Learning Tasks Faster using Spatio-Temporal Abstractions

(extended abstract of the MSc dissertation)

Miguel Fidalgo Martins
Departamento de Engenharia Informatica
Instituto Superior Tecnico

Advisor: Professor Manuel Fernando Cabido Peres Lopes

***Abstract*—Simple reinforcement learning algorithms, such as Q-learning, are great in learning how to solve problems while being provided minimal information and definition of the problem domain, which is powerful because it allows the use of a single algorithm to solve different problems. However simple approaches like Q-learning don't try to understand the problem nor the world they are in, they solve it by trying everything and learning/memorizing what to do in each step. This approach will rapidly show its flaw when trying to learn more complex tasks, where it might not be able to solve in useful time.**

In this work I present an approach that integrates reinforcement learning with Spatio-Temporal Abstraction, with the objective of allowing a faster learning of tasks specially as their complexity increases. This approach's focus is always in understanding the world by creating and updating a knowledge base that is then exploited and considered when making a decision. This approach autonomously creates, represents and learns knowledge of higher complexity and generalization than common reinforcement learning methods like Q-learning. I also present an implementation of this approach which I call Abstraction Agent, this agent being the result of the integration of a Q-learning agent with my approach.

Experimental results show that this approach increases the learning speed of the agent, reducing the total amount of steps necessary for the agent to consistently reach the goal and solve the task. This performance increase is also shown to remain even in worlds with no intuitive abstraction.

I. Introduction

Classic logic based A.I. methods, e.g. expert systems, are based on creating knowledge representations of domains/worlds in order to solve problems or perform tasks in them, but creating this representations of the knowledge domain is a very complex and time consuming task.

Modern methods like reinforcement learning take a dynamic and behavioral approach, trying to learn how to behave instead of basing it on a knowledge domain. In reinforcement learning instead of using a definition of the knowledge domain to know how to behave and decide, it tries to learn how to behave and make decisions by trial-and-error. However, by avoiding the definition of the knowledge domain these methods are more limited in terms of generalization and representation of complex behavior, hurting their effectiveness when learning more complex tasks.

This work's motivation is to try to integrate of the qualities of logical based systems into reinforcement learning and testing to discover if it allows for a faster learning of complex tasks, without the manual, complex and time consuming definition and representation of the knowledge domain that is necessary in logical based systems.

A. Background

Logical based systems have higher generalization capabilities then machine learning methods because information about the world and a model or representation of it are painstakingly manually coded into those systems. Some logical based systems have tools which use concepts such as inference to derive new information from the information given but, although this helps alleviate the need to explicitly explain the world, when the world or domain changes all that information is of no use.

Machine learning methods, such as reinforcement learning which is the focus of this work, have a higher adaptability then logical based systems because they rely on none to little previous knowledge of the domain/world to be able to learn what to do and find a solution, allowing the use of a single algorithm to solve different problems. A simple approach that is able to learn solutions from 0, is to simply learn what action to take in each state by randomly exploring the world, however this approach doesn't understand the world and as such isn't able to generalize what it learns, it has a very "shallow learning" similar to memorization. This approach is the best approach when the world is not very big or complex because exploring the world without stopping is faster then stopping to understand it, however as the world gets bigger and more complex it will gradually stop being efficient and eventually stop being effective.

A different approach is required, an approach where the agent autonomously tries to actually understand the world, an approach where the agent instead of trying to learn directly what action to do in a state, tries to understand the state, reasoning the implications of the values of a state and generalizing the state, performing a **state abstraction**. There are many ways to perform a state abstraction, some works create abstract states by ignoring specific state features and grouping the different states that match after the change, others create abstract states by grouping states "closer" to each other relying on state transition data. In summary, an abstract state is a generalization of states, i.e. it is a group of states that have something in common, e.g. the same

reaction/state change to an action or a similar location for behaviour/action decision purposes. It provides additional meaning to a state, the state stops being just itself, but itself and part of a group of states.

State abstraction is an essential step towards providing generalization capabilities to behavioral approaches.

In "The gardens of learning: a vision for AI" [1] the authors discuss not only what learning is and "shallow learning" or memorization, but they also discuss how learning should build upon itself in order to keep expanding the range of doable tasks, and one of the proposals to achieve this is the concept of adding macro-actions to the action-space to be able to learn and represent more complex behaviour. These macro-actions are simply a list of actions to be performed in order, they are the simplest forms of a temporally abstract action which are sometimes referred as skills. One work deeply related to this concept is the options framework [2], where an option is added to the action-space like in the gardens of AI proposal, but instead of being a simple macro-action it is a closed-loop control rule, i.e. instead of being a specific sequence of actions to perform, it is a new independent policy that at each time-step chooses an action to perform based on the state, which means it is responsive to state changes and has no specific number of actions and as such doesn't abstract a specific time interval.

Temporal abstraction, such as options, allow the representation of behaviour of higher complexity, which when learned accelerates the overall learning speed of complex tasks.

There are many factors that can affect the complexity of a task and how difficult it is to learn it, some of them are:

1) the dimension of the world or state-space;
2) the specificity of the behaviour required to achieve a goal or simply how difficult it is to reach a goal;
3) the scarcity of rewards or feedback given.

The impact of factor 1 and 2 can be mitigated by state abstraction and temporal abstraction respectively and the impact of factor 3 can be mitigated by using intrinsic motivation.

