained such that:

$$M(X^{ij}) \approx B(T_i, T_j)$$

Once trained on pairs of tasks from $\mathcal{T}_{source}$, the regression model is subsequently used to select source tasks for the tasks in $\mathcal{T}_{target}$. Given a target task T_j , the task $T_i \in \mathcal{T}_{source}$ that maximizes $M(X^{ij})$ is selected as the source task. Next, we describe the performance measures that were used to evaluate the framework proposed here.

3.2.3 Evaluation

For each target $T_j \in \mathcal{T}_{target}$, the best possible source task is defined by:

$$T^* = \arg \max_{T_i \in \mathcal{T}_{source}} B(T_i, T_j)$$

Let T_i be the source task selected by the model. To compare the model's choice for a source task to the optimal source task, we define the *loss* as:

¹Other representations for the vector X^{ij} were explored as well, including raw difference (i.e., $f_{i,k} - f_{j,k}$) as well as ratio (i.e., $f_{i,k}/f_{j,k}$). Representations that captured the absolute or squared distance between F_i and F_j did not perform as well as they were not sensitive to the order of the tasks in the pair.

$$\text{loss}(T_i) = B(T^*, T_j) - B(T_i, T_j)$$

We also evaluated the ranking of source tasks induced by the regression model. For a given target task T_j , let $R_j = [T_{\{1\}}, T_{\{2\}}, \dots, T_{\{P\}}]$ be the ranked list of source task according to the learned regression model, i.e., $\hat{B}(T_{\{k\}}, T_j) \geq \hat{B}(T_{\{k+1\}}, T_j)$. For each position k in the ranking, let $\text{rel}_k = B(T_{\{k\}}, T_j)$ be a measure of the relevance of the result at that position. A common measure to evaluate the quality of a ranking is the Discounted Cumulative Gain (DCG) [18]:

$$\text{DCG}_p(R_j) = \text{rel}_1 + \sum_{k=2}^p \frac{\text{rel}_k}{\log_2(k)}$$

where $p \leq P$. The normalized DCG (NDCG) is computed by $\frac{\text{DCG}(R_j)}{\text{DCG}(R_j^{\text{best}})}$ where R_j^{best} is the true (i.e., best possible) ranking of source tasks. A normalized DCG of 1.0 would indicate a perfect ranking.

For a baseline comparison, we consider the naive approach of selecting the most similar task according to the feature vectors used to describe the tasks. In other words, given target task T_j , the naive method would select the source task T_i that minimizes the squared distance between F_i and F_j , i.e., $L_2(F_i, F_j)$. The baseline approach does not perform any learning but nevertheless, we hypothesize that it will perform better than randomly selecting a source task.

3.3 Source Task Creation For Curriculum Learning

So far, we have only considered the case in which the *trainer* has to pick an appropriate source-task from a currently existing pool such that the agent benefits from knowledge transfer to a difficult target task. Next, we address the setting in which the trainer must construct a source-task given a particular learning agent and a difficult target task.

In curriculum learning, the goal is to generate a sequence of source tasks $M_1, M_2, \dots, M_t$ for an agent to train on, such that the final asymptotic performance increases, or the learning time to reach a desired performance threshold decreases, versus following any other curriculum. As an important step towards this, we first define the domain $\mathcal{D}$ of possible tasks:

Definition 3.1. A domain $\mathcal{D}$ is a set of MDPs that can be expressed by varying a set of *degrees of freedom*, and applying a set of *restrictions*.

The *degrees of freedom* F of a domain are a vector of features $[F_1, F_2, \dots, F_n]$ that parameterize the domain. For example, in the Quick Chess domain, possible degrees of freedom could be the size of the board, the number of each type of piece, or whether special rules such as castling or en passant are allowed. Each $F_i \in F$ has a range of values $\text{Rng}(F_i)$ that represents the possible values that feature can take. Furthermore, we assume there is an ordering defined over each $\text{Rng}(F_i)$ that corresponds to task complexity. Collectively, these degrees of freedom encode our domain knowledge in the task.

An instantiation of F in $\mathcal{D}$ results in a specific task (an MDP). We assume we have a generator τ that can create tasks given a domain and degree of freedom vector:

$$\tau : \mathcal{D} \times F \mapsto M$$

By *restrictions*, we mean the set of tasks that can be formed by eliminating certain actions or states, modifying the transition or reward function, or changing the starting or terminal distributions of MDPs generated by τ .

Informally, $\mathcal{D}$ captures the universe of possible source tasks for use within the curriculum and could be potentially infinite in size. Our goal is to create a subset of tasks in $\mathcal{D}$ that might be suitable for learning a given target task, using knowledge of the domain, and tailored to the performance and abilities of the learning agent.

Formally, given a target task MDP M_t and trajectory samples X consisting of tuples (s, a, s', r) from following some policy π_t on M_t , the goal is to create suitable source tasks $M_s \in \mathcal{D}$ that will lead to a policy in M_t that is better than π_t . Specifically, we want functions f of the following form:

$$f : M_t \times X \mapsto M_s$$

The overall process we propose is an incremental development of subtasks culminating in a full curriculum: an agent first tries learning M_t , but gets stuck at suboptimal policy π_t . X is generated from π_t , and used to generate a space of possible source tasks tailored for *this* agent at *this* particular point in its learning process. For now, we assume a separate process is available to select a suitable source task M_s from this space, and leave for future work an automated way of finding it. The procedure then repeats, with M_s possibly becoming the new M_t , until a curriculum emerges.

3.4 Search Space for Source Tasks

In this section, we describe several methods that can serve as f to create suitable source tasks for a target task. Intuitively, there are many different ways in which a task could be a useful source for transfer to M_t : it could have a smaller or more abstract state space; it could have some actions removed; it could focus on a useful subgoal; or it could drill a common mistake. Some of these source tasks could be generated by simply manipulating the degrees of freedom F , and indeed we consider that case first. However, in the rest of the section, we define additional *domain-independent* instantiations for f .

3.4.1 Task Dimension Simplification

The first method we propose, TASKSIMPLIFICATION (Algorithm 1), simplifies a task using knowledge of the domain’s parameterization. Here, SIMPLIFY is a function that changes one of the degrees of freedom $F_i \in F$ to a new $F'_i \in \text{Rng}(F_i)$, in order to make the task smaller or easier. In many domains, there is a natural interpretation for SIMPLIFY. For example, in Quick Chess, we could reduce the value of parameters such as the size of the board or the number of specific pieces. In multiagent settings, we can add cooperative agents or remove adversarial ones.

Algorithm 1 Task Simplification

```

1: procedure TASKSIMPLIFICATION( $M, X, \mathcal{D}, F, \tau$ )
2:    $F' = \text{SIMPLIFY}(F)$ 
3:    $M' \leftarrow \tau(\mathcal{D}, F')$ 
4:   return  $M'$ 
5: end procedure

```

TASKSIMPLIFICATION transforms the S, A, P, R elements of an MDP simultaneously, in a domain-specific way.

3.4.2 Promising Initializations

The second method is designed for tasks that have a sparse reward signal. In many RL problems, positive outcomes can be rare, especially at the onset of learning. An agent may have to reach the goal randomly or through some exploration scheme many times before the policy stabilizes. PROMISINGINITIALIZATIONS creates a task that initializes an agent near states that were found to have high reward.

Algorithm 2 Promising Initializations

```

1: procedure PROMISINGINITIALIZATIONS( $M, X, C, \delta, \rho$ )
2:    $Y \leftarrow \{(s, a, s', r) \in X : r \geq \rho^{th} \text{ percentile of all rewards in } X\}$ 
3:    $M' \leftarrow M$ 
4:    $S'_0 \leftarrow \{\}$ 
5:   for  $(s, a, s', r) \in Y$  do
6:      $S'_0 \leftarrow S'_0 \cup \text{FINDNEARBYSTATES}(s, X, C, \delta)$ 
7:   end for
8:    $M'.S_0 \leftarrow S'_0$ 
9:   return  $M'$ 
10: end procedure

```

Here, the parameter $\rho \in [0, 100]$ is a percentile that defines the fraction of rewards an agent has seen in its experience trajectory X that it should consider to be positive outcomes. FINDNEARBYSTATES is a domain-dependent function that returns a set/distribution of states that are close to a given state, using either a distance metric $C : S \times S \mapsto \mathbb{R}$ or a pseudo-distance based on steps away in a trajectory. The exact form depends on the representation used for the MDP.

If the state space is factored, we can perturb the state vector by some amount δ such that the distance from the original state to the perturbed state (measured by C) is less than δ . In our Quick Chess example, if the state space consists of the positions of all pieces on the board, we can use a distance metric that measures the least number of “moves” needed to transform one board configuration to another. FINDNEARBYSTATES would return all configurations that are δ steps away. If the state space is not factored (for example, in a tabular representation), then we can use the trajectory samples X to find states that are at most δ steps away from a high reward state, and explore these further.

3.4.3 Mistake-Driven Subtasks

Our next set of methods create subtasks to help an agent avoid and correct its *mistakes*. In principle, a mistake is any action or sequence of actions (e.g., an option [48]) taken in a state that deviates from the optimal policy.

In practice, the agent does not know the optimal policy while learning, so we propose 3 alternative characteristics to automatically identify mistakes. The first is any action that leads to unsuccessful termination of an episode, such as not reaching a goal state. Second is any action that results in no change in state. Finally, a mistake could be any action that incurs a large negative

reward. In the following methods, we use `ISMISTAKE` to denote whether a mistake was detected, using these criteria.

Action Simplification

The first mistake-driven subtask generation method we propose, `ACTIONSIMPLIFICATION` (Algorithm 3), prunes the action set to create a subtask where mistakes are less likely.

Action set pruning is especially useful in settings where actions have preconditions for success. For example, a robot must grasp an object before manipulating it. An autonomous car must be standing still before opening the doors. Intuitively, any complex, multi-stage policy could benefit from this type of guided exploration.

Algorithm 3 Action Simplification

```

1: procedure ACTIONSIMPLIFICATION( $M, X, \alpha$ )
2:    $M' \leftarrow M$ 
3:    $count(a) = 0, \forall a \in A$ 
4:    $Y \leftarrow \{(s, a, s', r) \in X : ISMISTAKE(s, a, s', r)\}$ 
5:   for  $(s, a, s', r) \in Y$  do
6:      $count(a)++ = 1$ 
7:   end for
8:    $A' = \{a \in A : count(a) > \alpha\}$ 
9:    $M'.A = M'.A \setminus A'$ 
10:  return  $M'$ 
11: end procedure

```

The parameter $\alpha \in \mathbb{Z}$ is a threshold on the number of times an action should lead to a mistake before it is pruned. In practice, it may be useful to set these thresholds so that only one action is eliminated at a time, or only eliminated in certain states.

Mistake Learning

In contrast, the second approach, `MISTAKELEARNING` (Algorithm 4), directly tries to correct mistakes by rewinding the game back some number of steps, and having the agent learn a revised policy from there. Intuitively, focusing training on areas of the state space where the agent made a “mistake,” gives access to this experience much faster, allowing the agent to also learn to correct itself much faster.

The question of how far back in the trajectory to rewind is an interesting challenge in and of itself. For now, `REWIND` is a simple method that looks back ϵ steps from s in trajectory X , and returns the found state. However, in principle it could be more complex, based on the type of mistake made or the situation where it was made. In our example of Quick Chess, we could rewind the game to determine what should have been done differently to avoid a checkmate.

3.4.4 Option-based Subgoals

The next method creates subtasks for learning subgoals. The options literature [48] identifies many approaches to finding subgoals. Many take a state-based approach, where the learner tries to find states that may have strategic value to reach. For example, McGovern and Barto [32],

Algorithm 4 Mistake Learning

```
1: procedure MISTAKELEARNING( $M, X, \epsilon$ )
2:    $M' \leftarrow M$ 
3:    $S'_0 \leftarrow \{\}$ 
4:    $Y \leftarrow \{(s, a, s', r) \in X : \text{ISMISTAKE}(s, a, s', r)\}$ 
5:   for  $(s, a, s', r) \in Y$  do
6:      $S'_0 \leftarrow S'_0 \cup \text{REWIND}(X, s, \epsilon)$ 
7:   end for
8:    $M'.S_0 \leftarrow S'_0$ 
9:   return  $M'$ 
10: end procedure
```

identify subgoals as states that occur frequently in successful trajectories. Menache et al. [33] try to find “bottleneck” states. Simsek and Barto [43] seek to create subgoals for “novel” states, since they facilitate exploration of regions of the state space that the agent normally doesn’t reach. Finally, graph-based approaches such as Mannor et al. [31] identify states by clustering over a state-transition map.

