12-11-2008

A survey of transfer learning methods for reinforcement learning

Nicholas Bone
Western Washington University

Cover Page

I declare that the attached assignment is wholly my own work, citations have been correctly placed, and that no part of it has been:

1. copied from any work produced by other person(s)
2. provided by other student(s)
3. taken from other person(s) program
4. copied from any other source including web sites

Last Name (PRINTED):	BONE
First Name (PRINTED):	NICHOLAS
Student Number:	
Course:	CS 514
Section:	N/A
Assignment:	final survey paper
Date:	11 December 2008
Signature:	

A Survey of Transfer Learning Methods for Reinforcement Learning

Nicholas Bone

bonen@cc.wwu.edu

Department of Computer Science, Western Washington University

December 2008

Abstract

Transfer Learning (TL) is the branch of Machine Learning concerned with improving performance on a target task by leveraging knowledge from a related (and usually already learned) source task. TL is potentially applicable to any learning task, but in this survey we consider TL in a Reinforcement Learning (RL) context. TL is inspired by psychology; humans constantly apply previous knowledge to new tasks, but such transfer has traditionally been very difficult for—or ignored by—machine learning applications. The goals of TL are to facilitate faster and better learning of new tasks by applying past experience where appropriate, and to enable autonomous continual learning agents. TL is a young field (within the RL context), and has only recently seen much interest from the RL community. However, it is rapidly growing, and in the last few years dozens of new methods have been proposed. In all cases surveyed, the proposed methods have been remarkably successful towards the goals of TL. This survey presents a novel classification of current TL methods, a comparative analysis of their strengths and weaknesses, and reasoned ideas for future research.

1 Introduction

A long-term goal of Machine Learning research is to create intelligent learning agents that can autonomously adapt to changing environmental pressures in complex domains in order to maintain effective behavior throughout their operating lives. More generally, the field of Machine Learning is concerned with the creation of artificial systems that can learn to improve their performance at a given task through experience. Thus, any machine learning problem is defined by the task to be solved, the performance measure by which to evaluate behavior, and what training experience is to be presented to the learning agent [Mitchell, 1997].

Most Machine Learning applications are focused on individual, static, and relatively simple learning tasks. The typical paradigm is to first define the task to be solved, then determine how to represent this task to the learning agent and decide which learning algorithm to use. When the task is sufficiently mastered such that performance reaches some target threshold, learning stops and the system is deployed. This paradigm treats each task as a separate problem; the human designers may reuse methods and domain knowledge, but the learning agent always starts from scratch. Humans, by contrast, exhibit what is sometimes called "lifelong learning" [Thrun and Mitchell, 1995], meaning that we don't stop learning when we're just "good enough", but can continue getting better at many different tasks throughout our lives. When faced with a new task, we regularly apply skills and knowledge that we have learned through past experience on a multitude of other tasks. To achieve the long-term goal above we must endow computers with similar capacity; how to do this is the focus of this survey.

Machine Learning is now sufficiently mature that a great many single-task applications (and even more complex applications composed of multiple independent sub-tasks) can be efficiently and effectively solved with any number of well-known and proven machine learning techniques. However, there has been relatively little research on solving broader, more complex, dynamic, and ongoing tasks, which we must do in order to achieve the high goal of autonomous and adaptive agents. There are three main reasons for this: (1) most Machine Learning applications do not require general-purpose adaptive solutions, but are actually fairly simple; (2) until recently, Machine Learning had not sufficiently proved itself on enough small tasks to be able to tackle larger ones; and (3) the general problem of creating adaptive agents that can be effective at diverse tasks is really hard.

Transfer Learning (TL) is a branch of Machine Learning that is attempting to help solve this problem by enabling *knowledge transfer* between related tasks. The main idea is that we don't want to have to start over every time we face a new task, but construct a learning agent that can leverage its past experience from previous tasks (or import distilled knowledge from other agents) in order to improve its performance on a new task. Such knowledge transfer may be useful in any learning scenario, however in this survey we consider Transfer Learning only within the context of Reinforcement Learning (RL)[1]. (See the next section for a brief background on RL.)

The main objectives of Transfer Learning are:

- To make hard tasks more tractable.
- To facilitate faster learning of new tasks.
- To reduce the sample complexity of new tasks.
- To enable agents to autonomously choose when to reuse what knowledge, and how.
- To enable autonomous, continuous, and progressive learning, with less dependence on human guidance.

The idea of the first objective is that complex tasks have really large search spaces such that initial random exploration (starting with no task knowledge) may take prohibitively long; however, given a good starting point for the search (in terms of a transferred policy) we may be able to efficiently find an effective solution. The next two objectives are closely related: by leveraging transfer we may be able to reduce training time in the target task, requiring fewer experiences in the target environment (which may be expensive to obtain, for instance in the case of a physical agent learning online) possibly at the expense of increased computation (which is usually relatively inexpensive). In some cases it is even faster to invest time "practicing" on a smaller version of the target task, and then transferring that knowledge, rather than learning from scratch on the target task. The fourth objective attempts to shift even more responsibility to

[1] For a survey of TL applied to classification, regression, and clustering problems see [Pan and Yang, 2008].

the learning agent, in the hopes of expanding the applicability of transfer to cases in which there may be a great many potential source tasks of uncertain relevance. The fifth objective addresses the continual learning case, where we are not just concerned with isolated tasks, but with creating long-lived adaptive agents.

Putting all of these together, the ultimate (and still somewhat far-fetched) goal is to create a learning system such that when faced with a new task, we the human designers need only simply describe the task to the learning agent and point it in the right direction, and it will learn the task quickly and effectively. No human will need to stop the experiment, tweak some parameters, and restart—the learning agent will be capable of performing all necessary adjustments to its own processes. And in some cases it may not be necessary to even define distinct tasks, but just provide an environment in which effective behavior is rewarded, and if the dynamics of the environment shift with the needs of the day then the agent will smoothly adapt.

The purpose of this survey is threefold:

1. To examine current transfer learning methods and how they are used;
2. To characterize the current state of the field based on a comparative analysis of the methods;
3. To identify specific problems that must be addressed in order to advance the field, along with ideas for future research.

The remainder of this survey is organized as follows. Section 2 presents a brief overview of Reinforcement Learning and defines some key terms; it may safely be skipped by readers who are familiar with RL methodology. Section 3 presents a novel categorization of Transfer Learning methods along with detailed explanations of how their authors used them. Section 4 presents a comparative analysis of the surveyed methods to draw conclusions about the current state of the field and to identify areas for future research. Section 5 concludes.

2 Reinforcement Learning Background

I should probably say something about ML in general and RL in particular [Sutton and Barto, 1998]. Describe the Agent-Environment interaction (state, action, reward). Mention Markov Decision Processes. Briefly introduce base algorithms such as TD and direct policy search, and identify some FAs like ANN, CMAC, and RBF. Anything else?

3 Current Transfer Learning Methods

The study of Transfer Learning is not new. Psychologists have long been aware of transfer in human learning, at least back to [Thorndike and Woodworth, 1901]. Even within Machine Learning, TL has been studied in classification and clustering problems for decades (often called *inductive transfer* in that arena, e.g., [Caruana, 1997]). However, only recently has there been much research in applying TL to reinforcement learning problems. Taylor notes this in his dissertation [2008], pointing out that:

> The 2005 NIPS workshop, "Inductive Transfer: 10 Years Later," [Silver et al., 2005] had few RL-related transfer papers, the 2006 ICML workshop, "Structural Knowledge Transfer for Machine Learning," [Banerjee et al., 2006] had many, and the 2008 AAAI workshop, "Transfer Learning for Complex Tasks," [Taylor et al., 2008a] focuses on RL. (2)

Thus, the branch of TL that focuses on RL problems is still immature. Although a great many methods have recently been proposed, there have not yet been enough practical comparisons or theoretical analyses on which to build a cohesive framework for future applications—we will discuss this further in Section 4.

The methods that have been proposed vary along many dimensions, including the RL methods on which they are based, the restrictions they place on the source and target tasks (especially how similar they must be), and on how they enact transfer. In this section we classify the methods based on *what* they transfer, in order to draw broad comparisons between methods and to highlight some areas for potential improvements.

Table 3.1 lists the methods included in this survey, grouped by what kind of knowledge they transfer. Each group is discussed further in its own subsection, as indicated in the table. After each method are some key advantages, drawbacks, and restrictions on its use, to provide the reader with a preliminary sense of how the methods compare and where they might be applicable.