Intrinsic motivation provides an agent the capability of learning and acting independently of external feedback. Intrinsic motivation and it's integration with temporal abstraction has been discussed in works like [3] and it is even discussed with reinforcement learning in mind in "Intrinsically Motivated Reinforcement Learning"[4].

The approach we are looking for, an approach that provides a higher understanding of the world and generalization capabilities to reinforcement learning that could help it solve tasks faster and solve more complex tasks, should be a combination of **state abstraction**, **temporal abstraction** and **intrinsic motivation**.

In my approach the ultimate goal is the same but the focus is always in understanding the world and later exploit this knowledge. My approach merges some qualities of logical based systems with behavioral approaches by in fact a combination of state abstraction, temporal abstraction and intrinsic motivation.

B. Contributions

In this work I present my approach, implement a reinforcement learning agent that integrates this approach with a Q-learning agent, which I call "Abstraction Agent", and also evaluate this new agent as being able to learn complex tasks faster then the original Q-learning agent it was built upon.

The Abstraction Agent autonomously creates and updates a knowledge base constituted by both space and temporal abstractions. This knowledge base will represent the understanding the agent has of the world. This knowledge base will be composed of 2 separate abstraction systems: World Abstraction and Knowledge Abstraction.

1) World abstraction: This abstraction system will be the main focus of this work. It will exploit the locality aspect of states to create and continuously update an abstract model of the world as the agent explores and learns the existing model of the world. This abstract model of the world is similar to the actual model of the world, but it is an abstraction of it, where instead of states it has abstract states and instead of actions it has abstract actions. This abstract model is created independently of rewards, allowing it to abstract the world even when there is only a single reward signal, the reward for the goal.

2) Knowledge abstraction: This abstraction system will complement the approach. It consists of the use of a neural network to abstract a state using its features and then try to predict the immediate reward of each action. It does this by learning correlations between the feature values of a state and the reward received when performing an action. This allows the agent to generalize this knowledge between multiple states that share feature values. It is constructed based on the work in [5].

C. Document Organisation

In the section II I will discuss various concepts and components used in my work, and other works that served as a base or inspiration for them.

In the section III I will provide a description of both my approach and the implementation of the Abstraction Agent.

In the section IV I will evaluate the Abstraction Agent against the Q-learning agent it was build upon in order to evaluate the impact of my approach on the performance of the agent.

In the section V I will provide a quick summary of the work.

II. Related Work

In this section it's reviewed various concepts used in my work and various works related to them.

A. Integration

The motivation of this work is integrating the qualities of logical based systems into reinforcement learning. The approach taken was basically: make a reinforcement learning agent that is able to autonomously create its own knowledge base of the world from its observations. So the problem to solve or the objective is in creating an agent that autonomously creates its own knowledge base of the world.

A world is constituted by states and actions, and as such the knowledge base of the world contains, hopefully generalized, information of the states and actions in the world. This additional information of the world, the one that constitutes the knowledge base, is obtained or represented by performing what is usually called an abstraction of the world and as such, because the world is constituted by states and actions, there are 2 types of abstraction: State Abstraction and Temporal Abstraction. Both these abstractions are essential to the solution I propose in this work.

Another essential concept in this work is intrinsic motivation. Many complex tasks are complex because they are scarce on reward signals. The agent is expected to create the knowledge base using intrinsic motivation, and then exploit it for extrinsic motivations.

There are many works about both State and Temporal abstraction and also intrinsic motivation. Some of them have approaches similar to this one, combining these concepts and even creating their own ”knowledge base”. I will now discuss this concepts in more detail, noting work related to them and their impact in my work.

B. Temporal Abstraction

Temporal Abstraction consists in the creation of abstract actions (groups of actions) which represent behaviour that takes undetermined steps to end (which is why its called ”Temporal” abstraction) and is composed by the original or ”primitive” actions which always take a single step to end.

The simplest form of temporal abstraction is a macro-action, which is simply a list of actions to be performed in order. One work deeply related to temporal abstraction is the options framework [2], where an option is an abstract action and is added to the action-space like in the gardens of learning [1] proposal, but instead of being a simple macro-action, it is a closed-loop control rule, as refered before.

”Intrinsically Motivated Reinforcement Learning”[4] is a work that integrates and discusses the use of options with reinforcement learning in more detail and also integrates the use of intrinsic motivation. In [4] the agent finds ”salient events” and creates options whose goal is to reach or create this ”salient events”. The definition of what constitutes a ”salient event” is manual and dependent on the world, and as such not a good fit for the approach we are looking for, for the Abstraction Agent.

The Abstraction Agent also uses options, but uses them in a way different from the options framework, some examples are:

- it integrates them with the concept of abstract state;
- it doesn't actually add them to the action-space, and as such;
- the option's policy only contains ”primitive” actions.

C. State Abstraction

State Abstraction consists in the creation of abstract states (group of states) which contain information relevant to the original or ”primitive” states it's constituted by.

A temporal abstraction, or option or sometimes called skill, exists for a reason, it has a goal, its policy representing the behaviour to achieve it. However finding those goals autonomously has been the discussion of many works.