OPTIONSUBGOALS (Algorithm 5) is designed to take any option discovery method (FINDOPTION) to create a subtask. Specifically, it creates a task to learn an option given the option’s termination set S_f and a pseudo-reward function R for completion. Since an option typically only involves a subset of the task’s complete state space, this subtask allows quick learning of how to reach important states. For example, in Quick Chess, capturing the queen would be an example of a useful subgoal.

Algorithm 5 Option Sub-goals

```
1: procedure OPTIONSUBGOALS( $M, X, V, \phi$ )
2:    $M' \leftarrow M$ 
3:    $(S_f, R) \leftarrow \text{FINDOPTION}(M, X, V, \phi)$ 
4:    $M'.S_f = S_f$ 
5:    $M'.R = R$ 
6:   return  $M'$ 
7: end procedure
```

Since our work takes place in the context of transfer learning, we introduce one additional option discovery method, FINDHIGHVALUESTATES (Algorithm 6), that uses high value states learned in a previous task as a subgoal. Specifically, it checks whether any of the learned values $V(s)$ for states encountered in our trajectory X exceed a threshold ϕ .

Instead of using trajectory samples X , we can also extract high value states directly from the value function. For example, with a tabular representation, we can simply lookup states of high value. With function approximation, an optimization routine would be used to solve for high value states.

3.4.5 Task-based Subgoals

An alternative to creating subgoals within an MDP is to create them directly at the task level. Specifically, we set the termination set S_f of the input MDP to be the initiation set S_0 of some

Algorithm 6 Find High Value States

```
1: procedure FINDHIGHVALUESTATES( $M, X, V, \phi$ )
2:    $S_f \leftarrow \{\}$ 
3:    $R \leftarrow M.R$ 
4:   for  $(s, a, s', r) \in X$  do
5:     if  $V(s) > \phi$  then
6:        $S_f \leftarrow S_f \cup s$ 
7:        $R(s, a, s') = V(s)$ 
8:     end if
9:   end for
10:  return  $(S_f, R)$ 
11: end procedure
```

other subtask, as shown in Algorithm 7:

Algorithm 7 Link Subtask

```
1: procedure LINKSUBTASK( $M, M_s, V$ )
2:    $M' \leftarrow M$ 
3:   for  $s' \in M_s.S_0, s \in M.S, a \in M.A$  do
4:      $R(s, a, s') = V(s')$ 
5:   end for
6:    $M'.S_f \leftarrow M_s.S_0$ 
7:    $M'.R \leftarrow R$ 
8:   return  $M'$ 
9: end procedure
```

For example, we can create a subtask that terminates where PromisingInitializations starts as follows:

$$\begin{aligned} M_1 &= \text{PROMISINGINITIALIZATIONS}(M_t, X, C, \delta, \rho) \\ M_s &= \text{LINKSUBTASK}(M_t, M_1, M_1.V) \end{aligned}$$

Applied to Quick Chess, this would create a task to reach configurations that are likely to lead to checkmate. The reward for reaching this terminal set is the value of the state in the subsequent task. This idea is similar to skill chaining [21], except that instead of learning options linking target regions to initiation sets, we link directly on tasks.

3.4.6 Composite Subtasks

Each of the previous subroutines f takes as input an MDP M_t and trajectory samples X , and returns a modified task MDP M_s . By passing the samples and resulting M_s as input to another function f , we can chain together arbitrary many subroutines to compose new source tasks.

Mathematically, let f and g be any two functions above. Assume we are given a target task MDP M_t and trajectory samples X from it. Then, the composite task $(f \circ g) = f(g(M_t, X), X)$, where for ease of exposition, we've left out the task specific threshold parameters.

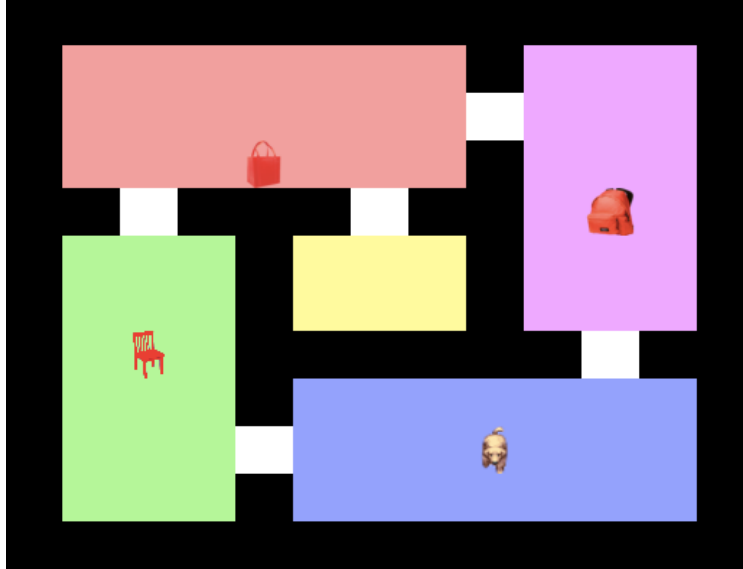


Figure 2: An example five-room layout with one virtual dog, one chair, bag, and backpack with the same color in our domain. It is also the target environment (target command: “move the bag to the yellow room”) used in our user study.

Most of the domain-independent functions described previously make specific modifications to a particular part of the target task MDP. In contrast, TASKSIMPLIFICATION can potentially make changes to the state and action space, as well as the transition and reward functions all at once. Thus, in practice, tasks should be composed using TASKSIMPLIFICATION first, followed by the others.

3.4.7 Summary

In summary, we presented several functions that could create suitable source tasks for a target task. They can be categorized into two types: the first (Section 3.4.1) allows for task creation using domain knowledge. The others are largely domain-independent, and rely directly on trajectory samples in the target task to create *agent-specific* tasks. We also showed how tasks of both types can be combined to create flexible source tasks for curriculum learning.

We claim that the functions outlined are broadly and generally useful. However, they are not the only possible methods; nor would every method apply to every domain. The next section moves on to experiments in domains for which we have concrete transfer learning results, using source tasks that can be generated with these functions.

3.5 Curriculum Construction by non-Expert Humans

The following section describes a sequential RL task with natural language command learning and introduces a curriculum design problem for non-expert humans.

3.5.1 Language Learning with Reinforcement and Punishment

Our domain is a simplified simulated home environment of the kind shown in Figure 2. The domain consists of four object classes: agent, room, object, and door. The visual representation of the agent is a virtual dog, since people are familiar with dogs being trained with reinforcement and punishment. The agent can deterministically move one unit north, south, east, or west, and pushes objects by moving into them. The objects are chairs, bags, backpacks, or baskets. Rooms and objects can be red, yellow, green, blue, and purple. Doors (shown in white in Figure 2) connect two rooms so that the agent can move from one room to another. The possible commands given to the agent include moving to a specified colored room (*e.g.*, “move to the red room”) and taking an object with specified shape and color to a colored room (*e.g.*, “move the red bag to the yellow room”).

In this sequential domain, the agent needs to learn to respond appropriately to different natural language commands in a variety of simulated home environments using reinforcement and/or punishment feedback. The learning algorithm for this study [30] connected the IBM Model 2 (IBM2) language model [7] with a factored generative model of tasks, and the goal-directed SABL algorithm [28] for learning from feedback. In SABL, feedback signals from a trainer are modeled as random variables that depend on the policy the trainer wants the agent to follow and the last action the agent took in the previous state. In general, reinforcements under this model are more likely than punishments when the agent selected an action consistent with the desired policy, and *vice versa* for punishment when the action was inconsistent. Using this model of feedback, SABL computes and follows the maximum likelihood estimate of the trainer’s target policy given the history of actions taken and the feedback that the trainer has provided. We adapted SABL to this goal-directed setting by assuming that goals are represented by MDP reward functions and that the agent has access to an MDP planning algorithm that computes the optimal policy for any goal-based reward function.

In contrast to previous work, we focus on studying how humans perform in designing curricula rather than in training the agent with reinforcement and punishment. Therefore, in this study, the human participants only choose the training curriculum, and the reinforcement and punishment on each of the curriculum’s tasks is carried out by an automated trainer, and is observed by participants.

Using this probabilistic trainer model and a curriculum from a human participant, an iterative training regime over each task in the curriculum proceeds as follows. First, the agent receives an English command. From this command, a distribution over the possible tasks for the current state of the environment is inferred using Bayesian inference. This task distribution is used as a prior for the goals in goal-directed SABL. The agent is then trained with SABL for a series of time steps, while the explicit reinforcement and/or punishment feedback is given at random times by the automated trainer. After completing training, a new posterior distribution over tasks is induced and used to update the language model via weakly-supervised learning. After the language model is updated, training begins on the next task and command from the curriculum.

As the agent learns additional tasks, it becomes better at “understanding” the language, successfully interpreting and carrying out novel commands without any reinforcement and punishment. For example, an agent might learn the interpretation of “red” and “chair” from the command “move the red chair,” and the interpretation of “blue” and “bag” from the command “bring me the blue bag,” thereby allowing correct interpretation of the novel command “bring me the red bag.”

3.5.2 Curriculum Design Problem Formulation

Here, we introduce a curriculum design problem for non-expert humans in our sequential RL domain, where the goal is to design a sequence of source tasks $M_1, M_2, \dots, M_n$ for an agent to train on such that it can complete the given target task M_t quickly with little explicit feedback. Each source task M_i is defined by a training environment, initial state, and a command to complete in that environment.

To aid our study of how humans form curricula for the agent to train on, we provided subjects a library of environments with different levels of complexities shown in the 4×4 grid in Figure 3. We organized the space of source environments a human could choose to include in their curriculum along two dimensions: the number of rooms and the number of moveable objects present in the environment. The cross product of these factors defines the overall complexity of the learning task, since these factors determine how many possible tasks the agent could execute in the environment and therefore how much feedback an agent could require to identify what the intended task is. For example, the environment in the top left of Figure 3 has the least complexity, because the only possible task the agent can complete is going to the yellow room. In contrast, the bottom right environment has the highest complexity, because the agent could be tasked with going to either the green, red, yellow, or blue rooms; or taking the bag, chair, or backpack to any of the rooms (excluding the room in which the object originates). For the ease of description, we number the environments in the grid from 1 (top left) to 16 (bottom right), from left to right and top to bottom. For example, the environments in the first row are numbered as 1–4 from left to right.

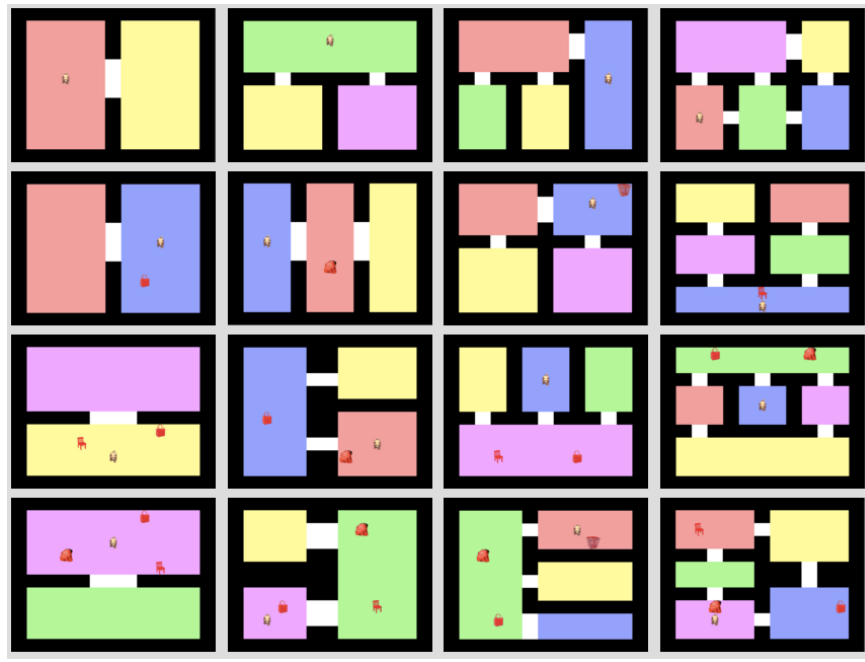


Figure 3: A library of environments provided in a 4×4 grid. They are organized according to the number of rooms and number of objects. There is a command list for each of the 16 environments.