TABLE 3.1 – LIST OF METHODS INCLUDED IN THIS SURVEY

Method	**Key Advantages**	**Restrictions and Drawbacks**
Methods to Learn Inter-Task Mappings (see section 3.1)		
MASTER [Taylor et al., 2008c]	- Autonomously learns a mapping from limited experience. - Supports weighted transfer from multiple source tasks.	- Requires a state-space with a natural distance metric. - Exponential in the number of state variables and actions.
MLC [Taylor et al., 2007b]	- Learns a mapping from limited experience and some human input.	- Requires a pre-specified classification of state variables (across both tasks) into semantically similar groups.
Transfer of Learned Policies (see section 3.2)		
PRQL [Fernandez and Veloso, 2006]	- Learns online which source task from a library, if any, is best for transfer.	- Requires that the tasks have identical state- and action-spaces; only the reward function may differ.
CHILD [Ring, 1997]	- Supports non-Markov environments.	- Requires that the tasks have the same state variables and action-spaces, and that the agent be the same for all tasks.

TGR [Ramon et al., 2007]	- Abstracts the state-space in first-order logic, supporting transfer between tasks with similar yet not identical state-spaces.	- Requires that the tasks have the same object-types in their state-spaces, and that the agent be the same.
TVITM-PS [Taylor et al., 2007b]	- Allows transfer between different state- and action-spaces, if given a mapping. - One of the few methods focused on direct policy search (instead of TD learning).	- Source and target agents must both use ANN-based policy search. - Risk of overfitting if transferring after too much source task training.
Transfer of Options (Learned Macro Actions) (see section 3.3)		
STASO [Konidaris and Barto, 2007]	- Learns options in agent-space, supporting transfer between related tasks with different state-spaces.	- Requires a pre-defined set of skills to learn. - Agent must be the same across tasks.
SMILe [Bonarini et al., 2006]	- Agent autonomously identifies and learns its own options. - Agent may use any base RL algorithm.	- All tasks must share same state-space, transition dynamics, and primitive actions; only the reward function may differ.
RMT-D [Torrey et al., 2007]	- Allows transfer between different state- and action-spaces, if given a mapping. - Agent autonomously identifies and learns its own options.	- Source and target agents must both use TD learning.
Transfer of Rules (Advice) (see section 3.4)		
Progressive RL [Madden and Howley, 2004]	- Allows free and independent choice of RL algorithm and symbolic learning algorithm.	- Source and target tasks must have same state variables and action space.
AI^2 [Torrey et al., 2006]	- Allows transfer between different state- and action-spaces, if given a mapping. - Can directly transfer human-supplied advice without requiring any source task.	- Source and target agents must both use TD learning.
Rule Transfer [Taylor and Stone, 2007a]	- Allows transfer between different state- and action-spaces, if given a mapping. - Supports three ways to transfer the advice.	- Target agent must use TD learning. - Effectiveness depends on some transfer parameters which cannot be optimized *a priori*.
Transfer of Action Values for TD Learning (see section 3.5)		
MLN Transfer [Torrey et al., 2008a]	- Allows transfer between different state- and action-spaces, if given a mapping.	- Source and target agents must both use TD learning. - Relatively high computational complexity.
Autonomous Shaping [Konidaris and Barto, 2006]	- Learns action values in agent-space, supporting transfer between related tasks with different state-spaces.	- Tasks must be "reward-linked" and share an agent-space (but problem-space may differ). - Agent must be the same across tasks, but may use any TD learning algorithm.
Q-Value Reuse [Taylor et al., 2007a]	- Allows transfer between different state- and action-spaces, if given a mapping.	- Source and target agents must both use TD learning.
Transfer of Samples (Recorded Experience) (see section 3.6)		
TSBRL [Lazaric et al., 2008]	- Source agent may use any RL algorithm; target agent may use any Batch RL algorithm.	- Tasks must have identical state and action spaces; only the transition and reward functions may differ.
ORT [Taylor and Stone, 2007b]	- Allows quick “upgrades” to a different agent representation with little or no loss of performance.	- Source and target agents must both use TD learning.
TIMBREL [Taylor et al., 2008b]	- Allows transfer between different state- and action-spaces, if given a mapping.	- Target agent must use instance-based RL (though source agent may use any method).
Transfer of Weights or Functions (see section 3.7)		

GPCRC [Jaśkowski et al., 2008]	- Allows transfer between different state- and action-spaces, if given a mapping. - Supports multi-task learning and transfer from multiple source tasks.	- Both source and target agents must use GP. - Significant risk of negative transfer.
Complexification [Taylor and Stone, 2007b]	- Allows quick "upgrades" to a more complex agent representation with little or no loss of performance.	- Does not transfer between tasks, but between internal function approximator representations. - Most applicable when the FA exhibits locality.
TVITM-VF [Taylor et al., 2007a]	- Allows transfer between different state- and action-spaces, if given a mapping.	- Source and target agents must both use TD learning and the same kind of function approximator.

The remainder of this section is divided into seven subsections, one for each grouping in Table 3.1. Each subsection explains the common ideas shared by all methods therein, provides a high-level overview of that type of transfer, and then describes each method in greater detail, including how the authors applied it and what results they achieved. In each section, the first method is exposed in greater detail than the subsequent methods. The purpose of this is to provide the reader with more concrete examples without being too repetitive by exhaustively examining every method. The choice of methods to "feature" was largely arbitrary, and should not be construed as a recommendation of any method over another.

The very next subsection, on inter-task mappings, is structured slightly differently from those that follow. It first explains what inter-task mappings are and why they are useful, and then presents two methods to automatically learn such mappings.

3.1 Transfer via Inter-Task Mappings

An underlying assumption of Transfer Learning is that there is some already experienced source task that is somehow related to the current target task. But simply knowing *that* two tasks relate is not enough (would you be a better chef if someone told you that cooking is like riding a bike?); we must also know *how* they relate. For a Reinforcement Learning agent, knowing how two tasks relate means being able to estimate the worth of any action from the current state based on "remembered" experience from "similar" states in the source task. Usually, however, exact relationships cannot be deduced, so we estimate the relationship based on some notion of "similarity" and assume that similar actions chosen from similar states will result in similar rewards and similar successor states. In some cases, this "similarity" may be obvious, such as if the state variables and action-space are identical across the tasks—indeed, six of the twenty-one methods listed in Table 3.1 impose this restriction. Another three methods require that the agent's perception of the state-space be consistent across tasks. The remaining twelve methods allow both state variables and actions to differ across tasks, and they do this using *inter-task mappings*.

The concept of inter-task mappings for transfer learning was introduced by Taylor and Stone in 2005 (superseded by [Taylor et al., 2007a]) for their "behavior transfer" algorithm (later reported as TVITM-VF; see section 3.7). The basic idea is to define a pair of functions: one from the state-space of the target task into the state-space of the source task, and the other from the action-space of the target into the action-space of the source. Then, any (*state, action*) pair encountered in the target task can be mapped to a (*state, action*) pair in the source task. If we already have a trained function approximator to estimate action-values in the source task then this will give us—via the mapping—a function approximator for the target task, which the target agent can use to direct its search (this is precisely how the Q-Value Reuse algorithm works, section 3.5).

In most transfer learning methods, when an inter-task mapping is used, it is supplied beforehand by the human designers. However, as one of the goals of Transfer Learning is to enable fully autonomous learning agents, it is desirable to create agents that can learn their own mappings. In the remainder of this section we present two such methods for autonomous learning of inter-task mappings. These are not transfer learning methods *per se*, because they do not actually transfer any knowledge from one task to another, but they generate inter-task mappings to facilitate other TL methods.

Algorithm 3.1.1 lists the "Modeling Approximate State Transitions by Exploiting Regression" (MASTER) algorithm developed by Taylor et al. [2008c]. Their premise is that, in some cases, exploration in the target task is quite expensive relative to offline computation, so it may be worth a great deal of computation and the overhead of eventual transfer in order to get a performance boost in the target task. Note that steps 3 and 4 assume a reliable distance metric on the target task state-space, such that "close" states by the metric are assumed to be "near" each other in the state-transition model.

Algorithm 3.1.1: MASTER *summarized from* [Taylor et al., 2008c]

1. Collect m (*state, action, new-state*) samples from the source task, and store in set S.
2. Collect t (*state, action, new-state*) samples from the target task, and store in set T (usually t is much less than m, especially if exploration in the target environment is expensive).
3. Train a transition model M on all samples (s,a,s') in T to predict s' given s and a, by trying to minimize the mean squared error (MSE) of $|M(s,a) - s'|$. (They used a neural network function approximator, but other regression methods should work just as well.)
4. For each possible mapping, P, from the state-variables and action-space of the source task to the state-variables and action-space of the target task, calculate the average MSE of $|M(P(s),P(a)) - P(s)'|$ over all (s,a,s') in S.
5. Return whichever mapping has the least MSE, or if there are multiple good ones,

then return a weighted collection of mappings normalized by the inverse of their MSE.