Q-cut[6] and L-cut[7] use graph theory to analyse the agent's state transitions and find bottleneck states, this bottleneck states being the ”border states” of strongly connected areas that would disconnect the graph if removed. Creating, learning and using options whose objective is to reach the bottleneck states allows an agent to search and navigate the state space more efficiently. Q-cut analyses the global state-space transitions known while L-cut only stores and analysis a local subset of the global state-space transitions known.

”Learning Purposeful Behaviour in the Absence of Rewards”[8] also analyses the agent's state transitions to find correlated features (features whose values change together), it then clusters this feature changes into a ”purpose” or goal and creates an option whose goal is to perform the purpose. In sum, the algorithm creates an abstract state from correlated feature changes and creates an option to enter or leave that abstract state.

When this works divide a graph or state-space into 2 they are performing a state abstraction which creates 2 abstract states, however this works perform state abstractions with the sole purpose of finding good goals for the options, so some of them don't even store the abstract states found. The Abstraction Agent, however, stores and integrates the abstract states with the notion of options to perform spatio-temporal abstractions of the world, the abstractions that compose the abstract model of the world that is the core of the ”world abstraction” component.

”Human-level control through deep reinforcement learning”[5] is a work which consists on the creation and usage of a neural network to learn and predict the Q-values of a state, a network which they call Q-network. This work and its integration of neural networks with reinforcement learning were the base for the creation of the ”Knowledge Abstraction” component of the Abstraction Agent.

D. Spatio-Temporal Abstraction

Spatio-temporal Abstractions are an integration between space and temporal abstractions.

In HEXQ[9] the underlying model of the world is abstracted into separate layers or abstract states and the agent learns policies or temporal abstractions to navigate through this layers. HEXQ abstracts the state-space of a single feature into an 'abstract state-space layer', from now on just 'layer', abstracting each single feature one at the time. The maximum number of layers is thus equal to the number of

features. HEXQ seems to be a good approach only in specific worlds that don't have many "exits", where the states are already constructed in a format where it's features already have a hierarchy (like s = (room position, room number, house number)).

"Hierarchical Reinforcement Learning using Spatio-Temporal Abstractions and Deep Neural Networks"[10] is a work that creates an abstraction structure and spatio-temporal abstractions similar to the Abstraction Agent. In this work[10] the agent stores it's state transitions into a transition matrix (a graph), uses spectral clustering to find suitable state abstractions of the graph and then creates temporal abstractions or policies to go from one abstract state to another one. The resulted abstraction should be somewhat similar to HEXQ[9] without the need for a built in hierarchy in the state's features.

The idea, present in both works, of abstracting the state-space into abstract states and then creating abstract actions to navigate from one abstract space to another was a clear idea on how to integrate state and temporal abstractions and thus create spatio-temporal abstractions and their representation.

The Abstraction Agent also creates abstract states and then abstract actions to navigate from one abstract state to another one.

E. Intrinsic Motivation

Intrinsic motivations are motivations that lead the agent into performing actions that might not be aligned with their extrinsic value, i.e. it leads the agent into performing actions that the agent doesn't consider the best action to reach the goal. This can be helpful because performing a not "optimal action" now, and thus reaching the goal later, might provide information and experience that allows the agent to reach the goal even faster in future.

In "Intrinsically motivated learning of hierarchical collections of skills"[3] it's said that it's not concrete in the psychology literature what drives intrinsic motivation but it should probably involve novelty, surprise, incongruity and complexity. As is said in this paper, 'It (surprise) cannot account for what makes many forms of exploration and manipulation "interesting", thus other factors like novelty, incongruity and complexity might be necessary to be taken into account when learning on bigger and more complex worlds'.

This was taken into account when creating the Abstraction Agent and their impact is:

- when considering an action, the agent overvalues the option's value by pretending that the agent can always reach the option's goal in 1 step - this represents "complexity", it allows the agent to start choosing complex actions sooner
- when starting an option, the agent doesn't stop following the option's policy even when it's bad and contradicts the global extrinsic policy - this represents novelty, it allows the agent to follow a new option's policy and improve it faster even if it delays reaching the goal
- when discovering a bridge (transition between 2 abstract states) it has never seen before, the agent will sometimes recheck if the abstraction is "good" - this represents incongruity, it allows the agent to reunite abstract states that are later found to be connected or that should have never been separated.

III. Learning Tasks Faster using Spatio-Temporal Abstractions - Abstraction Agent

In this section I present my approach which I called the "Abstraction Agent", an agent that is able to learn complex tasks faster (in fewer steps) by integrating state abstraction, temporal abstraction and intrinsic motivation with Q-learning.

In reinforcement learning methods the ultimate goal is for the agent to learn how to get as much reward as possible, with simple methods like Q-learning going for the straightforward approach of learning/memorizing what to do in each step. In my approach the ultimate goal is the same but the focus is always in understanding the world, creating a knowledge base that is then exploited and considered when making a decision.

The most important concepts are:

- **State Abstraction** - Not every state is completely different from one another, being possible to generalize knowledge or a behaviour to multiple states.
- **Temporal Abstraction** - It doesn't need to exist only an action that takes 1 step and a policy to reach the goal, there can be actions that take multiple steps, these being policies to intrinsic goals or sub-policies (in this work these are called "options", but other names include temporal abstraction and abstract action).