After selecting an environment to include in the curriculum, users select the corresponding command to be taught in it from a predefined list of possible commands. For example, the possible commands for environment 5 (second row and first column of Figure 3) are “move to the red

room,” and “move the bag to the red room.”

The target task (shown in Figure 2) has the maximum number of differently colored rooms and shaped objects. In the user study, it is shown on the right side of the grid to remind users the goal of the designed curriculum, but it cannot be selected as part of the curriculum (enforcing a separation between training and testing).

Note that when we list the possible commands for each environment, we do not include the command that will be used in the target task (“move the bag to the yellow room”). That is, for any environment that contains a bag, the only possible command is “move the bag to the red/green/blue/purple room” even when there is a yellow room. We are interested in studying whether users can figure out that they can construct a curriculum that includes the command “move to the yellow room” and the command “move the bag to the red/green/blue/purple room” to provide the learning agent enough information to master the target command.

We varied the order of the 16 environments in the grid to study the effect of the ordering of source tasks on human performance in designing curricula. Specifically, we transposed the grid, swapping environments 1 and 16, 2 and 12, 3 and 8, *etc.*, such that the difficulty level of the environments gradually decreases from left to right, and top to bottom. Participants were assigned to one of two experimental conditions which varied the ordering of source tasks in the grid:

- **Gradually Complex Condition:** the number of rooms increases from left to right, and the number of objects increases from top to bottom (Figure 3).
- **Gradually Simple Condition:** the number of rooms gradually decreases from top to bottom, and the number of objects gradually decreases from left to right.

3.5.3 User Study

To study whether non-expert humans (*i.e.*, workers on Amazon Mechanical Turk, known as “Turkers”) can design good curricula for an RL agent, we developed an empirical study in which participants were asked to select a sequence of source tasks for an agent to train on such that it can complete the target task quickly with little explicit feedback.

In our user study, human participants must first pass a color blind test before starting the experiment since the training task requires the ability to identify different colored objects. Second, participants fill out a background survey indicating their age, gender, education, history with dog ownership, dog training experience, and the dog-training techniques they are familiar with. Third, participants are taken through a tutorial that 1) walks them through two examples of the dog being trained to help them understand how the dog learns to complete a novel command successfully using reinforcement and punishment feedback, and 2) teaches them how to design and evaluate a curriculum for the dog. Participants are told that 1) their goal is to design a sequence of source tasks the dog will train on such that the dog can successfully complete the given target task quickly, and 2) higher payment would be given to the Turker if the dog performs well in the target task.

Following the tutorial, participants are requested to select environments and corresponding commands in any order to design their own curricula. Recall that the target task is shown on the right side of the screen to remind participants of the goal for the designed curricula. Upon finishing designing a curriculum (containing at least one task), participants can choose to evaluate their curriculum, watching the automatic trainer teach the agent the entire curriculum. Then, participants are required to redesign the curriculum at least once. We ask participants to explain their strategy for designing the initial curriculum and what things they identified that the dog needed to learn

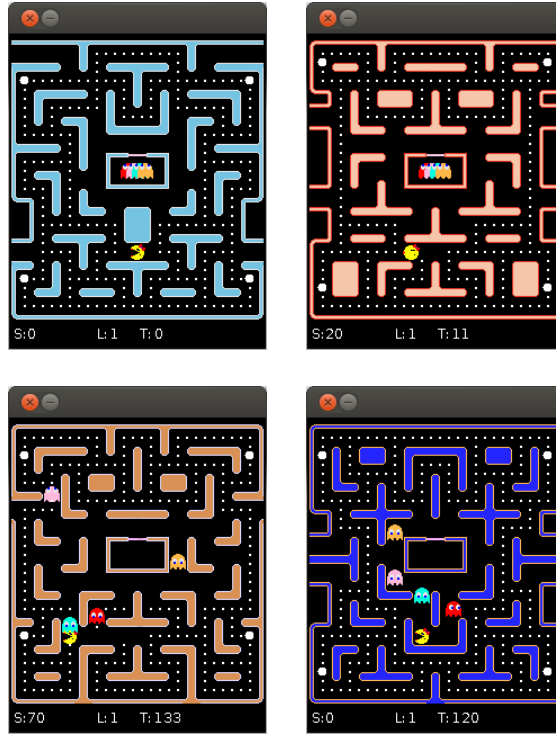


Figure 4: Screen shots of the game Ms. Pac-Man. In our experiments, the agent played 192 variations of the task, spanning 4 different mazes, shown above. The top-left image shows the configuration at the start of each game.

in the curriculum to successfully complete the target task. Participants were also required to explain how they redesigned the curriculum. Participants had the option of providing any additional comments about the experiment.

4 Results and Discussion

This section describes the experiments used to evaluate the proposed methods. More specifically, Section 4.1 describes the experimental evaluation of the proposed framework for learning inter-task transferability. Following, Section 4.2 describes a series of experiments demonstrating the methods proposed for source-task creation.

4.1 Learning Inter-task Transferability

To evaluate the proposed framework, we conducted a large-scale experiment in the Ms. Pac-Man domain. The following subsections describe the domain, the experimental methodology, and the results of the experiment.

Table 1: The Reward Structure of the Ms. Pac-Man Domain

Event	Reward (points)
Ms. Pac-Man eats a pill	10
Ms. Pac-Man eats a power pill	50
Ms. Pac-Man eats a ghost	200
Ms. Pac-Man eats an additional ghost while they are still edible	Apply a multiplier of 2 to the usual reward for each additional ghost that is eaten
Ms. Pac-Man is eaten by a ghost	Game Over

4.1.1 The Ms. Pac Man Domain

The framework for learning task transferability was evaluated using the Ms. Pac-Man domain, shown in Figure 4. The goal of the Ms. Pac-Man agent is to traverse a maze and earn points by eating edible items such as pills, while avoiding ghosts. The game typically starts with a large number of pills, four power pills located near each corner, and four ghosts that are initially placed in a lair that is inaccessible to Ms. Pac-Man. Shortly after the game starts, the ghosts leave their lair and may either chase Ms. Pac-Man or move about randomly. If a ghost catches Ms. Pac-Man, the game is over (we did not model the number of lives that are typically available to a human player). Whenever the agent eats one of the four power pills, the ghosts themselves become edible by Ms. Pac-Man for a short amount of time and their speed is reduced. If a ghost is eaten during that time, Ms. Pac-Man earns points and the ghost is sent back to the lair for a fixed amount of time, after which it starts to operate as normal. The agent’s action space consists of four actions, up, down, left, and right, though not every action is available in every state. Ms. Pac-Man eats pills, power pills and ghosts (when edible) whenever she gets within a small distance threshold of the object. Table 1 lists the rewards Ms. Pac-Man can get for different events in the game. The game ends when all the pills are gone, Ms. Pac-Man is eaten by a ghost, or 2000 time steps pass.

In our experiments, we used the Ms. Pac-Man implementation described by Taylor *et al.* [54]. The raw state space of the game is highly dimensional and also specific to each maze, thus making it unsuitable for learning. Therefore, in practice the state space in the Ms. Pac-Man game is typically represented by a set of local features that are ego-centric with respect to Ms. Pac-Man’s position on the board (see [40, 8, 52] for a representative sample of approaches). In this work, we used 7 heavily-engineered features defined in [54]. These features calculate properties such as the safety of junctions, and scores for the amount of pills and ghosts that could potentially be eaten along a certain direction. The agent learned the game using the Sarsa RL algorithm [45]. The action-value function was represented by a simple linear function approximator over those 7 features.

4.1.2 Experimental Methodology

We generated 192 variations of the Ms. Pac-Man task by varying several of the game’s parameters:

- **Maze:** each game was played on one of four different mazes, shown in Figure 4.
- **Number of ghosts:** the number of ghosts present in the game was varied from 1 to 4.

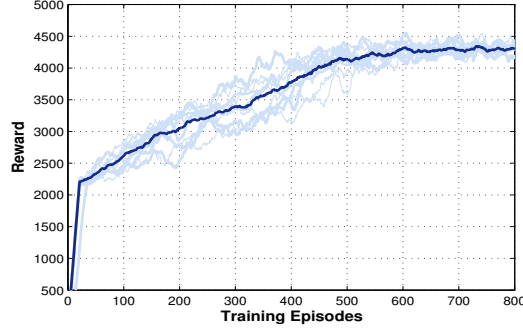


Figure 5: An example baseline test for one of the 192 tasks. The dark line indicates the reward averaged after 10 different runs (shown as the lighter lines), each starting with a different random seed. In this example, the policy converged after about 700 episodes.

- **Ghost slowdown:** when Ms. Pac-Man eats a power pill, the ghosts become edible and their movement speed is reduced. The *ghost-slowdown* parameter specified the amount of speed reduction and varied from 1 to 4, in increments of 1. When the Ghost slowdown is set to n , then the ghosts remain stationary every n^{th} game step when they are edible. Thus, a higher value makes the ghosts move faster, while a value of 1 makes them stop moving completely.
- **Ghost type:** the ghosts behaved according to one of three different modes: *Standard*, *Random*, and *Chaser*

The three different ghost behaviors are as follows: (1) *Standard ghosts* chase Ms. Pac-Man 80% of the time and move randomly the other 20%. When Ms. Pac-Man eats a power pill, the ghosts start moving away from the agent and eventually revert to their original behavior once they are no longer edible; (2) *Random ghosts* choose a random direction when reaching a junction 100% of the time. This makes it easier for Ms. Pac-Man to avoid them, but harder for Ms. Pac-Man to catch ghosts after eating a power pill. (3) *Chaser ghosts* have the same behavior as the Standard ghosts when inedible. However, after Ms. Pac-Man eats a power pill, they continue moving towards Ms. Pac-Man instead of fleeing. This makes it easy for Ms. Pac-Man to learn to eat ghosts (sometimes also too easy, since Ms. Pac-Man can learn to just stay in place and let the ghosts come to it, which does not transfer well to the normal setting).

Varying the four parameters resulted in $4 \times 4 \times 4 \times 3 = 192$ versions of the game. These 192 tasks constituted the full set of tasks $\mathcal{T}$. To compute transferability for all pairs of tasks, the agent first learned to play each task from scratch for 2,500 episodes (the number of total episodes was chosen such that the agent’s policy converged on each of the 192 tasks). Each episode consisted of playing a full game of Ms. Pac-Man. After each episode, the policy was frozen and the agent played an additional 10 games to compute a reliable estimate for the expected reward at each point during training. This procedure was repeated 10 times for each task in order to account for the stochastic nature of the domain. Thus, the agent played a total of $192 \times 2,500 \times (1 + 10) \times 10 = 50,800,000$ games to compute the baseline performance reward curves. Figure 5 shows an example baseline test for one of the 192 tasks. The bold line indicates the average reward curve from the 10 different runs.

Once the baseline curves were computed, the benefit of transfer was estimated for all task pairs. To do so, for each of the 36,672 pairs of tasks (T_i, T_j) in $\mathcal{T}$, the agent learned on task T_j for 30 episodes starting with the policy learned on task T_i (i.e., the agent transferred the policy from

Table 2: The features that describe each task

Feature	Description
<i>number-of-ghosts</i>	The number of ghosts
<i>ghost-slowdown</i>	The level of ghost speed reduction after Ms. Pac-Man eats a power pill
<i>ghost-type</i>	The behavior of the ghosts. There are three possible values: <i>Random</i> , <i>Standard</i> , and <i>Chaser</i> .
<i>num-nodes</i>	The number of nodes in the maze graph
<i>num-pills</i>	The number of regular pills in the maze
<i>distance-to-ghost</i>	The distance between Ms. Pac-Man and the ghosts at the start of the game
<i>distance-power</i>	The average distance between power pills
<i>distance-lair</i>	The average distance between the ghost lair and the power pills
<i>junctions-between-junctions</i>	The average number of junctions (i.e., nodes with more than 2 neighbors) that lie on the shortest path between any pair of junctions
<i>eccentricity</i>	The average eccentricity of nodes in the graph. The eccentricity for a node u is defined as $e(u) = \max\{d(u, v) : v \in V\}$ where d is the shortest-path function for a pair of nodes and V is the total set of nodes in the graph.
<i>eccentricity-junction</i>	The average eccentricity of junctions. The eccentricity for a junction node u is defined as $e(u) = \max\{d(u, v) : v \in J\}$ where $J \subset V$ is the set of nodes that are junctions.
<i>graph-diameter</i>	The diameter of the graph is defined as $\text{diam}(G) = \max\{e(u) u \in V\}$.
<i>num-nodes-d2</i>	Number of nodes with 2 neighbors
<i>num-nodes-d3</i>	Number of nodes with 3 neighbors
<i>num-nodes-d4</i>	Number of nodes with 4 neighbors

source task T_i to target task T_j). This process was repeated 10 times for each pair, such that in each run, a different one of the 10 policies computed during the baseline run was used as a starting point. Thus the agent played $36,672 \times 30 \times (1 + 10) \times 10 = 120,101,760$ games. The average reward with transfer and the average baseline reward over the first 30 episodes were then used to compute the jump start measure. The jump start measure requires a parameter m that denotes the size of the temporal window (in terms of number of episodes) to be used when averaging the rewards (see Section 3.1). We computed the jump start measure for $m = 1, 3, 5, 10, 15$, and the maximum, 30.

