T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE4197 Engineering Project I

PROJECT SPECIFICATION DOCUMENT

November 16, 2022

Title of the Project

Autonomous acquisition of arbitrarily complex skills for continuous reinforcement learning domains

Group Members

150119824 - Zeynep Ferah Akkurt
150118042 - Bahadır Alacan
150119825 - Merve Rana Kızıl

Supervised by

Assoc. Prof. Borahan Tümer

[illegible]

1 Problem Statement

Reinforcement learning (RL) is a type of machine learning technique where an agent takes an action in an environment, moves to the next state, and receives the environment's feedback (reward or punishment) regarding that action. The learner is not told which actions to take but instead must discover which actions yield the most reward in the long run.
Using a regular RL algorithm such as Q-Learning can solve a problem in a reasonable amount of time provided if the environment is sufficiently small. However, as the state space and/or the action space of the environment grow, learning with a regular RL algorithm, can take too much time since the learner takes only one-step at a time without any model that can improve the learning speed. In this case, one possible solution is to employ *hierarchical* RL (HRL) [1]. In HRL, the environment is split into sub-environments with subgoals (the goal state of the split environment) and for each sub-environment one or more sub-policies, each consisting of a sequence of actions, also called *options* or *skills*, are learned.
Environment's regions/communities (sub-environments mentioned above) can be represented via graphs. To construct such a graph, there are community detection algorithms [2–4] that use agent's experience collected via interaction with environment. However, sometimes these algorithms may have problems with dividing the environment into regions as expected because of their sensibility to the input parameters. *Skill coupling (SC)* [5] is one way of solving such problems for discrete RL environments.
The focus of this project is on showing the behaviour of SC method in a continuous RL domain and how to speed up learning using options.

2 Problem Description and Motivation

Real-world problems mostly have a continuous state/action space. SC [5] is a method proposed only for discrete environments. The motivation of this project is to show the SC can also be an applicable to HRL problems with a continuous state space.
In HRL, the environment is split into sub-environments and for each of them, an option/skill is learned. This way the number of decisions is expected to be dramatically reduced as shown for discrete environments [5].
Environments can be effectively modeled using graphs. For instance, communities, a common practice in graph theory, may be considered as good candidates to model the

sub-environments in RL where a community is defined as a group of nodes that are closely linked to each other compared with those nodes that are not in the group. The communities should be detected so that the skills can be constructed from the model to speed up the learning process in HRL. Communities can be detected statically or dynamically using the graph model of the environment as an input. For this project, *Dynamic community detection* (DCD) methods [2–4] are more suitable to employ in revealing the sub-environments (and subgoals) in a partial model of an environment that the RL agent builds during its learning phase for planning purposes. The detection process is dynamic because the graph of the environment changes in each episode until the agent experiences most of the actions in a prevailing fraction of state space. Since the graph is not static, subgoals should be detected in a dynamic manner. However, the problem with DCD algorithms is that they are highly dependent on the parameters and sometimes a small change may cause an increase in the number of communities or their sizes. This leads to a problem referred to as *oversegmentation* (i.e., there might be several different representations of the same environment hence the complexity of learning may significantly change).

We will use the DCD method presented in [2]. Communities should be detected as the agent gathers experiences by interactions with the environment and constructs partial graphs. As the agent explores more, the partial graph grows more similar to the real environment. The beauty of detecting communities dynamically is because whenever the partial graph is updated by the agent there is no need to detect all communities from scratch. In DCD algorithms, only those communities affected by the addition of new nodes or edges are disbanded.

As mentioned above, the agent constructs a partial graph until it experiences most of the states and actions. However, continuous environments have an infinite number of states. Hence, the environment should be represented as if it is discrete in some way so that skills can be constructed and used for the SC [5] mentioned above.

Our aim is to use an environment which has a continuous state space, represent it as a discrete one using the connectivity graph (explained later in Section 6), use this graph for community detection and construct the skills and then implement the functionality of SC. Finally, we evaluate the performance of SC in continuous RL domains.