They tested MASTER in conjunction with the Q-Value Reuse transfer algorithm (see section 3.5) in the "Mountain Car" domain, using a Sarsa(?) learning algorithm with CMAC function approximation. They tested transfer in three experiments: from 2D to 3D Mountain Car; from 2D Hand-Brake to 3D Hand-Brake Mountain Car; and from both 2D standard and 2D Hand-Brake to 3D Hand-Brake Mountain Car. In all experiments they compared the MASTER-generated mapping to transfer with an average-action mapping (like MASTER but ignoring action differences) and to learning without transfer. In the first experiment they also tested a hand-coded mapping and a fully average mapping (treating all states as equal). The MASTER mapping performed essentially just as well as the hand-coded mapping, and much better than learning without transfer; the fully average mapping performed much worse than no transfer; and the average-action mapping performed significantly worse than the MASTER mapping, but still much better than no transfer. The final experiment demonstrated that a weighted transfer from both source tasks (by the inverse MSE of each) performed significantly better than transferring from either one alone. In all, the agents using MASTER achieved performance thresholds anywhere from 4x to 25x faster than agents learning without transfer.

Taylor et al. presented another method to learn inter-task mappings, "Mapping Learning via Classification"[2] (abbreviated MLC in Table 3.1) in a previous work [2007b]. The idea is similar to MASTER, except that *reward* values are stored too, and instead of approximating a transition model and evaluating all possible mappings they train an ensemble of classifiers based on a subset of state variables that are common to both tasks, and use these classifiers to predict action and state correspondences. They tested this in conjunction with their TVITM-PS method (section 3.2) for transfer between ANN policies trained with NEAT. In a transfer experiment from "3 vs. 2 Keepaway" to "4 vs. 3 Keepaway", the transferred policy using the MLC mapping reached the target performance threshold in 240 total simulator hours (including 60 spent training in the source task), whereas it took 420 hours without transfer (although the hand-coded mapping was significantly better than MLC at 180 total hours). A second experiment in the "Server Job Scheduling" domain, from 2 job types to 4, took only 15 generations to reach the target performance threshold with transfer (5 in the source and 10 in the target) versus 21 generations when learning from scratch. In this second experiment, the learned MLC mapping was the same as the hand-coded one.

3.2 Transfer of Learned Policies

[2] In the original paper this method was unnamed, but Taylor later gave it this name in his dissertation [2008].

Imagine that you're trying to find a restroom in an unfamiliar building. In the last building you were in, the restroom was down the right-hand hallway from the entrance, and then 50 meters to the left at the next junction. Assuming this building is the same, you follow the right-hand hallway and then turn left at the T. If this works well, you might try following the same policy even for filling up your car: simply replace "restroom" with "gas station", "hallway" with "street", and "entrance" with "freeway exit". This is the main idea of *policy transfer*: assume the target task is essentially the same as the source, and directly follow the learned source policy in the target environment. If the tasks are similar enough, this can give a great performance boost over the initial random exploration required when learning from scratch; however if the tasks—and in particular the reward functions—are not similar enough, then performance can suffer, which is called *negative transfer* (discussed further in Section 4).

The basic procedure for Transfer of Learned Policies follows:

1. Train a policy in the source task.
2. Either:
    a. Initialize the target policy directly from the source policy (mapping state variables and actions if necessary), and use it as a starting point for exploration in the target task, modifying it directly according to the chosen learning algorithm.
    b. Initialize the target policy as if learning from scratch, but follow the recommendations of the source policy for a fixed "demonstration" period, in order to direct exploration and learning in the target task.

The two variations above are effectively very similar, but make different assumptions about the relationship between the source and target tasks, and so are subtly different in their applicability. The first variation assumes that the target task is a continuation of the source, and is most effective when the tasks are very similar. The second variation assumes that the reward structure of the target is similar to the source, but that the target task may have different branches worth exploring after the demonstration period ends.

Algorithm 3.2.1 lists the Policy Reuse Q-Learning (PRQL) method for Transfer of Learned Policies, as used by Fernández and Veloso in their "robot navigation domain" [2006]. Their experiments involved a set of grid-based mazes, all of which had the same structure (much like the interior of an office building), but different goal states. At each time step the agent senses its *x,y* coordinates, and can choose to move North, South, East, or West. The agent receives a positive reward if it finds the goal within some maximum number of steps (*H*, in the algorithm below), and no reward otherwise.

The primary contribution of PRQL is that it explicitly transfers from a library of source policies, and learns on-the-fly which policies are useful and which ones to ignore.

Algorithm 3.2.1: PRQL *summarized from* [Fernández and Veloso, 2006]

```
1. Given:
    a. $P_1,\ldots,P_n$, a library of policies trained on various source tasks in the problem
       domain, and $P$, an untrained policy for the target task;
    b. $K$, a maximum number of learning trials;
    c. $H$, a maximum number of learning steps per trial;
    d. Various learning rate and discount factor parameters (which are important
       to the implementation but tangential to the transfer).
2. Train in the target task for $k = 1$ to $K$ trials:
    a. Randomly select a source policy $P_k$ from the library, using softmax on the
       calculated policy gain (which begins at zero for each policy).
    b. If $P_k = P$ then follow the policy directly using Q-Learning.
    c. Otherwise, for steps $h = 1$ to $H$:
        i. Choose $p(h)$ to be an exponentially decreasing function in [0,1] (in
           their experiments, $p(h) = 0.95^{h-1}$).
       ii. With probability $p(h)$, take the action recommended by policy $P_k$.
      iii. Otherwise follow the target policy, $P$, using e-greedy action
           selection.
    d. Update the calculated policy gain of $P_k$ by averaging in the total reward
       received for trial $k$.
```

Fernández and Veloso [2006] introduce two new algorithms: Policy Reuse Q-Learning (PRQL), for policy reuse; and PLPR (Policy Library through Policy Reuse), to develop a policy library. PRQL uses Q-Learning but with a chance of following a randomly selected policy from the library (via softmax on the calculated policy gain) at each time step. PLPR adds newly learned policies to the library if they are sufficiently dissimilar from each policy already in the library. The authors demonstrated that PRQL can significantly speed learning. Furthermore, it can effectively learn which, if any, policies from the library to reuse while evaluating them online during learning on the new task, and in their experiments the overhead for this extra exploration was well compensated for in increased performance. However, if no policy is sufficiently similar to the current task then performance will suffer. Also, their algorithm is rather limiting in that the state-space, action-space, and state-transition function must be identical across all tasks in the domain (only the reward function is allowed to vary). These methods are of vary narrow applicability due to the severe restrictions on the domain. However, the general idea of policy reuse and online evaluation of the gain of various policies from a library is potentially useful in broader situations, and could perhaps be combined with other methods for cross-domain transfer. These methods are only useful when solving a series of very related problems within the same domain, and even then care must be taken so that the additional overhead does not dwarf the gains. As presented, these methods have very narrow applicability, and so could be greatly improved by allowing for cross-domain transfer. This could be done by abstracting the policies in the library via state and action inter-task mappings, so that a learned policy could still be followed in the translated domain. I like the idea of a library of reusable policies, and it's encouraging to see that, even when most of the policies in the library are bad for a given task, the algorithm can find the good one quickly enough to make it worthwhile.

Ring [1997] directly uses the same learning agent (CHILD) in all tasks, assuming the same state variables and action space. He trains it using Q-Learning and Temporal Transition Hierarchies (the latter introduced in his 1994 dissertation?).

Ramon et al. [2007] ….

Taylor et al. [2007b] focus on policy search methods, where policies are represented as ANNs mapping states to actions (where there is one output node for each possible action, and the highest activation wins). Given an inter-task mapping of state variables and actions, their TVITM-PS algorithm constructs an ANN for the target task given an ANN for the source, copying links from the analogous nodes. In the case of an incomplete mapping, the target ANN is augmented with small random weights to connect the remaining nodes. The authors convincingly demonstrated that TVITM-PS can reduce total learning time (source + target vs. target from scratch) when the source task is an easier version of the target. Their methods allow the dynamics of the tasks to differ, and are flexible in how much domain knowledge they require (in terms of the inter-task mapping). A disadvantage is that there is no mechanism to prevent overfitting to the source task. These methods are potentially very useful to any complex RL problem for which there are one or more simpler formulations that could be used as training, and for which we wish to use an ANN-based policy search agent. Their results show that use of an incomplete inter-task mapping may be much worse than trying to learn the mapping. Also, using an overly trained source policy can significantly impede generalization in the target task. It would be useful to allow some sort of "relaxation" of the source policy when transferring to the target agent to avoid overfitting.