All told, to compute both the baseline reward curves as well as the transfer reward curves, the agent had to play over 170 million games. This type of an experiment would be next to impossible on a single computer and therefore, we used our department’s Condor Cluster system [16]. A learning episode typically took about 0.5 – 0.75 seconds, though this duration could vary depending on the cluster machine being used. Based on logged data, the experiment took over 2,300 hours of compute time spread over 192 individual machines. We believe that this is the largest

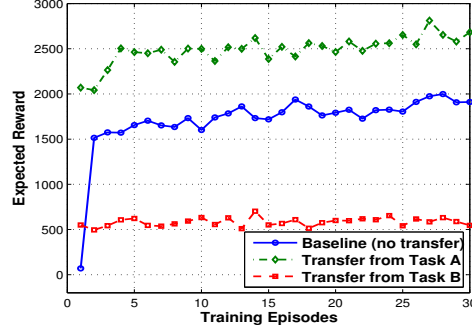


Figure 6: An example transfer result for a given target task and two potential source tasks. Task A is clearly the better source task, resulting in a large positive transfer.

computational experiment in transfer learning to date.

The framework for learning task transferability proposed in Section 3.2.2 requires that the agent has access to a real valued feature vectors that describes each task. Table 2 shows the task features that were used in our experiments. All of the features, except for *ghost-type*, are numeric. The *ghost-type* feature was originally nominal and therefore was converted into 3 different binary features, one for each type of ghost behavior. Thus, $F_i \in \mathbb{R}^{17}$. The features that were used to describe the tasks corresponded to the parameters used to generate the tasks, as well as graph-based features induced by the maze in each task. The features were not specifically selected or tuned to maximize performance. The graph-based features included domain specific attributes (e.g., the distance between Ms. Pac-Man’s starting position and the Ghosts’ lair) as well as general graph-based features such as *eccentricity* and a histogram of the nodes’ degrees (the last three features in the Table 2).

In our experiments, we explored two different implementations for the regression model M described in section 3.2.2: 1) Linear Regression, and 2) M5 Model trees [38]. Linear Regression was selected due to its simplicity, while the M5 Model tree was selected as it is able to handle non-linear problems. Both implementations can be found in the WEKA machine learning library [69]. The WEKA implementation uses a modified version of the original tree induction algorithm, called M5P [66] which added pruning as a part of the training stage.

4.1.3 The Transferability Matrix

Figure 6 shows an example transfer result for a target task and two different source tasks. In this case, transferring the policy from one of the source task to the target task results in positive transfer, while the other source task induces negative transfer. Figure 7 shows the whole transferability matrix computed for the set of 192 tasks considered in our experiments. In this example, each entry contains the expected benefit of transfer according to the *jumpstart*(30) measure for each pair of tasks (in other words, the jump start was computed over the first 30 training episodes on the target task). White values indicate high jump start while black values indicate low (possibly negative) jump start.

The order of the columns and rows of the matrix is not random but rather, the entries are sorted first according to the *maze*, then *ghost-type*, then *ghost-slowdown*, and then finally, *number-of-ghosts*. The last 1/4 set of columns in the matrix appear brighter than the rest because those tasks

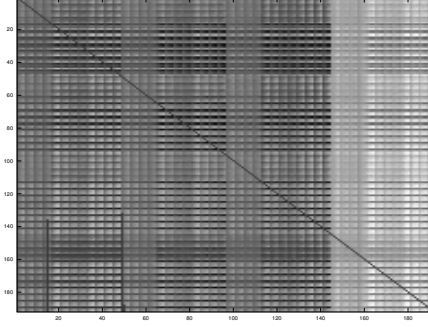


Figure 7: An example transferability matrix computed for each pair of the 192 tasks considered in our experiments. In this matrix, the entry at i, j amounts to the resulting $jumpstart(30)$ measure after transferring the policy learned on task T_i to task T_j . Light values indicate high jump start while dark values indicate low (possibly negative) jump start.

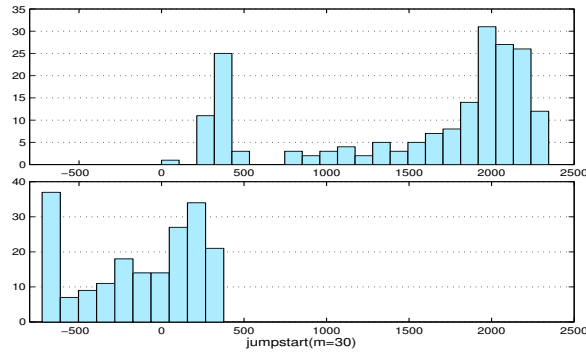


Figure 8: Histograms of the jump start measures for two randomly chosen target tasks (i.e., a histogram over the values in a given column of the transferability matrix). For the first target task (top histogram), virtually all source task result in positive transfer, while for the second, there are a large number of source tasks that induce negative transfer.

were much more likely to benefit from transfer. These tasks corresponded to tasks with the fourth maze, which proved to be much more difficult for the agent than the other three mazes. The grid-like pattern shows that transfer is not random and hence, we hypothesized that the parameters that define the tasks may be useful in predicting the benefit of transfer across tasks.

Figure 8 shows a histogram of the jump start measures for two randomly chosen target tasks (i.e., a histogram over the values in a given column of the transferability matrix). Even though the shapes of the histograms are similar, one of the target tasks is much more likely to benefit from transfer. For the first target task (top histogram), virtually all source tasks result in positive transfer. For the second target task, however, there are a large number of source tasks that induce negative transfer, which further motivates the need for effective source task selection.

4.1.4 Regression Model Performance

The performance of the regression model used to estimate transferability was evaluated using 10-fold cross validation at the task level. In other words, during each run, the tasks were split into 10 sets such that 9 of these formed the set $\mathcal{T}_{source}$ while the remaining fold was considered as the

Table 3: Regression Model Performance measured by Correlation Coefficient

Transferability Measure	Linear Regression	M5P Model Tree
$jumpstart(m = 1)$	0.54	0.74
$jumpstart(m = 3)$	0.64	0.85
$jumpstart(m = 5)$	0.65	0.87
$jumpstart(m = 10)$	0.66	0.87
$jumpstart(m = 15)$	0.65	0.86
$jumpstart(m = 30)$	0.61	0.83

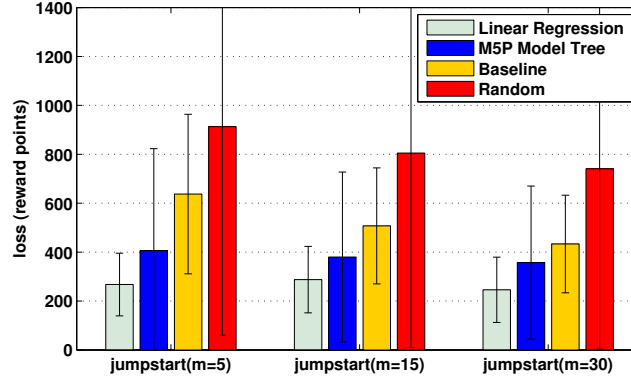


Figure 9: Source Task Selection $loss$ for three transferability measures. The two regression models were compared with the baseline source task selection model and with random source task selection.

set of target tasks $\mathcal{T}_{target}$. The regression model was trained on all pairs of tasks (T_i, T_j) such that $T_i, T_j \in \mathcal{T}_{source}$ and then tested on all pairs of tasks induced by the cross product of $\mathcal{T}_{source} \times \mathcal{T}_{target}$.

Table 3 shows the performance of the two regression algorithms that were used to predict the $jumpstart(m)$ measure for different values of m , the size of the temporal window used to compute the jump start. The results are reported in terms of the Correlation Coefficient (CC) between the actual and the predicted values. These results show that the difficulty of modeling task transferability depends on the measures used to estimate the benefit of transfer. For example, modeling the jump start after just 1 training episode on the target task is more difficult than modeling the jump start after 10 episodes on the target task. Overall, the CCs are high enough that we expect the ranking induced by the regression models to be useful for source task selection.

4.1.5 Source Task Ranking and Selection

Next, the proposed framework for source task selection was evaluated in terms of the expected $loss$, i.e., if the agent selects the source task that maximizes the expected transferability according to the regression model, how much worse does it do compared to selecting the optimal source task that it has already learned. Figure 9 shows the result of this test for two different regression algorithms, as well as the baseline approach. In addition, as a sanity check we computed the loss when randomly selecting a source task.

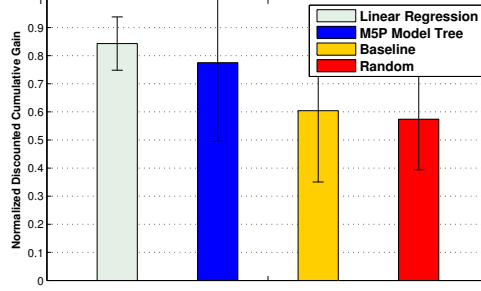


Figure 10: Evaluation of source task ranking using the learned regression model and the baseline case-based reasoning approach. The ranking was evaluated using the Normalized Discounted Cumulative Gain (DCG_p) and the $jumpstart(m = 5)$ measure (the results were similar for the remaining values of m used in this study). The value for p , the number of elements to be considered in the ranking (starting at position 1) was set to 20.

As we expected, the baseline approach which selects a source task based on task similarity in the task feature space performs better than randomly selecting a source task. Furthermore, the proposed method for learning task transferability substantially outperforms the baseline approach. While the Linear Regression (LR) model performed worse in terms of Correlation Coefficient when compared to the M5P Tree (M5P), the top source task selected when using LR tended to be a better source task than the one selected by M5P. The results so far were computed when approximately 172 tasks (i.e., 9 out of 10 folds) were available for training the regression model. An important question is whether performance would suffer as the training set becomes smaller. To obtain an answer, the number of tasks used to train the model was varied from 2 to 30 and we found that the expected loss converges after about 20 tasks (i.e., 400 pairs) are available for learning the regression.

The quality of the rankings were further evaluated using the Normalized Discounted Cumulative Gain measure. The results of this test are shown in Figure 10. Overall, LR performed the best. These results conclusively show that inter-task transferability can be learned even without samples or models of the target task. In particular, when faced with a new target task, a single good source task can be selected for transfer. These results naturally raise the question of whether it is possible to chain together multiple such source tasks sequentially to do even better. We examine that question next.

4.1.6 Multi-stage Transfer

In this section, we explore whether we can chain together a sequence of tasks $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_{target}$, such that learning T_1 makes it “easier” to learn T_2 , which makes it “easier” to learn T_3 , and so on. For simplicity, consider two stage transfer: we are looking for source tasks T_1 and T_2 such that transferring from $T_1 \rightarrow T_2 \rightarrow T_{target}$ gives better performance than training directly on T_{target} or any of the one-stage transfers $T_1 \rightarrow T_{target}$ and $T_2 \rightarrow T_{target}$.

Candidates for the tasks T_1 and T_2 can be determined recursively using the transferability matrix. We simply look at the column corresponding to the target task, and select the row (i.e. source task) that provides the best transfer. The selected task then becomes the column for the next recursive stage.

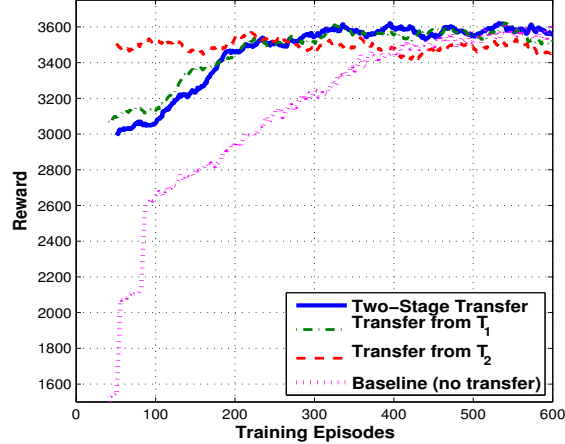


Figure 11: Performance on a target task using one and two-stage transfer. The transfer curves are offset to reflect time spent training in their source tasks. In this example, all methods of transfer result in jump start but there is no benefit of two-stage transfer relative to single-stage transfer.