The abstraction agent will be composed of mainly 3 parts:

- **World Abstraction** - Abstracts the state space into multiple abstract states. This abstract states are separated by bottleneck states. Options are created with this bottleneck states as intrinsic goals. By separating the state space into smaller abstract states and creating policies/options to go from one abstract state to another, the agent learns to navigate the state space independently of extrinsic rewards and explore the state-space more efficiently.
- **Knowledge Abstraction** - Uses a neural network to abstract the state, separating a state into its features and then providing them independently to the neural network as input. The neural network will predict the reward of each action based of the individual features and the relation between them instead of basing it on the state as a whole. This allows the neural network to learn about relations between individual feature values and the action's reward, knowledge that can be generalized to multiple states that share those individual feature values, allowing it to estimate the reward of each action in states it has never seen before.

- **Integration** - the core of the agent where it decides which action to take and learns. This part consists of the usual Q-learning implementation with additional considerations in order to integrate it with the world and knowledge abstraction.

A. World Abstraction

This component integrates both state and temporal abstraction, creating an abstract representation of the state space that allows it to learn how to better navigate the state space independently of extrinsic rewards and explore the state-space more efficiently, thus learning tasks faster.

In this component's spatio-temporal abstraction representation, abstract states are divisions of space (or subsets of S) separated by bottleneck states, and options always belong to an abstract state and are policies on how to go from the abstract state they belong to into a neighboring abstract state, through the bottleneck states that separate them. Abstract states are basically containers of options that are useful for traversing the state-space through its bottleneck states.

B. Options - temporal abstraction

In normal reinforcement learning approaches, such as Q-learning, the agent is trying to learn a policy to maximize the rewards it receives. If we give a task for the agent to learn in the way of a reward, it will learn a policy that says what to do to in each state to get that reward. However, it's hard to learn complex behaviour directly because the agent needs to perform the complex behaviour at least once to receive the reward, and even when it does, it takes time to converge into a desirable policy.

A better method to learn how to achieve a complex goal (or solve a complex problem) is to split it into smaller pieces. The abstraction agent learns multiple policies to achieve subgoals. This policies are **temporal abstractions or options** which could be sub-policies of the desired global policy and in turn accelerate learning of complex behaviour. These options also allow the agent to instead of just considering actions that take him 1 step away and whose value might be fairly similar, consider following a policy that will take him multiple steps away to someplace in the state space that can be more valuable, which allows the agent to explore the state space more efficiently by being able to navigate to more valuable places quicker.

The temporal abstraction or options used in the agent are based on the options or option framework presented on this works [2], [4]. Formally an option on the options framework is a triple o = (I, p, T) :

- I : initiation set : the set of states in which the option can be chosen or started
- p : is the policy followed while the option is being followed
- T : termination conditions: declares when the the option ends

In my approach some changes are made to the options that simplify and differentiate them and their use from the traditional options from the works[2], [4]. Some differences are:

- My options always belong to an abstract state and can be started anywhere on that abstract state but only on that abstract state. Therefore my options are only a pair o = (p,g), p being the policy and g the goal states of the option. This is because the I: initiation set is not only already implicit but somewhat static, the initiation set doesn't grow like the usual options, it is always equal to the set of states that constitute the abstract state and only grows as a side effect of the abstract state growing;
- My options don't have an option model, they don't have a transition probability model nor a reward model. The value of an option is updated similarly to a normal action when the option ends;
- My options learn and store the extrinsic value of each bridge transition so the agent can make a choice between multiple bridges if necessary;

The problem is finding useful intrinsic goals or subgoals to perform the desired task. We humans usually know the goal we want to achieve when we create subgoals, but the agent doesn't. As such, the agent will need to autonomously discover useful subgoals based on its observations and with no knowledge of the goal.

1) Segmented S-cut - state-space abstraction and subgoal discovery: The state space abstraction and sub-goal discovery, or Segmented S-cut, is based on the Segmented Q-cut [6] with inspiration on other works [7], [10].

The agent autonomously discovers useful subgoals based on its observations and with no knowledge of the goal. It does this by saving the state transitions of the agent in the form of a graph and then using spectral clustering to find the 2 clusters that best divide the graph, i.e. to find the best cut of the graph. Each cluster will be an abstract state. The state transitions from one cluster/abstract state to another, or bridges of the graph, will be an approximate equivalent to the bridges that would be found in a Min-Cut/Max-Flow of the graph, and if a cut is "good" they will provide good bottleneck states to be used as subgoals.

The algorithm initializes with 1 abstract state that contains only the first state. When the agent finds a state that has never seen before (that doesn't belong to any abstract state) he attributes that state to the abstract state that contains the previous state. After performing a spectral clustering on the graph corresponding to that abstract state, if a "good" cut is found, the abstract state and it's state transition graph will be divided into 2 abstract states, each with its own graph, and 2 options will be created using the bottleneck states as intrinsic goals (subgoal of the global policy), this options being policies to go from one abstract state to the other and vice-versa.

The calculation of the quality of a cut or how "good" it is, is similar to the one in Q-cut[6], i.e.

$$\frac{(Nr\_of\_states\_in\_1st\_cluster)*(Nr\_of\_states\_in\_2nd\_cluster)}{Nr\_of\_bridges}$$

The only difference is that in the Abstraction Agent, the $Nr\_of\_Bridges$ is equal to the number of bridges in the

cut **plus** the number of bridges (to other abstract states) that the abstract state being cut already had. This will allow the agent to consider the connectivity of the abstract state being cut to other abstract states in the calculation of the quality of the cut, not cutting abstract states whose cut quality might only be significant in the local spectrum.