3.3 Transfer of Options (Learned Macro Actions)

Transfer of Options is also sometimes called *skill transfer* [Torrey et al., 2007]. The main idea is to abstract behavioral knowledge into *skills*, which can then be leveraged in future tasks (or even later in the same task). This is akin to a human leaning to walk—once you've mastered the low-level muscle actuations required to balance and step, *walk* becomes a new primitive behavior that can be applied in a great variety of complex tasks. Transfer of Options is conceptually no different. The underlying assumption required for effective transfer is that the target task will require some of the same skills (compound behaviors) that are useful in the source task.

The basic procedure for Transfer of Options follows:

1. While training in the source task, identify important sub-goals:
    - May be human-supplied (e.g. "learn how to open a door").
    - May be learned automatically as high-reward or good-to-be-in-but-hard-to-get-to states.
2. Learn micro-policies to achieve these sub-goals through RL on the state-action sequences leading to them.
3. Translate these micro-policies to the target task, and store as a special set of macro-actions, *O*.
4. Train as normal in the target task, but add *O* to the action-space.

As mentioned above, an *option* is a micro-policy for achieving a particular useful outcome, and as such, its policy will differ from the base-policy used by the learning agent. Typically, options are composed of three parts: (1) a (possibly stochastic) policy specifying which primitive actions to take in each state; (2) an *initiation set* function which identifies those states in which the option is allowed; and (3) a *termination condition* which specifies (again, possibly probabilistically) when the option has completed based on the state [Konidaris and Barto, 2007]. Thus, it is not generally possible for the target agent to know how long it will take to execute a given option, nor whether it will achieve its intended result. To the agent, options are indistinguishable from primitive actions, and the agent can no more abort an option-gone-awry than it can any other action.

Algorithm 3.3.1 lists the Skill Transfer through Agent-Space Options[3] (STASO) method for Transfer of Options, as used by Konidaris and Barto in their "Lightworld" experiments [2007]. Their Lightworld domain is a set of tasks in which each instance is a sequence of adjoining rooms on a 2D grid, connected by doors which may be opened via wall-panels that may or may not require a key (which is also in the room). They call it "Lightworld" because each important object (doorway, panel, key) emits a different colored light, which the agent can perceive via 12 light sensors (three per side). These light sensors comprise the agent-space, whereas the problem-space is simply the agent's current location (room number and *x*,*y* coordinates), whether the door is open, and whether it has the key. The goal is to exit the rooms as quickly as possible.

Algorithm 3.3.1: STASO *summarized from* [Konidaris and Barto, 2007]

[3] This is my name for their method, combining their terms "skill transfer" and "agent-space options".

1. Pre-define a set of useful skills that are potentially applicable across all tasks in the domain.
2. Choose an agent representation for the domain, such that the state variables are separated into *problem-space* versus *agent-space*, where the former is sufficient to render each task Markov (presumably) and the latter represents perceptions common to all tasks in the domain.
3. In the source task(s):
    a. Train the agent using TD learning (they used Sarsa(?) with e-greedy action selection).
    b. Simultaneously learn an option in agent-space for each predefined skill, using trace-based tree-backup updates.
4. In the target task(s):
    a. Use the learned agent-space options as primitive actions.
    b. Also learn options for the same skills in problem-space, which are non-transferrable, but more specific to the task at hand and therefore more effective (once learned) than the agent-space options.

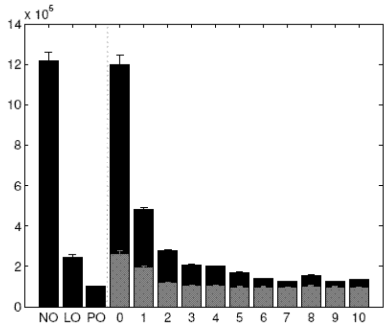

Figure 5: Total steps over 70 episodes for agents with no options (NO), learned problem-space options (LO), perfect options (PO), agent-space options with 0-10 training experiences (dark bars), and both option types with 0-10 training experiences (light bars).

Fig [#]. Reprinted from [Konidaris & Barto, 2007]. *NOTE: need to change or remove the image caption.*

Figure [#] shows some of their results. This experiment compared agents with varying amounts of option transfer across randomly generated lightworld instances, and measures the mean total number of steps taken to reach the exit over 70 episodes in the test task (smaller is better). The left-most bar is the control group of agents acting without transfer and without options (but still learning over the 70 episodes in the test task). The second bar allows the agent to learn the options in problem-space over the 70 episodes, and the third bar grants the agent with already perfectly-learned problem-space options (so for example *pickup-key* would be guaranteed to succeed in a minimal number of steps if the room has a key). The remaining eleven bars introduce agent-space options, and give the transfer results after 100 training episodes in each of *n* random source tasks (where *n* is the number below the bar, so bar 0 does not transfer anything but just allows the agent to learn options in agent-space in the test task). The taller numbered bars are with agent-space options only, and the shorter gray bars allow learning of problem-space options as well as transfer of action-space options (so the gray bars actually correspond to Algorithm 3.3.1). These results show that with sufficient training experience in this particular domain, their transfer algorithm is almost as effective as perfect knowledge.

Three notable weaknesses of Algorithm 3.3.1 are:

1. It requires an oracle (e.g., human) to pre-determine important skills.
2. It allows transfer only between tasks that share an agent-space.
3. It requires significant extra overhead to learn the option policies.

The third weakness is characteristic of all methods in this class, however the first two have been addressed by other researchers.

Bonarini et al. [2006] created a "Self-Motivated Incremental Learning" (SMILe) algorithm, by which the agent may define its own skills through self-directed exploration of the task environment. Their assumption is that the agent's state-space remains constant throughout all tasks, and that only the reward function may change. This is more restrictive than the agent-space restriction above, but allows all options to be in problem-space. Their main contribution is to allow the agent to discover important skills on its own, which eases the burden on the human designers. SMILe begins by randomly exploring the environment in order to estimate a transition model. It then identifies "interesting" states that are relatively hard to get to but easy to leave, and learns options to achieve each of those states. SMILe can then use those options as primitive actions when presented with a task (i.e. reward function) in that environment. They compared SMILe to standard Q-Learning on a series of five tasks in a simulated two-room environment containing light switches, a door, and a charger. The tasks progressed in complexity from "charge" to "charge, move to left room, close the door, switch the light off" (3). It took Q-Learning nearly two-million steps to learn all five tasks in sequence, whereas it took SMILe barely more than one-million, including its quarter-million "self-development" steps.

Torrey et al. [2007], in their "Relational Macro Transfer via Demonstration" (RMT-D) algorithm, also use automatically extracted (rather than human-supplied) skills, but in a more task-dependent way. They assume that some source task has already been learned (by any method), and that the agent has retained a transcript of the *state-action-reward* sequences for each training episode. They then use inductive logic programming to learn one or more "macros" (action sequences) that distinguish high-reward episodes from low-reward ones. They refine these macro sequences into finite state machines (FSMs) by inducing transition rules from the positive and negative examples among the saved episodes. They also support transfer between tasks with different state- and action-spaces, by applying an inter-task mapping to the learned FSMs before transferring to the target task. They tested RMT-D in the "*m*-on-*n* BreakAway" class of tasks in the RoboCup Soccer domain, in which there are *m* friendly players trying to score a goal against 1 goalie and *n* - 1 other defenders. They tested transfer from 2-on-1 BreakAway to both 3-on-2 and 4-on-3 BreakAway. Compared to standard RL, RMT-D exhibited a significant jumpstart in the probability of scoring a goal in both transfer scenarios, with approximately five-times greater likelihood after 500 training games.

3.4 Transfer of Rules (Advice)

Transfer of Rules (Advice) abstracts logical rules from the source task policy to follow in the target task. Madden and Howley [2004] always use the rules for unexplored states. Torrey et al. [2006] use a decaying linear function based on weighted state features to bias towards advice. Taylor and Stone [2007a] force advice-following for fixed number of episodes to train their FA, then allow it either as "Extra Action", "Value Bonus", or "Extra Variable".

Transfer of Rules (Advice): First train a policy in the source task. Then abstract that policy into general rules of the form "if (predicate expression on state variables) then (action)". Madden&Howley: if not explored then use advice, and negatively bias non-chosen actions. Torrey&al: use a decaying linear function based on weighted state features to bias Q-values towards advice, strongly at first but less over time. Taylor&Stone: force advice following for fixed number of episodes to force-train the FA, then allow it either as "Extra Action", "Value Bonus", or "Extra Variable". All these methods are fairly robust against negative transfer, as bad advice can be recovered from (e.g. Torrey's example). To be any use, the high-level abstractions used to describe the state conditions in the advice must be applicable in the target task, which will require a careful inter-task mapping if the tasks are substantially dissimilar.

1. Train a policy in the source task.
2. Abstract the source policy into a set of rules of the form "IF *predicate expression on state variables* THEN *action*".
3. While training in the target task, always follow advice for unfamiliar states, and gradually discount the advice as learning progresses.