A key question that we have not addressed so far is how to decide how many episodes to spend on each source task. Training on each of the subtasks T_1 and T_2 until convergence before transferring to T_{target} would be equivalent to just training on T_2 until convergence and performing single stage transfer to T_{target} . Therefore, in this preliminary test, we used a heuristic approach based on the intuition that an agent should train on a source task until additional training does not improve performance on the target.³

We hand-selected several of the more challenging tasks to serve as T_{target} . The results for one such target task are shown in Figure 11. All methods of transfer resulted in a jump start, but there was no benefit to using two stage transfer over single stage transfer. The results were similar for the other target tasks and overall, we were not able to find a two-stage transfer that was significantly better than its one-stage counterpart. Our hypothesis is that value function transfer is not suitable for two-stage transfer, since single stage transfer already initializes the policy in some area of the search space, and adding more stages does not noticeably refine this area. We leave for future work whether alternative RL and TL methods would facilitate finding a better two-stage transfer result.

4.1.7 Summary and Discussion

The framework for learning inter-task transferability was evaluated using a large-scale experiment in which the agent learned to play 192 variations of the Ms. Pac-Man game. To test our framework, the agent played over 170 million games, making this, to the best of our knowledge, the largest computational experiment in transfer learning conducted so far. Our results show that an agent can

² We define the target performance to be the total reward accumulated by the agent on the target task, for a fixed number of episodes (i.e. the area under the learning curve). Let A_{base} be the total reward accumulated by training directly on the target task without using transfer, and let $A_{transfer}^x$ be the total reward accumulated on the target task after training on the source task for x episodes, and using value function transfer. We used an incremental approach where the agent trained on the source task for 10 episodes, and used this to compute $A_{transfer}^x$. If the difference $(A_{transfer}^x - A_{baseline})$ was positive and increased, the agent trained on the source for 10 more episodes. This process was repeated until the difference no longer increased, at which point training on the source task was halted.

indeed learn to predict the transferability (as defined by the jump start measure) for an arbitrary pair of source-target tasks, provided training pairs for which the benefit (or detriment) of transfer is known. The learned transferability model was then used to effectively select relevant source tasks that improve the agent’s learning performance on a given target task.

There are several limitations and open questions that need to be considered for future work. First, our scenario considered the case where the agent transfers knowledge from just one individual source task. Future work will explore whether the learned transferability model is useful when the agent transfers knowledge from multiple source tasks instead. In addition, while efficiency was not addressed in this project, in practice generating a large set of data using every source-target pair is expensive. We found that only a small fraction of the source tasks are needed for the source task selection loss to converge and we believe that simple active learning frameworks can further reduce the number of task pairs needed to learn the transferability model. In our experiments, the agent learned the source tasks for the maximum amount of allowed time but we have also found that policies can successfully be transferred even when there is only limited exploration in the source task. Therefore, efficiency may also be improved if the agent can autonomously decide when to stop learning a source task and transfer the policy to a target task.

Evaluating efficiency in a setting like this poses additional challenges as it depends strongly on the number of potential target tasks to be solved by the agent in the future. To get a strong transfer result, the time (e.g., number of episodes) spent training on source tasks needs to be taken into account when comparing the performance with training without transfer. When there is only one target task, the amount of learning spent on source tasks is bounded by the amount of time required to learn the target task from scratch. Most recent frameworks for evaluating transfer assume that there is indeed only one target task [58] and therefore, there is a need to identify good measures for quantifying strong transfer results in the setting where the set of target tasks is large and potentially larger than the set of source tasks.

One aspect of the framework that makes it applicable in a wide variety of settings is that it is agnostic with respect to the reinforcement learning algorithm or transferring learning method being used. At the same time, this property limits the potential for deeper theoretical analysis. Another open question is whether a similar methodology can be used to discover and subsequently model two-stage transfer, i.e., situations in which the agent learns multiple source tasks in a precise order such that learning on the target task is improved. A follow-up result from this study is that two-stage transfer sequences are rare or perhaps non-existent in the task space we considered. Future work will examine whether this is a limitation of the domain, or a limitation of the transfer learning method (i.e., value-function transfer).

4.2 Source Task Creation for Curriculum Learning

In this section, we apply the methods described in Section 3.4 to create a curriculum in two challenging multiagent domains: Ms. Pac-Man and Half Field Offense. First, we demonstrate the effectiveness of domain-dependent and domain-independent subtasks in a simple one-stage curriculum (i.e. classic transfer learning paradigm) applied to Ms. Pac-Man. Then, in Half Field Offense, we utilize multiple functions from Section 3.4 to create a successful *multistage* curriculum for learning. Furthermore, we show that the sequence of tasks in a curriculum matters, and provide empirical evidence that such curricula can be formed recursively.

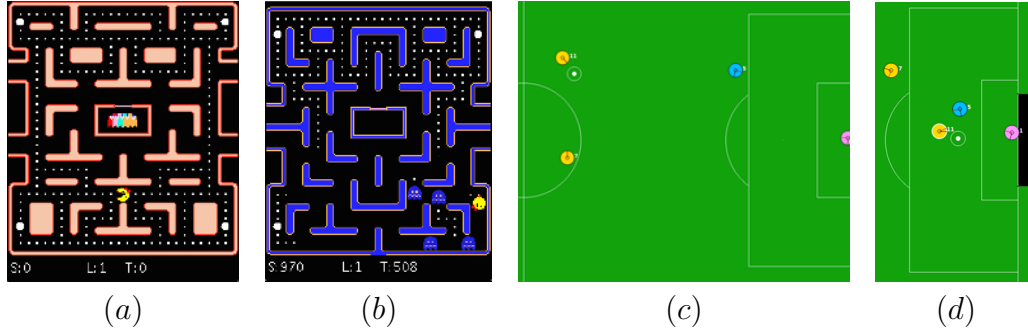


Figure 12: Examples of tasks in Ms. Pac-Man (a and b) and Half Field Offense (c and d). (a) Maze 1 (b) Maze 4 (c) HFO initial configuration and 2v2 dribble task (d) 2v2 shoot task. In HFO, offensive players are colored yellow, defensive players are blue, and the goalie is pink. The ball is shown by the white circle.

4.2.1 Ms. Pac-Man

Ms. Pac-Man (see Figure 12a and 12b) is a game where the agent’s goal is to traverse a maze and accrue points by eating objects such as pills, while avoiding the four ghosts. At the start of the game, there are a large number of pills throughout the maze, four power pills located at each corner, and four ghosts that are initially placed in an area inaccessible to Ms. Pac-Man. If a ghost catches Ms. Pac-Man, the game is over; however, if Ms. Pac-Man eats one of the four power pills, the ghosts themselves become edible by the agent.

We used the Ms. Pac-Man implementation described in [54, 44]. The agent’s state space was represented by a set of local features described in [54], that are egocentric with respect to the agent’s position on the board. Learning was done using Q-Learning [47], and transfer via value function transfer.

4.2.2 Maze Simplification

The first experiment is an application of the TASKSIMPLIFICATION method. The domain of Ms. Pac-Man comes with four different maze levels, some of which are easier for the agent to learn than the others. Thus, intuitively, one way to apply the TASKSIMPLIFICATION method is to train an agent on an easier maze and transfer the learned policy to a harder one. The results of such an application are shown in Figure 13. Here, the target task was maze level four (Figure 12b). The TASKSIMPLIFICATION principle was used to generate a source task by changing the maze level from four to one (Figure 12a). The transfer curve shows the effects of learning for 5 episodes on the source task and then learning for an additional 20 episodes on the target task. The baseline curve in contrast shows the result of learning for 25 episodes directly on the target task. Both curves are averaged over 20 runs. The results clearly show that applying TASKSIMPLIFICATION results in jumpstart and substantial improvement in the expected reward over the first 25 episodes.

4.2.3 Avoiding Ghosts

Next, we illustrate the use of an agent-specific source task, MISTAKELEARNING, in the Ms. Pac-Man domain. We consider a mistake to be the event where Ms. Pac-Man is eaten by a ghost, which

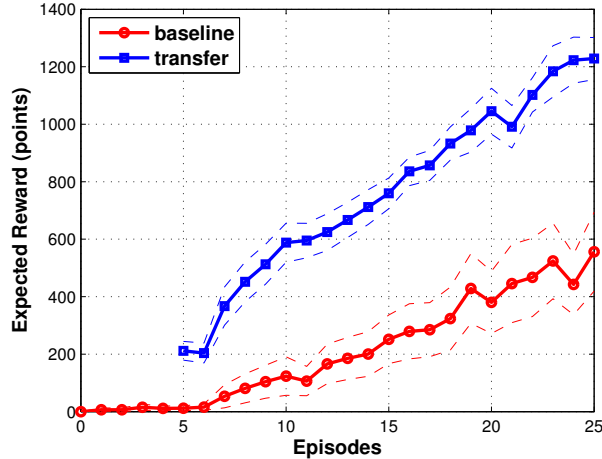


Figure 13: Results of TASKSIMPLIFICATION applied to the Ms. Pac-Man domain. See Section 4.2.2 for details. Dashed lines indicate standard error.

is a terminal non-goal state. Whenever a mistake occurs, we spawn the following task:

$$M_{mistake} = \text{MISTAKELEARNING}(M_t, X_t, \epsilon)$$

This call creates a subtask that rewinds $\epsilon = 50$ game steps from the moment the episode was terminated. The agent subsequently trains for 5 episodes in the generated subtask, after which training in the target task is resumed. The result of this test is shown in Figure 14. For this experiment, we measured the agent’s performance as a function of the number of game steps, since episodes spent on learning in the generated subtasks were much shorter. Results are averaged over 20 trials. The plot shows that the application of MISTAKELEARNING results in much faster learning when compared to the baseline approach of restarting each episode from the initial configuration upon episode termination.

So far, the two examples show that both domain-dependent and domain-independent methods can be used to generate effective source tasks for a given target task. The next set of experiments demonstrate how, in addition, they can also be used to design a curriculum for an agent learning a task that may be too difficult to learn from scratch, or even using a single source task.

4.2.4 Half Field Offense (HFO)

Half field offense [19] is a subtask of Robocup simulated soccer in which a team of m offensive players try to score a goal against n defensive players while playing on one half of a soccer field. The domain poses many challenges, including a large, continuous state and action space, coordination between multiple agents, and multiagent credit assignment. Each of these difficulties makes learning hard, especially early on when goal scoring episodes can be rare.

Each HFO episode starts with the ball and offensive team placed randomly near the half field line. Likewise, the defensive team is randomly initialized near the goal box. A sample starting configuration can be seen in Figure 12c. The goal of the offensive team is to move the ball up the field while maintaining possession, and take shots to score on goal. An episode ends when either (1) a goal is scored, (2) the ball goes out of bounds, (3) the defense captures the ball, or (4) the episode times out. The reward structure of the domain is shown in Table 4.2.4.

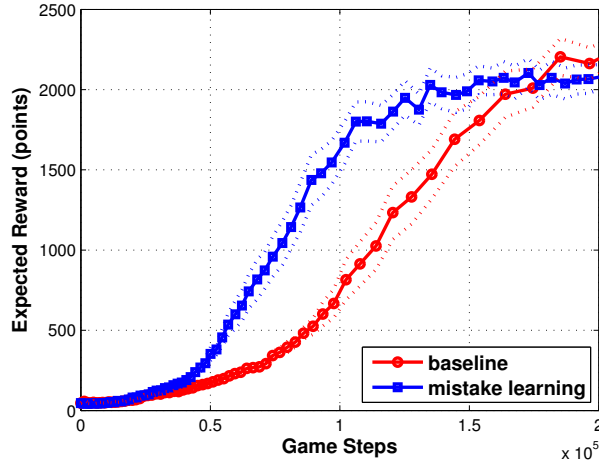


Figure 14: Results of MISTAKELEARNING applied to the Ms. Pac-Man domain. See Section 4.2.3 for details. Dashed lines indicate standard error.