I will call S-cut the process of dividing an abstract state into 2 using spectral clustering and creating the corresponding options.

This process will repeat itself recursively, i.e. if S-cut finds a good cut and splits an abstract state into 2, an S-cut will also be performed in each of the new 2 abstract states, and in being able to find good cuts those 2 new abstract states will again be each split into 2 and an S-cut will be performed on them. This recursion ends when an S-cut was performed on all abstract states and no more good cuts were found. This recursion is equivalent to the one in Segmented Q-cut in [6], where each segment in Segmented Q-cut will correspond to an abstract state in Segmented S-cut.

The differences between Segmented S-cut and Segmented Q-cut are:

- Segmented S-cut uses spectral clustering to find a good cut, which doesn't need a source state nor a target state like in Segmented Q-cut;
- For the spectral clustering algorithm, the arcs in the graph have a fixed capacity of 1 or no capacity. This is because it's beneficial to consider arcs equal to one another instead of rating them on some measure (like frequency) that is normally exploration based and doesn't necessarily hold any information about the underlying state transition structure that can be helpful in abstracting it;
- The state transition's frequency are saved and used when calculating if a cut is good. Regardless of the cut's quality, there are 2 conditions that need to be met in order for the S-cut to proceed (in the following examples the S-cut is trying to split an abstract state into abstract state A and abstract state B):
    1) There needs to be at least 1 bridge from abstract state A to B and another from abstract state B to A. This exists so a cut can't happen without the knowledge that both abstract states can reach the other.
    2) All bridges from A to B **or** from B to A need to have a certain minimum frequency value. This exists to make sure that the cut is real and the abstraction good, and not just an illusion created by a lack of exploration. A minimum frequency value of 3 is used because the lower the value the sooner the abstractions are made, which is beneficial, and values above 3 show the same results.
- With each cut only 2 options are created, one to go from one abstract state to the other and vice-versa. The way the goals are attributed to options is explained in further detail in the implementation part of the dissertation.

2) Intrinsic Motivation - incongruity: The goal of the world abstraction is to create and update a spatio-temporal abstraction representation of the state space, and the sooner the representation evolves, this is, the sooner the world is partitioned into abstract states and options, the faster its benefits will be of use to the Abstraction Agent, and as such the faster the Abstraction Agent will learn the task.

It's then important that the agent abstracts the world as soon as possible, however the faster it tries to abstract the world the worst the abstractions will be. It's a trade off between abstraction speed and the abstraction's quality.

This is where incongruity comes in, when the agent finds a new bridge between 2 abstract states, whether its between abstract states that already had bridges between them or discovering that 2 abstract states are actually connected, it will reassess the quality of the cut between those 2 abstract states, reuniting them if the quality is no longer sufficient. This allows the agent to abstract the world as fast as possible without lowering the abstractions quality, because when and if bad abstractions happen they can always be fixed later.

C. Knowledge Abstraction

As said above, not every state is completely different from one another, being possible to generalize a behaviour to multiple states.

To do this the agent constantly updates a neural network with observations, the neural network trying to estimate the immediate reward of performing an action on a state. More specifically, the user manually classifies the features that it wants the neural network to pay attention to and their feature size (number of different values the feature can have). Generally the user will tell the neural network to pay attention to all features that are not continuous and each of those feature's size. This is a sort of manual pre-abstraction that is common to all states.

After this pre-abstraction the state is fed as input to the neural network, which will then try to find suitable abstractions and correlations from the feature value's input in order to justify the immediate reward it observed. This allows the agent to be able to generalize behaviour to different states that have similar individual feature value's, including states it has never seen before. Example: the state is made of 4 features (x,y,z, HasLavaInFront). The objective is for the knowledge abstraction to learn that moving forward is bad when HasLavaInFront is true, regardless of x,y,z.

This neural network integration with reinforcement learning is similar to the one on the work[5], with some differences being:

- The use of a normal neural network is used, in comparison with the convolutional neural network used in [5], because there is no "locality" present in the features, i.e. the states are not image pixel information whose feature distance in the state is relevant
- The agent creates 2 databases of observations and updates them every step. Every X steps the agent will update the neural network using the timesteps saved in this databases. One database stores timesteps

whose reward is (approximately) 0 and the other one stores all other timesteps. This separation exists so the timesteps with reward don't become diluted in the much larger number of timesteps that have no reward signal, allowing the agent to keep updating the neural network with useful timesteps that have a reward signal without overfitting. Other details are explained in the implementation part of the dissertation.
- This neural network will predict only the immediate reward of an action, not the Q-values which take future action's Q-value into consideration.

D. Integration

This integration component is the core of the agent, where the base Q-learning will integrate the options from the world abstraction and the immediate reward estimates from the knowledge abstraction, where the inner mechanics of how the agent chooses an action and learns are. The integration of the options with Q-learning is based on the work "Intrinsically motivated reinforcement learning"[4] with some differences, a major one being:

- Options aren't added to the action space, and as such the action space is fixed and all policies only consider the actions in the action space. Options aren't considered in any policy. Learning an option's value and considerations for choosing an option (when an action needs to be chosen) are done separately. This is explained in further detail in the implementation part of the dissertation.