Advantages:

- Agent can take advice from multiple sources (including humans).

- Rules are more human-friendly than abstract policy functions.

Drawbacks:

- Rules may have to be carefully scaled to avoid negative transfer (e.g., "*distance* < 20" probably doesn't mean the same thing in both tasks).
- Agent forced (at first) to follow rules, but cannot modify them.

Algorithm 3.4.1: Progressive RL *summarized from* [Madden and Howley, 2004]

```
1. Given:
     a. Some high-level abstraction of the state-space shared by all tasks in the
        domain, $H(S)$;
     b. A set of (state, action) samples from a trained agent in the source task.
2. Train a symbolic learner, $R$, to map from $H(s)$ to $a$ for all samples $(s, a)$. (They
   compared the C4.5, Naïve Bayes, and PART algorithms for training their
   symbolic learner, with similar results, suggesting that any reasonable training
   method may be used.)
3. While training in the target task (using any TD learning method; they used Q-
   Learning):
     a. If the current state, $s$, has not been encountered before by the agent:
          i. Take the action $a$ suggested by $R(H(s))$.
         ii. Update the Q-values of all other actions in state $s$ to some
             small negative bias, to increase the likelihood that $a$ will be
             chosen again next time (unless it results in a large negative
             reward).
     b. Otherwise use e-greedy action selection based on the current Q-values
        for state $s$.
4. Optionally repeat steps 2 and 3, retraining $R$ with samples from the current
   task once it is sufficiently mastered, in order to transfer to a new target task.
```

Madden and Howley [2004] present "Progressive RL", which features a reinforcement learner and a symbolic learner working in tandem, where the latter is responsible for abstracting the policy of the former, and for suggesting promising actions when the RL agent encounters an unexplored state. In their experiments, they used tabular Q-learning for the reinforcement learner, and compared C4.5, Naïve Bayes, and PART for the symbolic learner. The authors demonstrated that the rules abstracted from simple tasks can greatly increase performance on more complex tasks such that (in some cases) it is more efficient to "practice" on simpler tasks first rather than attempting to directly solve a complex task from scratch. Their methods allow free and independent choice of RL algorithm and symbolic learning algorithm. The disadvantages are that the domains must be identical (state-space must have fixed dimensions, and same action space), that the simpler task must be "mastered" (we need a near-optimal policy to abstract), and that there's no way of knowing a priori—when faced with a difficult task—whether it would be worthwhile to practice on a simpler task first. Their methods are general and flexible, so could easily apply to any RL situation in which there is a sequence of related tasks in the same domain. However, such methods (as currently defined) do not apply to cross-domain tasks, and would be of limited use if there is no natural progression of difficulty. As discussed already, the state-spaces must be quite similar and the action spaces identical to apply this form of knowledge transfer between tasks. There is no such restriction on the reward function, but if it's markedly different on the target task than on the previous experience then this method would backfire. The authors point out that their symbolic learners use only propositional logic, so are capable of learning things like "don't walk into a wall" but not complex strategy. Some extension to the introspective part of Progressive RL to allow for abstracting of multi-step plans could make it more effective. Also, it currently relies on a human-specified projection of the state-space into relevant features, but perhaps could be extended to learn such an abstraction and even learn homomorphisms between related tasks in similar but different domains. I like the idea of separating the algorithm into two independent parts—essentially a task learner and a policy analyzer. Though of limited applicability as they propose it, such a division of labor could potentially be applied in similar transfer learning situations to give a great deal of flexibility in agent design.

Taylor and Stone [2007a] introduce the "Rule Transfer" method, by which a set of exemplary state/action pairs from the pre-trained source policy is distilled into a "decision list", which is then translated to the target domain. This knowledge may then be treated as "advice" by a learning agent in the target domain, in one of three ways: (1) *Value Bonus* increases the Q-value of the recommended action by a fixed amount (only applies to agents using value-function approximation); (2) *Extra Action* allows the agent to choose "follow advice" as a new single action; (3) *Extra Variable* creates a new state variable which points to the recommended action. The main advantage of Rule Transfer is that it allows transfer between tasks in different domains. The three transfer methods are all low-cost, and allow flexibility in the design of the target agent. Because the knowledge is encapsulated in rule form, the design of the target agent need not be similar to the source agent. The main disadvantage is that the transfer requires an inter-task mapping to be provided (although the authors sketched some ideas of how such a mapping might be learned). Also, it's not clear in advance how much help such transfer might be for a given task, and what configuration would be most useful, requiring more human intervention. Such knowledge transfer could be very useful in many RL applications provided that there existed useful inter-task mappings. Their results indicate that the transfer is most useful in jump-starting agent performance, so the effort to transfer must be weighed against the default (transferless) learning curve. There is nothing in the research they present that I would necessarily avoid. However, their experiments indicate that a great deal of human intervention is required in order to effectively use their methods. The authors sketch out an idea of how inter-task mappings might be learned. As presented, their sketch has a long way to go to become a general purpose algorithm, but this is probably the most important improvement we could make to increase the applicability of their methods. There are also possible improvements in how the advice is used by the target agent. For instance, it might be useful to decrease the Value Bonus over time, as experience in the new domain eclipses the usefulness of the rule recommendations. Also, perhaps the various advice methods could be combined, or multiplied to allow advice from multiple related source tasks (perhaps each with only a partial mapping to the target domain).

3.5 Transfer of Action Values for TD Learning

Transfer of Action Values retains a function approximator (FA) from the source task to bias search in the target task. Konidaris and Barto [2006] train their FA in “agent-space” to avoid translation. Taylor et al. [2007a] linearly combine the old FA estimates with the new. Torrey et al. [2008] transfer a Markov Logic Network via a fixed demonstration period.

Transfer of Action Values (for TD Learning): Main idea is to first train an action-value function approximator in the source task, and then use those action-values to bias search in the target task. Konidaris&Barto: train an FA in “agent-space” which directly translates to the new task. Taylor&al: map the target states/actions back to the source, and linearly combine the looked-up Q-values with the current estimates for the new task. Torrey&al: learn a Markov Logic Network (really just a fancy kind of FA) from source task experience, and directly use it (translated via an inter-task mapping if needed) for a fixed “demonstration” period in the target task to get the target agent’s FA off to a good start. All of these methods require extra memory and computation, since two FAs must be kept on hand (at least during early training in the target), and often a translated lookup is required. My main idea for improvement is twofold: (1) incorporate the real cost of maintaining and looking-up the old FA into its use, to bias the new agent away from it; and (2) allow the agent to estimate the relative value of the old FA versus the currently-being-learned one in different parts of the state-space, and to dynamically discount the old FA (perhaps in a state-dependent way) as the new one improves, until it eventually autonomously decides to forget it altogether.

1. Use Temporal Difference learning (TD) in the source task to train a function approximator (FA) to estimate state-action values.

2. Use TD again in the target task, and use the (translated) FA from the source task to bias action-value estimates in the target (either for a fixed "demonstration" period or via some linear combination of the source and target FAs).

Advantages:

- Agent can flexibly combine the old estimates with the current evolving ones (perhaps even in a state-dependent way).

Drawbacks:

- Extra memory and computation is required, since now two FAs must be maintained instead of one (and the old one may require translation each time it is used).

Algorithm 3.5.1: MLN Transfer *summarized from* [Torrey et al., 2008]

1. Retain a Q-value function, $Q(s,a)$, from about halfway through the learning curve on the source task (see the text for why not to use the fully-trained Q-values), along with a great many sample states.
2. Train a Markov Logic Network (MLN) from the sample state-action values:
    a. Hierarchically cluster all (s,a) samples into some number of ordered bins based on the Q-value of each sample (so (s_1,a_1) and (s_2,a_2) will be in the same bin if $Q(s_1,a_1)$ and $Q(s_2,a_2)$ are nearly equal).
    b. Use inductive logic programming (they used the Aleph algorithm) to learn a set of rules to classify the samples into their bins based on features of the state-space.
    c. Rank all proposed rules by precision, and greedily choose the subset that maximizes the overall harmonic mean of precision and recall.
3. While training in the target task (using any TD learning method; they used used Sarsa(?) with a Support Vector Machine function approximator), for some fixed "demonstration" period:
    a. For each encountered state, s, and possible action, a, estimate $Q(s,a)$ = the sum over all bins b of $(p_b * Q(t_b,a))$, where p_b is the inferred probability (based on the MLN) that $Q(s,a)$ falls into bin b, and t_b is the state in b most similar to s, based on the MLN rules they both satisfy.
    b. Take the action with the highest estimated Q-value, and train the target agent (off-policy) normally, based on the received reward.
4. At the end of the "demonstration" period, discard the MLN and continue learning using the now partially learned target policy.