Table 4: Reward structure in HFO

Event	Reward
Goal	1.0
Ball out of bounds	-0.1
Ball with offense	0
Ball captured by defense	-0.2
Ball captured by goalie	-0.1
Episode times out	-0.1

As done in Kalyanakrishnan et al. [19], we focus on learning behaviors for the player with the ball. The player with the ball has to choose one of the following actions:

- Pass k : A direct pass to the teammate that is k -th closest to the ball, where $k = 2, 3, \dots, m$.
- Dribble: A small kick in the cone formed between the player and the goalposts, that maximizes its distance to the closest defender also in the cone.
- Shoot j : A full power kick towards one of j evenly spaced points on the goal line.

Offensive players without the ball follow one of several fixed formations to provide support. The agent’s state space consists of distances and angles to points of interest, which are listed in Table 4.2.4. We used CMAC tile coding for function approximation, Sarsa for the learning algorithm [47], and value function transfer to transfer knowledge.

Table 5: Feature space for the player with the ball in HFO. We index offensive players by their distance to the ball. Thus, the player with the ball is O_1 and its teammates are $O_2, O_3, \dots O_m$.

Feature	Description
<i>dist-to-goalie</i>	Distance from O_1 to the goalie
<i>dist-to-defender-in-cone</i>	Distance from O_1 to the closest defender in the dribble cone
<i>dist-to-teammate_i</i>	Distance from O_1 to each teammate O_i , for $i = 2, 3, \dots m$
<i>dist-teammate_i-to-closest-defender</i>	For each O_i , the distance to its closest defender, $i = 2, 3, \dots m$
<i>dist-teammate_i-pass-intercept</i>	For each O_i , the shortest distance between a defender and the line between O_1 and O_i , $i = 2, 3, \dots m$
<i>min-ang-teammate_i-defender</i>	For each O_i , the smallest angle between O_i , O_1 , and a defender, $i = 2, 3, \dots m$
<i>dist-to-shot-target_i</i>	Distance from O_1 to location i on the goal line, $i = 1, 2, \dots j$
<i>dist-goalie-to-shot-target_i</i>	Distance from goalie to location i on the goal line, $i = 1, 2, \dots j$
<i>dist-shot_i-intercept</i>	Shortest distance between a defender and the line between O_1 and location i on the goal line, $i = 1, 2, \dots j$
<i>ang-goalie-shot-target_i</i>	Angle between goalie, O_1 , and location i on the goal line, $i = 1, 2, \dots j$
<i>ang-defender-shot-target_i</i>	Smallest angle between a defender, O_1 , and location i on the goal line, $i = 1, 2, \dots j$

Space of tasks

Half field offense has a number of degrees of freedom that allow creating many different types of tasks. We list some of the relevant degrees of freedom in Table 4.2.4. In addition to these, various aspects of the field (such as the size of the goals, the goal box, etc.), the players (such as visibility, stamina, etc.), and the world physics can also be changed.

These degrees of freedom allow us to quickly create many domain-specific source tasks, using the TASKSIMPLIFICATION rule. For example, we can add more teammates or reduce the number of defenders to give the offense more options. We can change the defensive team behavior to train against opponents of varying difficulty. We could also change various aspects of the world size and physics to make scoring and movement easier.

However, we can also create agent-specific source tasks by observing the behavior of the agent on the target task. For example, after observing generally unsuccessful trajectories on the target task, we could use MISTAKELEARNING to recreate situations where the agent lost the ball or failed to score, in order to learn how to avoid or resolve them. Another option would be to build upon successful trajectories using PROMISINGINITIALIZATIONS, which would create tasks that initialize the offense at different positions near the goal, allowing them to drill on how to shoot.

Combined, the methods from the previous section form a space of tasks that can be used to create a curriculum. In the next section, we illustrate the formal specification of some of the tasks

Table 6: Half Field Offense degrees of freedom

Parameter	Range
Number Offense Players	$\{0, 1, \dots 4\}$
Number Defense Players	$\{0, 1, \dots 5\}$
Defense Behavior	$\{\text{Agent-2D, Helios, WrightEagle}\}$
Formation Type	$\{\text{Flat, Box, Trapezoid}\}$
Field Width	20 – 68
Field Length	20 – 52.5
Max ball speed	0 – 5
Max player speed	0 – 1
Wind Noise	0 – 1

and their creation that we found to be useful in our experiments.

4.2.5 2v2 HFO Curriculum

We first consider the target task of 2v2 half field offense, where 2 attackers must score against 1 defender and 1 goalie. We used agents from the released binaries of the Helios team to form the defensive team [2]. Helios and WrightEagle consistently place among the top teams in the annual Robocup 2D Simulation League tournament, making even this small version of half field offense a challenging task.

Let M_{2v2} denote the target task’s MDP, and X_{2v2} be a set of (presumably generally unsuccessful) samples collected from M_{2v2} . We can generate this task $M_{2v2} = \tau(\mathcal{D}, F_{2v2})$, using the following instantiations for the degree of freedom vector (the order of parameters is the same as in Table 4.2.4):

$$F_{2v2} = [2, 2, \text{Helios}, \text{flat}, 68, 52.5, 2.7, 1, 0.3]$$

The following are specific subtasks that could be created using the methods from Section 3.4:

Shoot Task

One useful skill to learn is where a goal can be scored from. After having obtained some experience in the target task with at least a few goals, it is very likely that similar scenarios are also possible to score from. We can gradually expand this set of states that lead to a high reward termination using PROMISINGINITIALIZATIONS, where we use a Euclidean distance metric C over the agent’s relative distances and angles to other players, to measure state proximity:

$$M_{shoot} = \text{PROMISINGINITIALIZATIONS}(M_{2v2}, X_{2v2}, C, \delta, \rho)$$

A sample scenario can be seen in Figure 12d. Essentially, this task creates different configurations of players near the goal, and drills shooting. In our experiments, we set $\delta = 3$ and $\rho = 0.10$.

Dribble Task

Initially while exploring, the agent takes many shots on goal from far away, which are unlikely to score. A skill the agent needs is the ability to move the ball up the field, maintaining possession away from defenders, until the agent reaches a state that it can score from. This can be accomplished by chaining `ACTIONSIMPLIFICATION` with M_{shoot} using `LINKSUBTASK`:

$$\begin{aligned} M_1 &= \text{LINKSUBTASK}(M_{2v2}, M_{shoot}, V_{shoot}) \\ M_{dribble} &= \text{ACTIONSIMPLIFICATION}(M_1, X_{2v2}, \alpha) \end{aligned}$$

`LINKSUBTASK` creates a subtask M_1 where the goal is to reach situations that the agent is likely to score from, as learned in M_{shoot} . `ACTIONSIMPLIFICATION` prevents the agent from taking shots on goal from far away, since these actions usually lead to defense captures, and adds this restriction to M_1 . An example of the initial configuration for the dribble task is shown in Figure 12c. In our experiments, we set $\alpha = 100$.

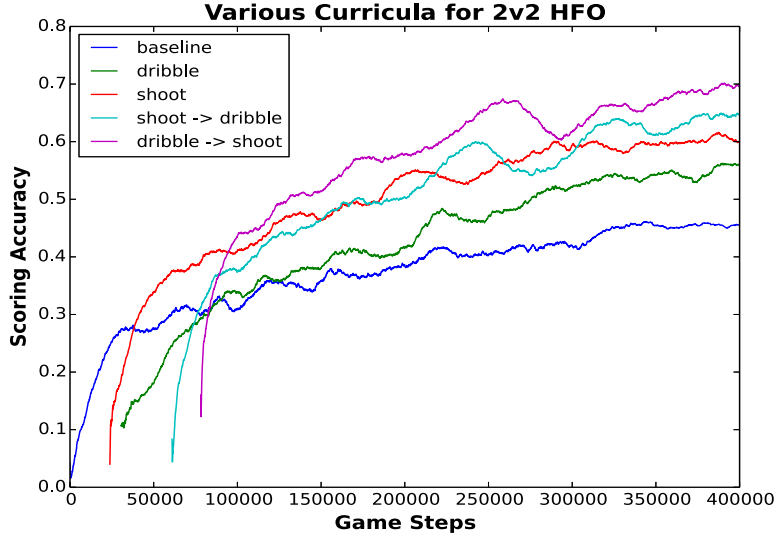


Figure 15: Goal scoring accuracy on 2v2 HFO for agents following different curricula. Standard error (not shown to avoid clutter) ranged from 0.015 to 0.027 over the last 200 episodes for all curves.

2v2 Curriculum Results

Figure 15 shows the performance on the target task of 2v2 HFO for learners following various curricula composed of the 2 tasks above. For each curriculum, we trained on sub tasks until convergence. Offsets in the curves represent time spent training in source tasks. Labels indicate the curricula used; baseline is learning on the target task without transfer.

The teams of agents were evaluated on their goal scoring ability: the fraction of times they are able to score a goal. Since each episode results in binary goal or no goal scored result, we used a sliding window of 200 episodes around each point to determine the average goal-scoring rate at each time step. All results are averaged over 25 trials. From Figure 15, it is clear to see that using a sequence of tasks to guide training significantly improves the final performance.

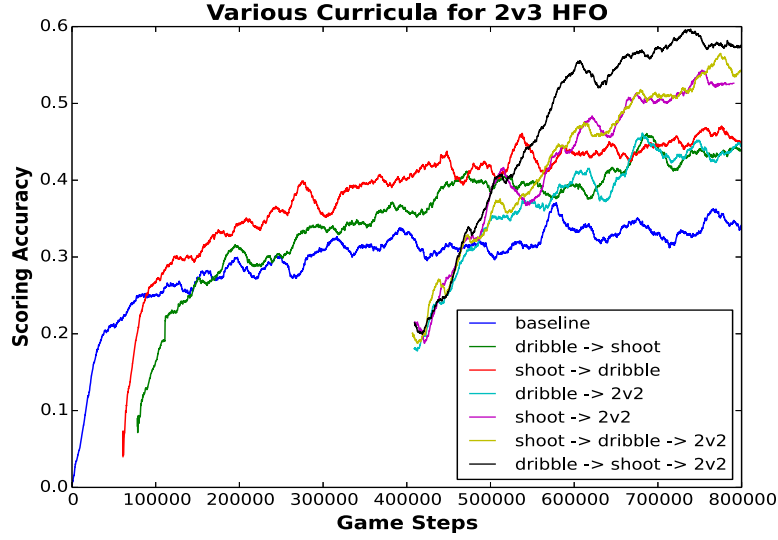


Figure 16: Goal scoring accuracy on 2v3 HFO for agents following different curricula. Standard error (not shown to avoid clutter) ranged from 0.010 to 0.039 over the last 200 episodes for all curves.

4.2.6 Extension to 2v3 HFO

In this section, we extend the problem to the harder task of 2v3 half field offense, where there are now 2 defenders and a goalie. 2v3 is fundamentally harder than 2v2, since the additional defender means both attackers can now be marked. We can generate this target task $M_{2v3} = \tau(\mathcal{D}, F_{2v3})$ using the following degree of freedom vector:

$$F_{2v3} = [2, 3, \text{Helios}, \text{flat}, 68, 52.5, 2.7, 1, 0.3]$$

This time, we can use TASKSIMPLIFICATION to simplify the degree of freedom vector to recreate the 2v2 task from the last section, allowing us to use it as a source for 2v3:

$$M_{2v2} = \text{TASKSIMPLIFICATION}(M_{2v3}, X_{2v3}, D, F_{2v3}, \tau)$$

Doing this also allows us to utilize the dribble and shoot tasks, since they are derived from M_{2v2} . Thus, we now consider 3 possible source tasks for a curriculum: $M_{dribble}$, M_{shoot} , and M_{2v2} . Results of various curricula composed of these source tasks can be seen in Figure 16.

Again, using a multistage sequence of tasks provides better asymptotic performance than a curriculum composed of a subset of its source tasks. Interestingly, we also find that the most effective curriculum in 2v2 HFO is a subset of the best curriculum in 2v3 HFO when considering this space of tasks. This observation suggests that an automated procedure to create curricula could be designed recursively.