A simplified overview of how the agent chooses an action:

- With epsilon chance it chooses a random action - end, otherwise
- If an option is active
  - Check if the option is still available in the current state
    - If not available, deactivate the option
    - If available, choose the option's highest value action - end
- If no option is active
  - Find the action with the best value
  - Check the options available, if the highest valued option has higher value than the action found
    - That option becomes active and the option's highest valued action is chosen (an option has it's own policy) - end
  - Else
    - The previously found action is chosen. - end

("- end" signifies the action step ends, returning the action chosen).

A pseudo-code of this action step is showed in the dissertation. The value of an action, regardless of the policy it's present on, is an addition of the learned action-value or Q-value and an estimation of the immediate action reward given by the knowledge abstraction.

The way the agent learns is:

- Timestep update is received (timestep update is (state, action, next-state, reward, episode-ended);
- Timestep update is sent to both world and knowledge abstraction components;
- All options available in that state use the timestep update to update their policy;
  - If an option has reached it's goal, and therefore ended, its option value (in the global spectrum) is updated.
- The Q-learning global policy uses the timestep to update its policy.

IV. Evaluation

I now present the parameters and systems used to evaluate the agent and then discuss the results that came from the Abstractions Agent's evaluation. Most demonstrations and graphs are confined only to the dissertation for space saving reasons.

A. Domain

This agent is implemented for performing a task in a world with discrete actions, discrete states and an episodic setting. A task is characterized by a number of specific states, the goals, which the agent needs to achieve in order to perform the task. An episode ends after a task is completed (the goal was reached) or after a certain timeout has been reached. The timeout generally used is 1.000.000 timesteps or 20 seconds.

The world doesn't need to be deterministic, for optimal performance a learning rate of 1 is used for deterministic worlds and 0.01 or 0.1 for stochastic worlds.

The discount factor can be between]0,1[but it needs to be high enough that the reward for achieving the goal can propagate to the entire state space. The abstraction agent is meant for complex tasks that sometimes require more than 1000 steps to reach the goal, so a discount factor of 0.999 is used.

B. Performance evaluation graphs and parameters

Most graphs to demonstrate and compare the performance of different agents are in the format:

- reward - cumulative rewards the agent receives in a episode (y-axis);
- timesteps - cumulative timesteps of the agent over the entire run, or, episodes - the episode count (x-axis).

Parameters used in the worlds unless otherwise specified:

- epsilon = 0.1
- discount factor = 0.999
- learning rate = 1
- $nrEpisodesForError$ = 200
- Graphs are an average of 10 runs
- Timeout = 1.000.000 timesteps or 20 seconds.

C. Worlds and maps

3 worlds are used for the evaluation of the agent in the dissertation, however only the most important one is present here: a toy world with different maps mostly composed of rooms.

This toy world has the agent travel through a 2 dimensional world and try to reach a goal position in the world. The states are s = (position x, position y, orientation, spaceInFront), e.g. s = (6, 56, up, "Lava") means the agent is in the coordinates (6,56) in the world, is facing up (or north) and has a lava pit in front of him. The action space is composed of only 3 actions: move forward, turn right and turn left. A position (x,y) in the map can be either: an empty space where the agent can be, a wall or obstacle, or a lava pit. The agent's spawn/source position and goal position are special empty places with those properties. Each position in the map has a correspondent "spaceInFront" feature value of: empty space - " ", wall - "X", lava - "L", source - "S" and goal - "G". There exists many maps with different layouts to test different principles but most of the maps are composed of rooms.

D. Toy world evaluation

Lets now explore and see how the world abstraction and the knowledge abstraction affect the agent's performance. The abstraction agent's core is also Q-learning. The abstract agent's world and knowledge abstraction can be individually deactivated, and if both are deactivated then only the Q-learning core remains, thus making the abstraction agent equal to a simple Q-learning agent. As such a simple Q-learning agent was used as baseline for comparison in order to evaluate the impact of the components.

1) World abstraction: As explained previously, the world abstraction abstracts the world into abstract states separated by bottleneck states, and options are policies on how to go from the abstract state they belong into a neighboring abstract state through the bottleneck states that separate them.

As such, optimal maps to show the world abstraction's strength and usefulness are maps constituted of rooms, where the agent needs to go through several bottlenecks to achieve the goal.

It is expected and demonstrated in 1 that in such maps the abstraction agent is able to learn faster how to reach the goal then the Q-learning agent.

A visual representation of the abstraction done in the world can be seen in Fig.1.

The world abstraction and its options allow the agent to learn faster by improving its exploration and allowing a faster "policy to goal" convergence. By having an option that allows the agent to go to a neighboring abstract state the agent has always available an action which can take it multiple steps away from where it is, rapidly allowing it to explore higher valuable localizations (which also contain other options), and because an option value is also affected by the value of other available options, then the value of an option represents the value of its abstract state and the discounted value of its neighboring abstract states, similar to how a state is valued in Q-learning. This is demonstrated in 1 by looking at the frequency maps of both agents in the same map.

Another benefit of the world abstraction is that the agent is learning options/sub-policies independently of extrinsic rewards and it's updating and improving them all at the same time (in the same episode). In a way, the abstraction agent instead of trying to learn the whole global policy is learning multiple sub-policies at the same time and then joining them.