Konidaris and Barto [2006] consider an abstract "agent-space" of "sensations" separately from the "problem-space" of the task, and to learn a value function for the agent-space that can be used as a predictor of the state-value in subsequent "reward-linked related tasks". In their experiments they used Sarsa(?) for the RL, and a gradient-descent supervised learning algorithm to approximate the agent-space value function. The main advantage of their method is that it can greatly reduce the initial time to solve a novel task without requiring any additional human intervention in the form of shaping rewards. The main disadvantages are the extra overhead and that it may slow convergence to an optimal policy in the new task (as evidenced in the graph above). This method of predicting state-value based on agent sensations seems generally useful in cases of "reward-linked related tasks". Their division into problem-space versus agent-space also allows for a simplified state-descriptor while retaining other sensations as potential reward predictors. This is useful in part because many learning algorithms benefit from minimal state-spaces. The authors focused on reducing the time needed to find a first solution, which their method worked well at, but at the cost of slowed convergence toward an optimal solution, so this method should be avoided when rapid convergence is the goal. It seems that the convergence problem could be taken care of by quickly decaying the pseudo-reward signal, or perhaps only using it the first time a new state is explored. I like the idea of an agent's "sensation" as being distinct from the minimal state-descriptor of the task. In general, it seems that it would be difficult for an agent to autonomously (without human guidance) learn to distinguish which of its sensations identify the problem-state, which contain useful reward predictors, and which are irrelevant to the task at hand, but doing so could greatly increase learning performance on new but related tasks.

3.6 Transfer of Samples (Recorded Experience)

Transfer of Samples collects many (*state*, *action*, *reward*, *new-state*) samples from the source task to use as additional training for the target agent. Taylor and Stone [2007b] transfer samples from one agent to another on the same task. Lazaric et al. [2008] use batch RL to train on transferred samples based on their calculated "relevance". Taylor et al. [2008b] use recorded samples on-line as needed to bolster the confidence of a model-based learner.

Transfer of Samples (Recorded Experience): Main idea is to collect a great many (*state*, *action*, *reward*, *new-state*) samples while training in the source task, and then use these (possibly translated if necessary) directly for simulated training in the target task. This both transfers hopefully useful experience, and saves the target agent from having to collect as many of its own samples, which are often much more costly to obtain than simulated samples. Lazaric&al: use batch RL to train on transferred samples based on their calculated "relevance" in the new task (and choose between multiple potential source tasks based on inferred inter-task "compliance").

Taylor&Stone: directly transfer (s, a, q)-type samples for initial training of the target agent. Taylor&al: use recorded (s, a, r, s′) samples on-line for a model-based learner, as needed, to fill in the gaps when not enough target task data is available to properly estimate the transition and reward functions for the current state/action pair (also using an inter-task mapping).

1. Collect many (*state*, *action*, *reward*, *new-state*) samples from the source task.
2. Either:
    - Train the target agent offline using these samples.
    - Use selected samples as needed in the target task to support current observations.

Advantages:

- Saves the target agent from having to collect so many of its own samples (which is often much more costly than offline computation).
- Well-suited to instance-based model learning algorithms (unlike most other methods which are primarily for TD learners).
- Can easily transfer between different types of agents on the same task (e.g. for a quick upgrade).

Drawbacks:

- Extra overhead to collect and store source samples.
- Of limited use if exploration in the target task is cheap.

Algorithm 3.6.1: TSBRL *summarized from* [Lazaric et al., 2008]

```
1. Collect m (state, action, reward, new-state) samples from each of n possibly
   relevant source tasks, and store them in separate sets $S_1,…,S_n$ (so $S_i$ is the set of
   samples from source task i).
2. Collect t (state, action, reward, new-state) samples from the target task, and store
   in set T (usually t is much less than m, otherwise we wouldn't need to
   transfer).
3. For each source task i, calculate the "compliance" of $S_i$ and T:
    a. For each sample $x_1 = (s_1,a_1,r_1,s_1')$ in $S_i$ and $x_2 = (s_2,a_2,r_2,s_2')$ in T:
        i. Set w to a Gaussian-weighted "distance" between $(s_1,a_1)$ and
           $(s_2,a_2)$, normalized over all samples in $S_i$, using any distance
           metric appropriate to the domain.  So w will be greatest for
           pairs that are very close, and fall sharply as distance increases.
        ii. The reward compliance, $RC(x_1,x_2)$, is w times the Gaussian-
            weighted difference $|r_1 – r_2|$ (so RC is high only when the
            rewards are similar).
```

        iii. Similarly, the transition compliance, TC(x_1,x_2), is w times the Gaussian-weighted "distance" between s_1' and ($s_1 + s_2' - s_2$) (assuming a continuous transition model on a continuous state-space).

    b. The task compliance, C(S_i,T), is the product of the weighted sums of RC(x_1,x_2) and TC(x_1,x_2), over all x_1 in S_i and x_2 in T.

4. For each sample x from each source set S_i, calculate the "relevance" of x and T:
    a. Set c to the compliance of x with T, normalized over all samples in S_i.
    b. Set d to the "average distance" between x and the samples in T (using the same distance metric as in 3.a.i above).
    c. The relevance, R(x,T), is the Gaussian of $(1 - c)/d$, meaning that the most relevant samples for transfer are either very compliant (c close to 1), very far from most of T (d large), or both.
5. Set $U = T$, and choose $m - t$ more samples to add to U, drawn from all the S_i weighted by each S_i's "compliance" with T, and within each S_i choosing samples according to their "relevance" with T.
6. Train the target agent on the m samples in U using any batch RL algorithm (they used Fitted Q-Iteration).

Taylor and Stone [2007b] define an "offline representation transfer" (ORT) algorithm for use between representations that use value function approximators (in their experiments, an ANN and an RBF, both trained using Sarsa). The idea is that the source agent records many (*state, action*, *Q-value*) tuples, and these are then used to train the target agent's function approximator before it interacts with the real environment. This might be desirable in instances where the old agent uses an inadequate representation for the task at hand, so we replace it with a new agent but don't want to lose the accumulated experience. A clear advantage is that adequate task performance can be maintained when switching representations. Also, the "offline" portion (training the new agent from the source data) is typically much less costly than acquiring experience in the real domain. The main disadvantages are: (1) limited usefulness; (2) time it takes to gather the training samples from the source agent; and (3) the restriction that both agents use value function approximation. This seems most useful when experimenting with different agent representations for a given task. Leveraging the ORT algorithm would allow faster switching between representations. Care should be taken that the transferred Q-values do not bias the new learner in the wrong direction; if the old agent's performance was really poor, it might not be worth transferring at all. The ORT algorithm would be more useful if it were not restricted to agents with value function approximators.

3.7 Transfer of Weights or Functions

Transfer of Weights or Functions: Idea is to directly reuse or transfer part or all of the FA from the source task to the target. Tasks must either be very similar or there must be a good inter-task mapping so that weights can be faithfully transferred. Taylor&Stone: "complexification" transfers back to the same task, but allows a more complex internal representation in the FA, such as if learning has plateaued because the current FA isn't sophisticated enough. Taylor&al: also require the same type of FA, but transfer between tasks via a mapping. Jaskowski&al: introduce *crossbreeding* for GP to directly reuse subtrees of evolved programs in other tasks (also requires a mapping for state variables, and that the operators—internal nodes—be the same). In all of these cases, once the transfer happens (usually before training in the target) it is immediately incorporated into the target agent with no further computational or memory requirements. These methods assume a great deal of similarity between the source and target tasks, and don't allow the learning agent to "know" that its biases are coming from a previous task.