4.2.7 Summary and Discussion

In this report, we introduced the problem of curriculum learning in reinforcement learning. As a step towards this goal, we presented a series of functions that utilize domain knowledge and observations of an agent’s performance to create subtasks tailored to the agent. We showed how

Table 7: Summary of percentage of participants for different command selections in the gradually simple condition

#	Selected Command	Gradually Simple Con	
		Initial Cur	Final Cur
1	move to the yellow room	58%	55%
2	move to the yellow/red/blue/green/purple room	85%	85%
3	move the bag to the red/blue/green/purple room	43%	63%
4	move the bag/basket/backpack/chair to ... room	75%	90%
5	# 1 + # 3	20%	33%
6	# 2 + # 4	60%	75%

Table 8: Summary of percentage of participants for different command selections in the gradually complex condition

#	Selected Command	Gradually Complex Con	
		Initial Cur	Final Cur
1	move to the yellow room	78%	75%
2	move to the yellow/red/blue/green/purple room	95%	90%
3	move the bag to the red/blue/green/purple room	35%	55%
4	move the bag/basket/backpack/chair to ... room	65%	85%
5	# 1 + # 3	23%	40%
6	# 2 + # 4	60%	75%

these subtasks could be used as components of a multistage curriculum to significantly improve an agent’s performance in two challenging multiagent reinforcement learning domains. A challenging next step in this research agenda is to develop automated methods for selecting from among the space of subtasks that our functions generate, in order to create a fully automated, individualized RL curriculum.

4.3 Curriculum Construction through Crowd-sourcing

This section summarizes the results of our user-study, which was run on Amazon Mechanical Turk (AMT). We consider data from 80 unique workers, after excluding 17 responses which we identified as users who simply pushed through the AMT task as fast as possible to be paid. We identified such users as those whose completion time was shorter than 5 minutes (the average completion time was 15 minutes 43 seconds, with a standard deviation of 8.8 minutes) or if both designed curricula contained only a single task. There were 40 participants for each of the experimental conditions (gradually complex and gradually simple).

4.3.1 Participant Performance

Recall that participants were told that their goal was to design a curriculum the dog would train on such that the dog could successfully complete the novel command “move the bag to the yellow room” in the target environment (Figure 2) with little explicit feedback. Therefore, we first examined whether users could successfully identify the need to communicate color and object concepts separately in their curriculum in two experimental conditions. We measured this by analyzing the

percentage of users who included both the command regarding moving to any colored room and the command contained any move-able object. Results in Tables 7 and 8 (the last row) show that in the gradually complex condition, 60% of users captured the idea of teaching the agent both color and object references separately in their initial curriculum, and this number increased to 75% in their final curriculum. The gradually simple condition showed exactly the same results.

Then, we were interested in studying whether users could figure out to teach the agent two more specific concepts separately—the yellow room (the room the agent needs to move to in the target task) and the bag object (the object the agent needs to move in the target task). We evaluated this by computing the percentage of users who combined the command “move to the yellow room” and the command “move the bag to the red/blue/green/purple room” in their curriculum. Surprisingly, in the gradually complex condition, only 23% of users introduced the yellow room and bag concept to the agent in their initial curriculum, and 17% more users captured this idea in their final curriculum. The gradually simple condition produced similar results. However, there is still some evidence showing that more users tended to teach the agent these two specific concepts the agent needed to learn in the target task. Specifically, in the gradually complex condition, we find that 1) 78% of users tried to train the agent to move to the yellow room, and 2) a total of 65% of participants wanted to teach the agent to move an object (bag/basket/backpack/chair) to some colored room, where 53.8% of them focused on teaching it to move the bag.

In both the initial and final curricula, a chi-squared test shows that the number of users who selected each type of commands in Tables 7 and 8 was not significantly different ($p > 0.05$) between the two experimental conditions, suggesting that the ordering of source environments does not affect human performance in identifying the concepts the agent needs to learn to complete the target task.

4.3.2 Concept Introduction

We hypothesized that users would gradually introduce more complex environments or commands to the agent in their curriculum. To validate this, we analyzed the changes in the environment and command complexity. We found that in the gradually complex condition, only 37.5% (or 45%) of users consistently increased environment complexity in their initial (or final) curriculum. However, a total of 50% (or 60%) of users selected the simple command regarding moving to some colored room first, and then consistently chose more complex object-moving commands in their initial (or final) curriculum. The gradually simple condition showed similar results. This suggests that users preferred to consistently introduce more complex commands rather than environments to the agent in each curriculum. A chi-squared test shows that the number of users who consistently introduced more complex environments or commands was not significantly different ($p > 0.05$) between two experimental conditions.

There is another interesting finding that users tended to introduce more complex commands to the agent across different curricula in both experimental conditions. In particular, in the gradually complex condition, for the 37 users who kept or increased the curriculum length, 54% of them only replaced the command regarding moving to some colored room with more complex object-moving command, or added new object-moving commands in the final curriculum. In the gradually simple condition, 62% of the 34 users who kept or increased the curriculum length only introduced more complex object-moving commands to the agent in their final curriculum. Therefore, as we expected, both within a curriculum and between curricula, users tended to gradually introduce more complex commands to the agent rather than more complex environments.

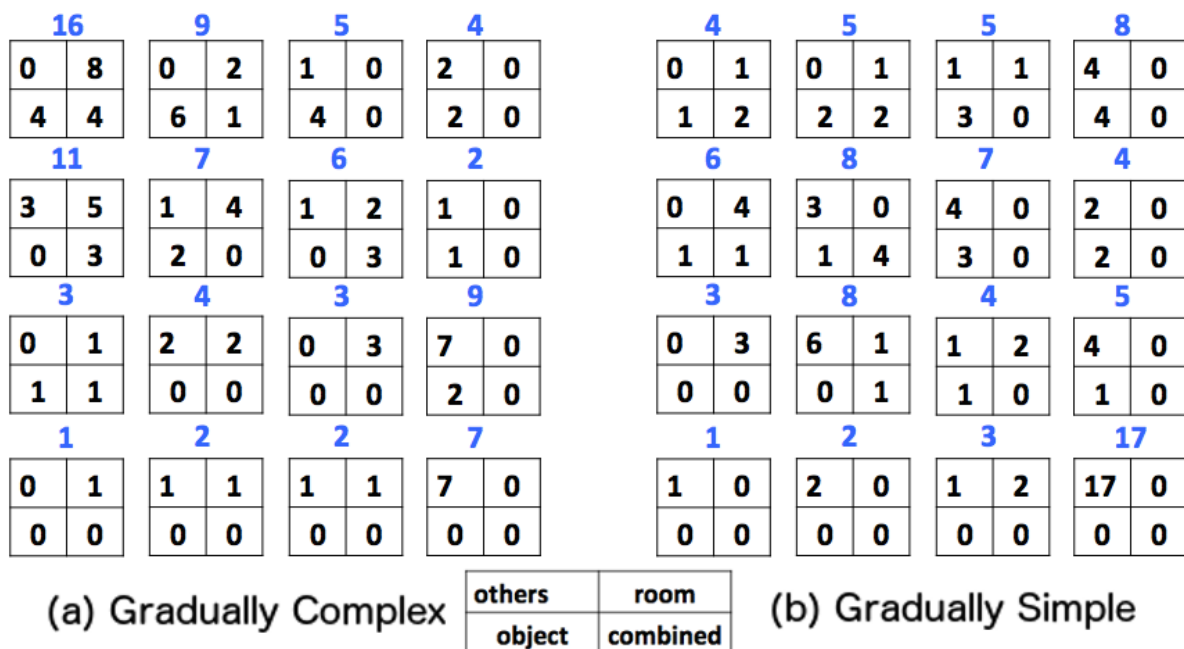


Figure 17: The number of times each of four transitions being followed for each environment in the initial curricula in the two experimental conditions. There are 16 corresponding squares for 16 environments. The blue number represents the total number of times all four transitions being followed in each environment.

4.3.3 Transition Dynamics

Although previous results show that less than half of users consistently increased the environment complexity in their curriculum, we observed that a considerable number of users implemented this in segments. It suggests that most users considered increasing the environment complexity when designing curricula. A better understanding of how users select more complex environments might give us insights into the active selection of better curricula.

We hypothesized that different users would choose to increase the environment complexity in different ways, and it might be affected by the ordering of source environments. In particular, for the 4×4 grid (shown in Figure 3), we defined four different ways for users to increase the environment complexity: room transition, object transition, combined transition, and others. For a given task M_i in a curriculum, a transition to M_{i+1} is a *room transition* if and only if the number of rooms increases between M_i and M_{i+1} . If the number of objects increases, it is an *object transition*, and if they both increase it is a *combined transition*. All other cases are considered as *other transitions*. We aim to study the most popular transition followed by users in two experimental conditions by computing the frequency of each of four transitions being followed for each environment.

Figure 17 summarizes the number of times each transition type (room, object, combined, and other) was used from each environment in the initial curricula in the two experimental conditions. We observe that the room transition was the most-frequently used in the gradually complex condition, while the object transition was the most-frequently used in the gradually simple condition. A chi-squared test shows that the differences of the total number of times each transition type being followed when users design initial curriculum between two experimental conditions was statistically significant ($p \ll 0.01$), verifying that the ordering of source environments does affect the

way humans use to increase the environment complexity.

4.3.4 Environment Preference

We hypothesized that some source environments in the grid would be preferred by users when designing their curricula. Analyzing the properties of these environments might enrich the general principles regarding efficient curricula and inspire the development of new machine learning algorithms that accommodate human teaching strategies. Therefore, we explored user preference in each environment by computing the ratio of the number of users who selected corresponding environment at least once to the total number of users.

Figure 18 summarizes user preference in each of the 16 environments when designing an initial or final curricula in two experimental conditions. A larger dot represents a higher probability of the corresponding environment being chosen. We find that when designing initial curriculum, users were more likely to select 1) Environments 1, 2, 5, and 16 in the gradually complex condition, and 2) Environments 5, 6, 12, and 16 in the gradually simple condition. This finding implies that users preferred to choose 1) the simplest environments that only contain one important concept (Environments 1 and 2 are the two simplest ones that refer to a yellow room, and Environment 5 and 6 are the two simplest ones that include an object) that the agent needed to learn for the target task, and 2) more complex environments that are more similar to the target environment (Environment 12 and 16 are two of the most similar ones to the target environment).

Compared to the initial curricula, Figure 18 shows that most environments had a higher probability of being included in the final curricula in the two experimental conditions, due to the fact that most users tended to increase the curriculum length. In particular, Environments 7, 11, and 12, and 3, 10, 12, and 15 gained the most probability in the gradually complex and gradually simple conditions, respectively. As discussed before, users tended to focus on teaching the agent more complex object-moving tasks (building on previous tasks) when redesigning curricula, and most of these environments provide a good chance for the agent to learn object reference with a relatively large number of different colored rooms.

We also note that users had a lower probability of choosing the two simplest environments (1 and 2) after varying the order of the 16 environments. Fisher’s exact test shows that the frequency of each of the 16 environments being selected by users into initial or final curricula was not significantly different ($p > 0.05$) between the two experimental conditions, suggesting that the ordering of source environments does not influence participants’ preference in choosing environments. We believe that knowing users prefer 1) isolating complexity, 2) selecting simplest environments they can to introduce one complexity at a time, 3) choosing environments that are most similar to the target environment, and 4) introducing complexity building on previous tasks rather than backtracking to introduce a new type of complexity can be highly useful for the design of new machine learning algorithms which accommodate human teaching strategies.

5 Conclusion

In this project, we introduced the problem of curriculum learning in reinforcement learning. The problem consists of designing a *sequence* of source tasks for an agent to train on, such that final performance or learning speed on a difficult target task is improved.