The improvement in the exploration of the agent and the "simultaneous solving of sub-problems" benefit of the world abstraction allows it to actually improve the learning speed of the abstraction agent even when the map has no rooms or no apparent good abstractions. What this means is that the improvement provided by the world abstraction is affected by the map, but even in maps with no apparent good abstractions the state-space is still abstracted and the improvement in the learning speed still exists.

This is also demonstrated by using an "open room" map, where there is no "rooms" and no bottleneck states, and still the world abstraction abstracts the state-space and improves the learning speed of the abstraction agent.

E. Knowledge Abstraction

The knowledge abstraction tries to find extra meaning in the features of a state from the agent's observations. More concretely the knowledge abstraction (or its neural network) tries to predict the reward the agent will receive for each action using the state as input. In order to evaluate the knowledge abstraction lava pits were added to the 24-rooms map. The map has 2 versions, one where the lava kills (a negative reward is given and the episode immediately ends) and one where it doesn't kill (just a negative reward is given). The input states are s = (position x, position y, orientation, spaceInFront).

The knowledge abstraction is able to abstract the states, learn and then predict that when in a state with the feature spaceInFront = "L" the action "MoveForward" gives a negative reward. By doing this it is able to generalize this knowledge to all states that have the feature spaceInFront = "L", even in states it has never seen or visited. The Q-learning agent has to first fall into a lava pit to learn that in that specific state the action "MoveForward" is bad. As such the Q-learning agent will fall into every lava pit it has never seen, while the Abstraction Agent after learning that generalized knowledge will not fall into lava pits anymore.

F. World and Knowledge Abstraction

A joint evaluation of both components working together was performed. The evaluation can be seen in demonstrated in 1 and it shows that the Abstraction Agent with both components active is able to learn faster then the Q-learning agent. However a flaw was found on the world abstraction component. This flaw affects the agent when the maps

contain negative reward signals (such as lava pits), it creates instabilities and sometimes, depending on the random exploration provided by e-greddy, can even prevent the agent from reaching an optimal solution or even converge into a solution.

G. World Abstraction flaw

There can be seen some instabilities at the end (cumulative reward) result of the policy from the Abstraction Agent in the maps where the lava doesn't "kill". This instabilities were not expected when performing this evaluation and they were a warning that allowed for a flaw to be discovered upon further analysis. This instabilities are created by the option's policy grading. In summary, because an option's policy is independent of the extrinsic goal, the objective of an option's policy will be to lead the agent to **any goal** of that option while maximizing intrinsic reward, **regardless** of the extrinsic value of that goal. This usually results in leading the agent to the closest goal of that option even if that goal is worse (in the global spectrum) in relation to others. This affects the agent's performance in worlds with negative rewards (like the maps with lava that were used), where the choice of intrinsic goal of an option might lead the agent to a part of the state-space where it has to receive a negative reward in order to reach the goal.

V. Conclusions

The objective of this work is to integrate the qualities of logical based systems into reinforcement learning so that it's able to learn more complex tasks and faster without a manual, complex and time consuming definition and representation of the knowledge domain that is necessary in logical based systems.

In the introduction I it's discussed that the approach we are looking for, to perform the objective mentioned above, should be a combination of **state abstraction**, **temporal abstraction** and **intrinsic motivation**.

In the related work II, multiple works are mentioned that already use this concepts and integrate them with reinforcement learning. Many parts of my approach are based on this works.

It is then, in section III, that I present my approach or ideas for an integration of reinforcement learning with the autonomous creation, representation and learning of knowledge of higher complexity and generalization then common reinforcement learning methods like Q-learning, by using the 3 concepts above. My approach is composed of 3 components: the world abstraction, the knowledge abstraction and its integration with reinforcement learning.

The world abstraction autonomously discovers bottleneck states by analysing state transition graphs. These bottlenecks are seen as potential subgoals. It abstracts the state space into multiple abstract states, which are separated by these bottleneck states, and creates temporal abstractions or options using this bottleneck states as intrinsic goals. The world abstraction thus integrates both state and temporal abstraction and intrinsic motivation to create an abstract representation of the underlying model of the world, an abstract model that allows it to learn and represent more complex behaviour on how to navigate the state space independently of extrinsic rewards. This abstract model allows it to explore the state-space more efficiently and thus learn tasks faster.

The knowledge abstraction uses a neural network to abstract the state, separating a state into its features and then providing them independently to the neural network as input. This neural network is able to learn knowledge that can be generalized to multiple states that share individual feature values, allowing it to estimate the reward of actions in states it has never seen before.

The integration component is where the world and knowledge abstraction are integrated with the learning process and decision making of reinforcement learning, specifically, where the base Q-learning will be integrated with the options from the world abstraction and the immediate reward estimates from the knowledge abstraction.

I implement this approach and thus create an agent which I call Abstraction Agent, this agent being the integration of a Q-learning agent with my approach.

The Abstraction Agent is compared with a Q-learning agent in order to evaluate the impact of my approach in the agent's performance.

Experimental results show that:

The world abstraction is able to autonomously create a useful abstract model of the world which contains learned representations of more complex behaviour on how to navigate in it, increasing the learning speed of the agent mainly by increasing the efficiency of its exploration. This performance increase is shown to remain in different worlds, even in worlds with no "rooms", i.e. with no intuitive abstraction.

The world abstraction is also shown to have a flaw, which is discussed in greater detail in the dissertation.