Algorithm 3.7.1: GPCRC *adapted from* [Jaskowski et al., 2008]

```
1. Given: some population, $S$, of individuals from one or more source tasks, evolved using Genetic Programming (GP) using some fixed set of operators for the internal nodes (they used *And*, *Or*, *Nand*, and *Nor* to evolve solutions to digital logic design tasks).
2. Train a population, $T$, in the target task using GP as normal (for direct policy search), but with the additional *crossbreeding* operator:
    a. Randomly select a target individual, $t$ from $T$, and source individual, $s$ from $S$.
    b. Randomly select a sub-tree of $t$ to mutate, and replace it with a randomly selected sub-tree from $s$.
    c. For each terminal $x$ in the transplanted sub-tree that does not have a correspondence (or defined inter-task mapping) in the state-space of the target task, bind all occurrences of terminal $x$ to a randomly selected state feature.
```

Jaskowski et al. [2008] present code-reuse in multi-task GP. They use standard GP applied to benchmark digital logic design tasks, with And, Or, Nand, and Nor functions, but introduce a "crossbreeding" operator, which is like genetic crossover but applies between members of different "species" (i.e., individuals from distinct populations that are working on different tasks). In their experiments, the populations were evolved (i.e. tasks were solved) in parallel, and crossbreeding replaced mutation. Their crossbreeding operator is easy to implement in any GP system (the only difficulty being choosing how to relabel the terminals), and significantly improved learning performance in their tests. It is also flexible in that it can be used either when solving tasks in parallel or sequentially, and can easily be extended to leverage multiple source tasks. However, it only applies to GP, and there is a real risk of negative transfer (decreased performance) on some tasks, depending perhaps on how (dis)similar the tasks are and how crossbreeding is applied. This could be useful in many GP applications in which there are many related tasks from the same domain. As the authors say, "the problems' search spaces have to overlap" in order for code reuse to be effective. Careless application of crossbreeding could result in negative transfer, but this should generally be avoidable if the evolutionary parameters are configured correctly. The authors supposed that the negative transfer to the even-parity tasks resulted not from crossbreeding directly but from elimination of the mutation operator (because in their experiments they replaced mutation with crossbreeding) which impeded exploration of the search space. It might be more effective—especially given a large library of evolved solutions to different related tasks—to dynamically vary the rate of crossbreeding from various sources based on the measured performance of the resulting individuals. If a random insertion of code proves useful, then try inserting more code from that same solution; but if it's detrimental, then slightly decrease the chance of reusing that same source. I have often thought of learning in terms of the development and careful reuse of strategies from past experience, and this paper presents and interesting instantiation of that concept. I'm curious how effective such code resuse could be on less similar tasks given some cross-domain mapping.

Taylor et al. [2007a] focus on Temporal Difference methods to learn state-action values, using a Semi-Markov Decision Process (SMDP) Sarsa(?) algorithm, and a Cerebellar Model Arithmetic Computer (CMAC) function approximator to estimate the expected total reward given a state configuration and choice of action. Their novel contribution is the transferring of weights from an already-learned CMAC to a new CMAC for a different task, based on a hand-constructed inter-task mapping. The most notable advantage of their methods is that it allows transfer between tasks with different state and action spaces (unlike most previous work). They also clearly demonstrated that training first on a simpler, related task, can greatly reduce total training time to learn a target task. The main disadvantage is that the transfer requires an inter-task mapping to be provided. Also, the presented algorithm only applies to weight-based state-action value estimates. Their behavior transfer algorithm could be applied to many reinforcement learning domains in which there are multiple related tasks to be learned. It would be especially useful when there is a natural progression of task complexity. Currently, however, this only applies for weight-based state-action value function approximation learners. There is nothing in the research they present that I would necessarily avoid. One interesting caveat they discovered is that there seems to be an optimal amount of training on an easier task before progressing to a harder task in order to minimize total training time, and it's yet unknown how to discern this on-line (i.e., when to switch from task A to task B). I agree with the ideas hinted at by the authors: expand the behavior transfer algorithm to apply to more types of RL algorithms (such as policy search methods); automatically discover inter-task mappings; develop heuristics of when to progress to the next more difficult task (assuming that we don't really need proficiency at the training tasks); allow transfer from multiple related source tasks.

4 Discussion

The focus of this section is to address the following questions:

- What can we infer from the presented methods about the current state of the field of Transfer Learning?
- To what extent has Transfer Learning research achieved its objectives?
- What major challenges remain?
- What further research is needed to advance this field towards its objectives?

We shall address each of these questions, in order, in its own subsection.

4.1 The State of the Art

What can we infer from the presented methods about the current state of the field of Transfer Learning? Two things are clear just from the dates of the referenced research: (1) it is young; (2) it is growing fast. Of the twenty-one presented methods, fifteen were published in the last two years. Most of the current research is focused on proposing new transfer methods, rather than on studying, relating, or improving existing methods. Because many of these methods have been developed essentially simultaneously, there has yet been little apparent inter-method transfer between the different research groups. However, every single paper has brought further evidence that transfer does indeed work.

4.2 Are We There Yet?

To what extent has Transfer Learning research achieved its objectives? To answer this, let us revisit the objectives of Transfer Learning as given in the Introduction:

- To make hard tasks more tractable.
- To facilitate faster learning of new tasks.
- To reduce the sample complexity of new tasks.
- To enable agents to autonomously choose when to reuse what knowledge, and how.
- To enable autonomous, continuous, and progressive learning, with less dependence on human guidance.

The methods and results we've presented clearly demonstrate that all five of these objectives have been met to some extent. Though no paper has yet attempted transfer on a task that demonstrably cannot otherwise be learned, all papers have reported some performance gains, and some have even shown better asymptotic performance when using transfer. The TIMBREL method, for instance (Section 3.6), achieved significantly better asymptotic performance in the target task versus learning without transfer, and the published TGR results (Section 3.2) show this distinction as well (though it's not clear with the TGR results that learning without transfer would not eventually catch up, if given enough time). Many methods have shown that when the source task is a smaller version of the target task (perhaps even artificially constructed for training purposes), the total time to learn both the source and target tasks, including transfer, can be significantly less than the time it would take to learn the target task from scratch (CHILD, TGR, and TVITM-PS from Section 3.2; SMILe from Section 3.3; Q-Value Reuse from Section 3.5; ORT from Section 3.6; Complexification and TVITM-VF from Section 3.7; the other methods would probably exhibit this too, but it wasn't reported in their results). The three methods in Section 3.6, as well as the MASTER method (Section 3.1), were all designed specifically to limit the required number of samples drawn from the target task, and all transfer learning methods—when they work—achieve this as a side-effect. Both MASTER and MLC (Section 3.1) help facilitate autonomous transfer by automatically constructing inter-task mappings; MASTER, PRQL (Section 3.2), and TSBRL (Section 3.6) all explicitly assess the potential usefulness of each possible source task, when given more than one from which to choose.

The fifth objective is the most general and also the most elusive. None of the presented methods decreases human involvement in design of the learning agents; on the contrary, more design work is required in order support the transfer method. However, given that we are transferring knowledge, methods such as MASTER and MLC relieve humans of having to hand-code an inter-task mapping. PRQL and TSBRL also take steps towards automatic selection of source tasks, further alleviating human involvement. Furthermore, if we consider the performance gains that transfer learning can achieve, it would probably take a lot more human guidance (perhaps in the form of tailored heuristics and shaping functions) to achieve similar gains without transfer. Although most of the methods presented still consider tasks individually, a few methods explicitly address the continual learning case, most notably CHILD (Section 3.2), Progressive RL (Section 3.4), and SMILe (Section 3.3), with the latter showing the most promise because it does not even require notification when the task (i.e. reward function) changes.

Transfer Learning has not yet realized its objectives to its full potential (and there will always be room for improvement), but it has taken important steps toward each. Not all hard tasks are now tractable, but research has shown that some tasks may be made easier with transfer. Almost any task may potentially be learned faster with transfer, but we might not always have an appropriate source task to transfer from. The last two objectives are the hardest, in part because the only known general purpose methods for enabling autonomy (such as MASTER) are computationally too complex to scale well, and these objectives remain an important area of future research.

4.3 Why Not?

What major challenges remain? The most important obstacles to the advancement of Transfer Learning relate to the last two objectives above, which attempt to endow the learning agent with sufficient autonomy to solve its own problems without coming to a human for help. Enabling such autonomy requires solving Transfer Learning for the general case, and not just for individual special cases. More specifically, before we can create fully autonomous general purpose learning agents, we need to solve the following problems:

- Efficiently support large libraries of task knowledge (which may require or benefit from some standard universal knowledge representation).
- Quickly assess the relevance of multiple source tasks (or multiple transferable instances of saved knowledge) in order to choose an appropriate one for transfer.
- Automatically determine how to apply the chosen knowledge to the target task (perhaps via a learned inter-task mapping).
- Automatically detect *negative transfer* when it occurs (transferred experience impeding performance), in order to discard the bad transfer and either select different knowledge to transfer or continue learning without transfer.
- Automatically detect when the current task changes, and adapt transfer strategy in response.

4.4 What's Next?

What further research is needed to advance this field towards its objectives? As noted in Section 4.1, this field is far from mature. It is rich in methods but poor in analysis, like piles of bricks without enough mortar. Certainly the methods need to be developed and evolved, but in order to solve the five problems identified in the previous section, we most urgently need further analysis. Here are some specific recommendations:

- We need a broad comparison of existing methods applied to a variety of "benchmark" tasks, to gather preliminary data on their relative efficacy for different learning scenarios.
- We need more experiments using variations of methods, to help them evolve into generally robust forms (e.g., we'd like to be able to say things like "we use a parameter value of 0.95 because it's widely accepted as near-optimal in the literature").

- We need more research on efficiently choosing and transferring from a large library of potential source tasks (or otherwise encoded knowledge), which is currently an under-researched area of TL.
- Some of the most general methods have high computational complexity, so we need to research heuristics on how to efficiently approximate the same effect.