We approached the problem along three main lines. First, we looked at whether an agent can learn to predict the benefit of transferring a policy from one task to another such that it can

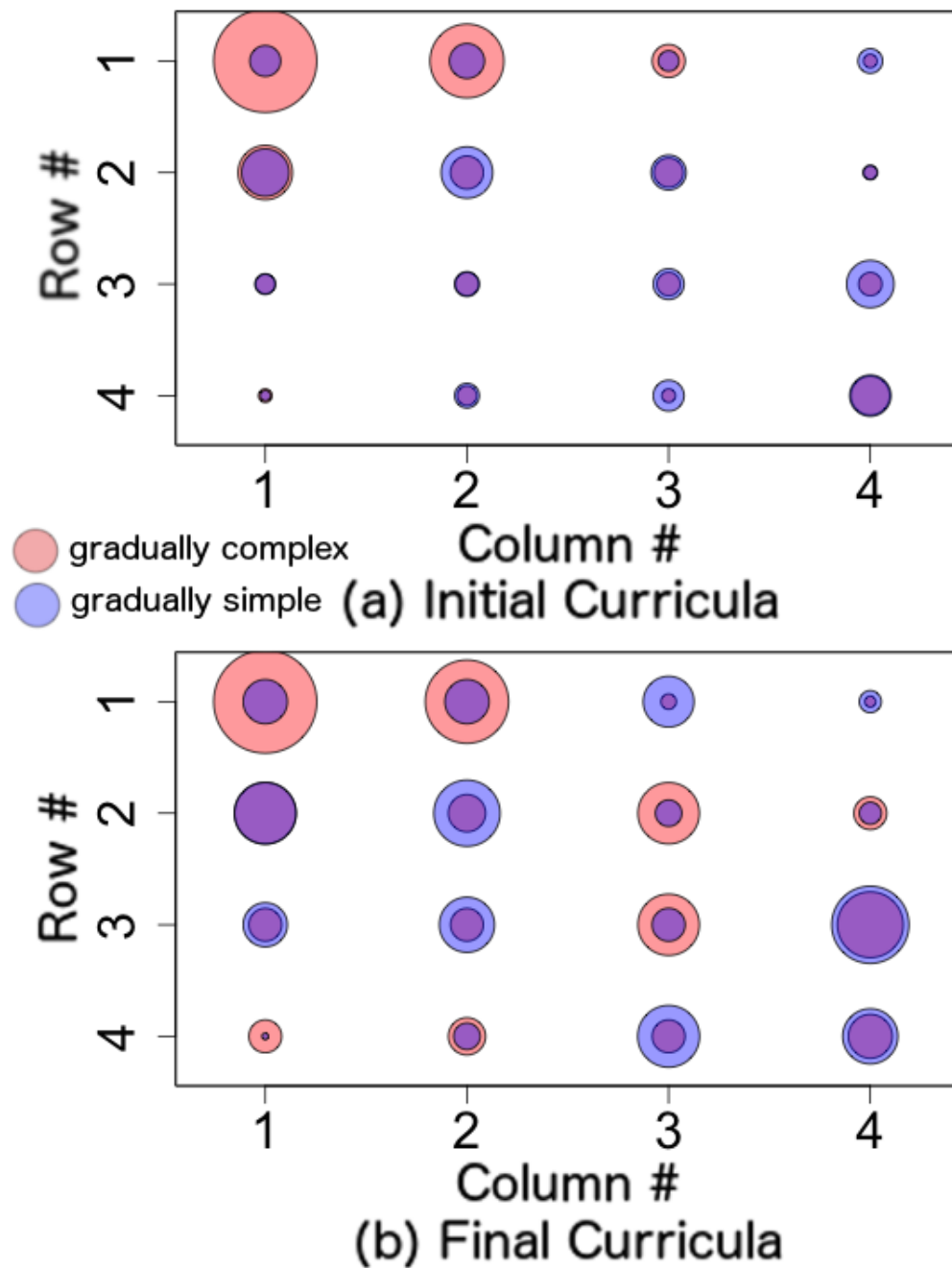


Figure 18: The probability of each environment being included in the initial or final curricula in the two experimental conditions. The purple circle represents the overlap of probability.

appropriately select a good source task for a given target task. To solve that problem, we proposed a framework for source task selection in settings where neither samples from the target task, nor a model of the task, are available to the learning agent. Instead, the agent used task descriptors (i.e., a low-dimensional feature vector describing some aspects of the task) to learn the expected benefit of transfer, i.e., transferability, between source tasks and target tasks. The framework was evaluated using a large-scale experiment in which the agent learned to play 192 variations of the Ms. Pac-Man game.

To test our framework, the agent played over 170 million games, making this, to the best of our knowledge, the largest computational experiment in transfer learning conducted so far. Our results show that an agent can indeed learn to predict the transferability for an arbitrary pair of source-target tasks, provided training pairs for which the benefit (or detriment) of transfer is known. The learned transferability model was then used to effectively select relevant source tasks that improve the agent's learning performance on a given target task.

Next, we presented a series of functions that utilize domain knowledge and observations of an agent's performance to create subtasks tailored to the agent. We showed how these subtasks could be used as components of a multistage curriculum to significantly improve an agent's performance in two challenging multiagent reinforcement learning domains. A challenging next step in this research agenda is to develop automated methods for selecting from among the space of subtasks that our functions generate, in order to create a fully automated, individualized RL curriculum.

Finally, we explored the possibility of using non-expert humans to create a curriculum for a learning agent. We presented an empirical study designed to explicitly explore how non-expert humans design curricula for an agent to train on, allowing the agent to complete a target task with little explicit feedback. Our most important finding was that users followed some salient patterns when selecting and sequencing environments in the curricula, which we plan to leverage in the design RL algorithms in the future. Our goal will be to develop inductive biases in learning algorithms that can benefit from the types of tasks and transitions non-expert human teachers use more frequently. Future work will 1) allow users to create a sequence of novel source tasks for the agent to train on, 2) come up with a stable way to show the score of the designed curricula to motivate users to design better ones, and 3) implement an RL algorithm that can leverage all interesting salient patterns followed by non-expert humans to design better curricula.

References

- [1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.
- [2] Hidehisa Akiyama and Tomoharu Nakashima. Helios2012: Robocup 2012 soccer simulation 2d league champion. In *RoboCup 2012: Robot Soccer World Cup XVI*, pages 13–19. Springer, 2013.
- [3] Haitham B Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1206–1214, 2014.
- [4] Haitham B Ammar, Karl Tuyls, Matthew E Taylor, Kurt Driessens, and Gerhard Weiss. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 383–390. International Foundation for Autonomous Agents and Multiagent Systems, 2012.
- [5] Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. An automated measure of MDP similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.
- [6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.
- [7] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.
- [8] Peter Burrow and Simon M Lucas. Evolution versus temporal difference learning for learning to play Ms. Pac-Man. In *IEEE Symposium on Computational Intelligence and Games*, pages 53–60. IEEE, 2009.
- [9] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.
- [10] Kyriakos C Chatzidimitriou, Ioannis Partalas, Pericles A Mitkas, and Ioannis Vlahavas. Transferring evolved reservoir features in reinforcement learning tasks. In *Recent Advances in Reinforcement Learning*, pages 213–224. Springer, 2012.
- [11] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 1996.
- [12] Eric Eaton, Marie desJardins, and Terran Lane. Modeling transfer relationships between learning tasks for improved inductive transfer. In *Proceedings of the European Conference on Machine Learning*, pages 317–332. Springer-Verlag, 2008.
- [13] Anestis Fachantidis, Ioannis Partalas, Grigorios Tsoumakas, and Ioannis Vlahavas. Transferring task models in reinforcement learning agents. *Neurocomputing*, 107:23–32, 2013.

- [14] Fernando Fernández, Javier García, and Manuela Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010.
- [15] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727. ACM, 2006.
- [16] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.
- [17] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2625–2633, 2013.
- [18] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.
- [19] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*, pages 72–85. Springer Verlag, Berlin, 2007.
- [20] W. Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *The Fifth International Conference on Knowledge Capture*, September 2009.
- [21] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems*, 2009.
- [22] M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2010.
- [23] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In Andrew McCallum and Sam Roweis, editors, *Proceedings of the Twenty-Fifth Annual International Conference on Machine Learning (ICML-2008)*, pages 544–551, Helsinki, Finland, July 2008.
- [24] Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.
- [25] Alessandro Lazaric and Marcello Restelli. Transfer from multiple MDPs. In *Advances in Neural Information Processing Systems*, pages 1746–1754, 2011.
- [26] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551. ACM, 2008.

- [27] Yong Jae Lee and Kristen Grauman. Learning the easy things first: Self-paced visual category discovery. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1721–1728. IEEE, 2011.
- [28] Robert Loftin, Bei Peng, James MacGlashan, Michael L Littman, Matthew E Taylor, Jeff Huang, and David L Roberts. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. *Journal of Autonomous Agents and Multi-Agent Systems*, pages 1–30, 2015.
- [29] Ramon Lopez De Mantaras, David McSherry, Derek Bridge, David Leake, Barry Smyth, Susan Craw, Boi Faltings, Mary Lou Maher, Michael T Cox, Kenneth Forbus, et al. Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, 20(03):215–240, 2005.
- [30] J. MacGlashan, M. L. Littman, R. Loftin, B. Peng, D. L. Roberts, and M. E. Taylor. Training an agent to ground commands with reward and punishment. In *Proceedings of the AAAI Machine Learning for Interactive Systems Workshop*, 2014.
- [31] S. Mannor, I. Menache, A. Hoze, , and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 560–567, 2004.
- [32] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368, 2001.
- [33] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *13th European Conference on Machine Learning*, pages 295–306. Springer, 2002.
- [34] Lilyana Mihalkova and Raymond J Mooney. Using active relocation to aid reinforcement learning. In *FLAIRS Conference*, pages 580–585, 2006.
- [35] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, Singapore, May 2016.
- [36] Trung Nguyen, Tomi Silander, and Tze Y Leong. Transferring expectations in model-based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2555–2563, 2012.
- [37] Theodore J Perkins and Doina Precup. Using options for knowledge transfer in reinforcement learning. In *Technical Report*. University of Massachusetts, 1999.
- [38] Ross J. Quinlan. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, Singapore, 1992. World Scientific.
- [39] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *Machine Learning: ECML 2007*, pages 699–707. Springer, 2007.

- [40] David Robles and Simon M Lucas. A simple tree search method for playing Ms. Pac-Man. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 249–255. IEEE, 2009.
- [41] Paul Ruvolo and Eric Eaton. Active task selection for lifelong machine learning. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI-13)*, July 2013.
- [42] Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, June 2013.
- [43] O. Simsek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758, 2004.
- [44] Jivko Sinapov, Sanmit Narvekar, Matteo Leonetti, and Peter Stone. Learning inter-task transferability in the absence of target task samples. In *Proceedings of the 2015 ACM Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. ACM, 2015.
- [45] Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.
- [46] Burrhus F Skinner. Reinforcement today. *American Psychologist*, 13(3):94, 1958.
- [47] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [48] Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.
- [49] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.
- [50] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.
- [51] Richard S Sutton, Anna Koop, and David Silver. On the role of tracking in stationary environments. In *Proceedings of the 24th international conference on Machine learning*, pages 871–878. ACM, 2007.
- [52] István Szita and András Lörincz. Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man. *J. Artif. Intell. Res.(JAIR)*, 30:659–684, 2007.
- [53] Matthew E. Taylor. Assisting Transfer-Enabled Machine Learning Algorithms: Leveraging Human Knowledge for Curriculum Design. In *The AAAI 2009 Spring Symposium on Agents that Learn from Human Teachers*, March 2009.
- [54] Matthew E. Taylor, Nicholas Carboni, Anestis Fachantidis, Ioannis Vlahavas, and Lisa Torrey. Reinforcement learning agents providing advice in complex video games. *Connection Science*, 26(1):45–63, 2014.

- [55] Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer, 2008.
- [56] Matthew E Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 283–290. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
- [57] Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005. ACM Press.
- [58] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.
- [59] Matthew E Taylor, Peter Stone, and Yaxin Liu. Value functions for rl-based behavior transfer: A comparative study. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 880. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.
- [60] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer Learning via Inter-Task Mappings for Temporal Difference Learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.
- [61] Matthew E Taylor, Shimon Whiteson, and Peter Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 37. ACM, 2007.
- [62] Andrea Lockerd Thomaz and Cynthia Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *AAAI*, volume 6, pages 1000–1005, 2006.
- [63] Sebastian Thrun. Is learning the n-th thing any easier than learning the first. In *Advances in Neural Information Processing Systems*, volume 8, pages 640–646, 1996.
- [64] Christopher M Vigorito and Andrew G Barto. Intrinsically motivated hierarchical skill learning in structured environments. *Autonomous Mental Development, IEEE Transactions on*, 2(2):132–143, 2010.
- [65] L. S. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 1978.
- [66] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning*. Springer, 1997.
- [67] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

- [68] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022. ACM, 2007.
- [69] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufman, San Francisco, 2nd edition, 2005.

List of Symbols, Abbreviations, and Acronyms

AMT: Amazon Mechanical Turk.

CBR: Case based reasoning. A framework in which new tasks are solved by re-using solutions from previously solved similar tasks.

DCG: Discounted Cumulative Gain. A measure used to evaluate the quality of an estimated ranking with respect to the ground truth ranking.

HFO: Half-field Offense, a domain based on the 2D Robocup Simulator.

LR: Linear Regression. A method for modeling the relationship between a dependent variable and one or more explanatory variables using a linear function.

MDP: Markov Decision Process. A mathematical framework for modeling decision making under uncertainty.

RL: Reinforcement Learning. A class of algorithms concerned with how agents can take actions in an environment as to maximize some notion of reward.

TL: Transfer Learning. A methodology in which training on a source task is leveraged to speed up or otherwise improve learning on a target task.