The knowledge abstraction is shown to able to learn generalized knowledge that allows the agent to predict the value of actions in states it has never been in, thus increasing its performance.

In summary, the experimental results conclude that my approach, that integrates spatial-temporal abstraction and intrinsic motivation with reinforcement learning, allowing an agent to autonomously create, represent and learn knowledge of higher complexity and generalization then common reinforcement learning methods like Q-learning, is able to increase the overall learning speed, reducing the total amount of steps necessary for the agent to consistently reach the goal and solve the task.

References

[1] O. G. Selfridge, "The gardens of learning: a vision for ai," *AI Magazine*, vol. 14, no. 2, p. 36, 1993.

[2] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

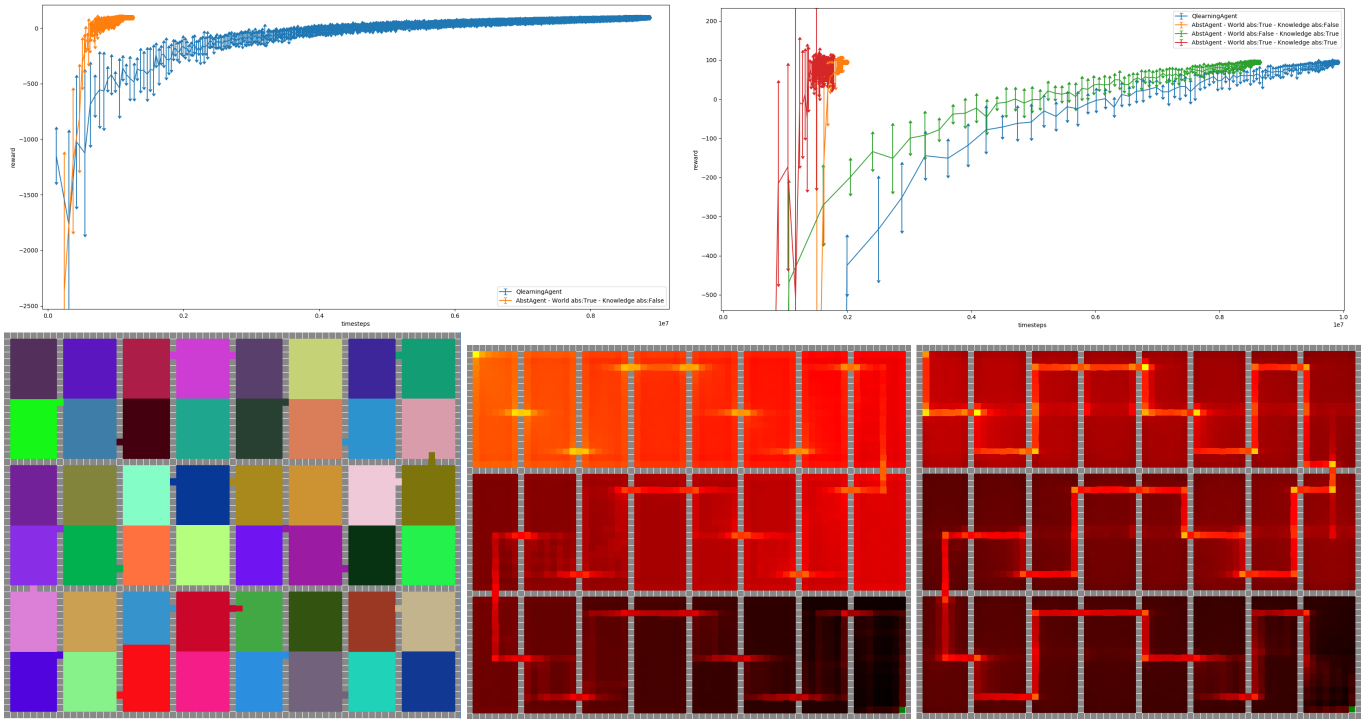

Figure 1. 1st image: Comparison between the Q-learning agent and the Abstraction Agent with world abstraction active in the 24-rooms map. 2nd image: Comparison between the Q-learning agent and the Abstraction Agent with both components active in the "Lava 24-rooms" map, that has lava that doesn't kill. 3rd image: Visual representation of the abstraction done over the world. 4th image: Comparison of the state frequency (Q-learning is left and Abstraction Agent is right). The brighter a position in the map the more it was visited.

[3] A. G. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *Proceedings of the 3rd International Conference on Development and Learning*, 2004, pp. 112–19.

[4] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2005, pp. 1281–1288.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[6] I. Menache, S. Mannor, and N. Shimkin, "Q-cut - dynamic discovery of sub-goals in reinforcement learning," in *European Conference on Machine Learning*. Springer, 2002, pp. 295–306.

[7] Ö. Şimşek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 816–823.

[8] M. C. Machado and M. Bowling, "Learning purposeful behaviour in the absence of rewards," *arXiv preprint arXiv:1605.07700*, 2016.

[9] B. Hengst, "Discovering hierarchy in reinforcement learning with hexq," in *ICML*, vol. 2, 2002, pp. 243–250.

[10] R. Krishnamurthy, A. S. Lakshminarayanan, P. Kumar, and B. Ravindran, "Hierarchical reinforcement learning using spatio-temporal abstractions and deep neural networks," 05 2016.