5 Conclusion

This survey has defined the field of Transfer Learning as applied to Reinforcement Learning based on its context within the broader field of Machine Learning and by the problems it attempts to solve. We have presented a novel classification of current TL methods based on *what knowledge* they transfer, along with in-depth analyses of the algorithms, their strengths and weaknesses, how their authors used them, and what results they achieved. Based on the surveyed methods, we conclude the following:

- The field is immature yet growing;
- TL is demonstrably quite effective in many cases, but we don't yet have broad general purpose methods to apply it in arbitrary RL domains;
- To advance the field further, we most urgently need more cross-method analysis to provide a more cohesive framework on which to build the next generation of transfer learning methods.

Transfer Learning is absolutely necessary in order to fulfill the dream of fully autonomous learning agents. Truly adaptive learning agents must be able to efficiently index many source tasks, quickly assess their relevance, and flexibly change transfer strategy in response to the demands of the current task. There have already been remarkable achievements in this young field, but many challenges remain. If the current level of research activity is any indication, we are likely to see even greater advances in the near future.

References

[Banerjee et al., 2006] Banerjee, B., Liu, Y. & Youngblood, G.M., 2006. ICML workshop on "Structural Knowledge Transfer for Machine Learning".

[Bonarini et al., 2006] Bonarini, A., Lazaric, A. & Restelli, M., 2006. Learning Reusable Skills through Self-Motivation. In *Proceedings of the 23rd International Conference on Machine Learning*. Pittsburgh, PA, pp. 1-4. Available at: http://home.dei.polimi.it/lazaric/papers/bonarini2006learning.pdf.

[Caruana, 1997] Caruana, R., 1997. Multitask Learning. *Machine Learning*, 28(1), 41-75. Available at: http://www.springerlink.com.ezproxy.library.wwu.edu/content/x4q010h7342j4p15/fulltext.pdf.

[Fernández and Veloso, 2006] Fernández, F. & Veloso, M., 2006. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. Hakodate, Hokkaido, Japan: ACM, New York, NY, pp. 720-727. Available at: http://delivery.acm.org/10.1145/1170000/1160762/p720-fernandez.pdf.

[Jaskowski et al., 2008] Jaskowski, W., Krawiec, K. & Wieloch, B., 2008. Multi-Task Code Reuse in Genetic Programming. In *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*. Atlanta, GA: ACM, New York, NY, pp. 2159-2164. Available at: http://delivery.acm.org.ezproxy.library.wwu.edu/10.1145/1390000/1389040/p2159-jaskowski.pdf.

[Konidaris and Barto, 2006] Konidaris, G. & Barto, A., 2006. Autonomous Shaping: Knowledge Transfer in Reinforcement Learning. In *Proceedings of the 23rd international conference on Machine learning*. Pittsburgh, PA: ACM, New York, NY, pp. 489-496. Available at: http://delivery.acm.org/10.1145/1150000/1143906/p489-konidaris.pdf.

[Konidaris and Barto, 2007] Konidaris, G. & Barto, A., 2007. Building Portable Options: Skill Transfer in Reinforcement Learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*. pp. 895-900. Available at: http://www-all.cs.umass.edu/~gdk/pubs/agentopt.pdf.

[Lazaric et al., 2008] Lazaric, A., Restelli, M. & Bonarini, A., 2008. Transfer of Samples in Batch Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning*. Helsinki, Finland: ACM, New York, NY, pp. 544-551. Available at: http://delivery.acm.org/10.1145/1400000/1390225/p544-lazaric.pdf.

[Madden and Howley, 2004] Madden, M.G. & Howley, T., 2004. Transfer of Experience between Reinforcement Learning Environments with Progressive Difficulty. *Artificial Intelligence Review*, 21(3-4), 375-398. Available at: http://www2.it.nuigalway.ie/m_madden/profile/pubs/ai_review_04.pdf.

[Mitchell, 1997] Mitchell, T.M., 1997. *Machine Learning*. New York, NY: McGraw-Hill.

[Ramon et al., 2007] Ramon, J., Driessens, K. & Croonenborghs, T., 2007. Transfer Learning in Reinforcement Learning Problems Through Partial Policy Recycling. In *Proceedings of the 18th European Conference on Machine Learning*. pp. 699-707. Available at: http://www.cs.utexas.edu/~lilyanam/TL/ramon_driessens_croonenborghs.pdf.

[Ring, 1997] Ring, M.B., 1997. CHILD: A First Step Towards Continual Learning. *Machine Learning*, 28(1), 77-104. Available at: http://www.springerlink.com.ezproxy.library.wwu.edu/content/v62227364485314w/fulltext.pdf.

[Silver et al., 2005] Silver, D. et al., 2005. NIPS workshop on "Inductive Transfer: 10 Years Later".

[Sutton and Barto, 1998] Sutton, R.S. & Barto, A.G., 1998. *Reinforcement Learning*. Cambridge, MA: MIT Press.

[Taylor, 2008] Taylor, M.E., 2008. Autonomous Inter-Task Transfer in Reinforcement Learning Domains. Available at: http://www.cs.utexas.edu/~mtaylor/Publications/Thesis-taylor.pdf.

[Taylor et al., 2008a] Taylor, M.E., Fern, A., Driessens, K., Stone, P., Maclin, R., & Shavlik, J., 2008. AAAI workshop on "Transfer Learning for Complex Tasks". Available at: http://www.cs.utexas.edu/~mtaylor/AAAI08TL/index.htm.

[Taylor et al., 2008b] Taylor, M.E., Jong, N.K. & Stone, P., 2008. Transferring Instances for Model-Based Reinforcement Learning. In *Proceedings of the European Conference on Machine Learning and Principles and Pratice of Knowledge Discovery in Databases (ECML PKDD)*. Antwerp, Belgium, pp. 488–505. Available at: http://www.cs.utexas.edu/~mtaylor/Publications/ECML08-taylor.pdf.

[Taylor et al., 2008c] Taylor, M.E., Kuhlmann, G. & Stone, P., 2008. Autonomous Transfer for Reinforcement Learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Estoril, Portugal: International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 283-290. Available at: http://delivery.acm.org.ezproxy.library.wwu.edu/10.1145/1410000/1402427/p283-taylor.pdf.

[Taylor and Stone, 2007a] Taylor, M.E. & Stone, P., 2007. Cross-Domain Transfer for Reinforcement Learning. In *Proceedings of the 24th International Conference on Machine Learning*. Corvalis, OR: ACM, New York, NY, pp. 879-886. Available at: http://delivery.acm.org.ezproxy.library.wwu.edu/10.1145/1280000/1273607/p879-taylor.pdf.

[Taylor and Stone, 2007b] Taylor, M.E. & Stone, P., 2007. Representation Transfer for Reinforcement Learning. In *AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*. Arlington, VA: Association for the Advancement of Artificial Intelligence, pp. 1-8. Available at: http://www.cs.utexas.edu/~mtaylor/Publications/AAAI07-Symposium.pdf.

[Taylor et al., 2007a] Taylor, M.E., Stone, P. & Liu, Y., 2007. Transfer Learning via Inter-Task Mappings for Temporal Difference Learning. *Journal of Machine Learning Research*, 8, 2125-2167. Available at: http://www.cs.utexas.edu/~mtaylor/Publications/JMLR07-taylor.pdf.

[Taylor et al., 2007b] Taylor, M.E., Whiteson, S. & Stone, P., 2007. Transfer via Inter-Task Mappings in Policy Search Reinforcement Learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. Honolulu, HI: ACM, New York, NY, pp. 156-163. Available at: http://delivery.acm.org.ezproxy.library.wwu.edu/10.1145/1330000/1329170/a37-taylor.pdf.

[Thorndike and Woodworth, 1901] Thorndike, E. & Woodworth, R., 1901. The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review*, 8, 247–261. Available at: http://psychclassics.yorku.ca/Thorndike/Transfer/transfer1.htm.

[Torrey et al., 2008] Torrey, L. et al., 2008. Transfer in Reinforcement Learning via Markov Logic Networks. In *2008 AAAI Workshop on Transfer Learning*. Association for the Advancement of Artificial Intelligence. Available at: http://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.aaai08.pdf.

[Torrey et al., 2007] Torrey, L. et al., 2007. Relational Macros for Transfer in Reinforcement Learning. In *Proceedings of the 17th Conference on Inductive Logic Programming*. pp. 1-15. Available at: http://pages.cs.wisc.edu/~ltorrey/papers/torrey_ilp07.pdf.

[Torrey et al., 2006] Torrey, L. et al., 2006. Skill Acquisition via Transfer Learning and Advice Taking. In *Proceedings of the 17th European Conference on Machine Learning*. pp. 1-12. Available at: http://pages.cs.wisc.edu/~ltorrey/papers/torrey_ecml06.pdf.