Conclusions drawn in this project will be essential since this project will extend the applicability of SC to continuous RL domains which further involves solving the oversegmentation problem that pulls down the learning performance of SC.

3 Main Goal and Objectives

The main goal of this project is to show the behaviour of SC proposed in [5] in a continuous RL Domain. To achieve this goal, objectives can be listed as follows:

- **Objective #1 - Representing the continuous RL domain:** Working with continuous state spaces is a challenging problem since the agent may never visit a state it has been previously at (i.e., the states are non-Markov). Hence, the agent may never discover the environment in a sufficient way to learn. Thus, environment should be represented, not only with its interactions but also with additional information that can be gained from the agent's interactions along with environment properties (i.e., Distance graphs etc. discussed in Section 6).

- **Objective #2 - Detecting subgoals/communities and constructing skills:** DCD algorithms use a graph based model of the environment which has a finite number of states and utilize this model to find communities with subgoals. After obtaining this graph based representation of the environment, any DCD algorithm can be employed to find communities and skills.

- **Objective #3 - Learning Skills and Improving skill learning process with Intra-Option Learning:** In continuous domains of HRL, since the usage of skills is to decrease the number of decisions that the agent takes, subpolicies of skills can be a problem if learned skills are not optimal. So, skills should be learned with an algorithm (i.e., function approximation etc.) that can find the subpolicies of each skill effectively.

  Learning skills along with primitive actions is essential for learning speed. Intra-option is one way to learn skills along with other consistent actions/skills. With this method, it is expected to be more efficient and faster. Because while a skill is chosen to be taken, other skills that are consistent with the chosen one will also be used to update the main policy.

- **Objective #4 - Evaluating Performance of Skill Coupling in Continuous Domains:** The behaviour of SC with intra-option learning in the Pinball Domain as a continuous environment will be evaluated. Results will be used to make a comparison with the work of [6] and [7].

4 Related Work

4.1 Autonomous acquisition of arbitrarily complex skills using locality based graph theoretic features: A syntactic approach to hierarchical reinforcement learning

This article focuses on the oversegmentation problem that Dynamic Community Detection (DCD) algorithms have in HRL. To overcome this problem, they introduced Autonomous Skill Acquisition and Coupling (ASKAC) methods (and Autonomous Skill Acquisition (ASKA) without SC) for discrete RL domains. ASKA is an HRL method with autonomous skill acquisition and it obtains skills with DynaMo [2] by using transition graphs that represent the partial model of the environment. ASKAC is the extension of ASKA with SC, a novel concept in their work. The model of the environment is created by the RL agent that interacts with the environment and constructs a partial model, where the partial model becomes more similar to the actual environment as learning proceeds. They use a dynamic community detection algorithm (Dynamic Community Detection by Incrementally Maximizing Modularity (DynaMo) presented in [2]) because the graph of the environment changes in each episode until agent experiences most of the actions and visits most of the states. Since the graph is not static, subgoals are also detected in a dynamic manner. However, as it mentioned, the problem with DCD algorithms is that they are highly dependent on the resolution parameter and sometimes a small change can cause an increase of the number of communities or their sizes. This leads to a problem referred to as oversegmentation where the algorithm mispartitions a subenvironment further into smaller components (i.e., there might be several different representations of the same environment hence the complexity of learning may significantly change). They introduced SC, a method to overcome oversegmentation problem that DCD algorithms have. They improve the robustness of DCD based HRL approaches against the resolution parameter with ASKAC by using SC. [5]

Our intended approach is based on the community detection and SC method as the approach used in this article. They work on discrete state space, however, our study will be on continuous state space.

4.2 Intra-Option Learning about Temporally Abstract Actions

This article explores option learning methods that do not require an option to execute until its termination or not executed at all. They call these methods intra-option learning methods which are examples of off-policy learning. It is stated that if the options are Markov, we can look inside the options and use a special temporal-difference approaches to learn about an option before it terminates. In SMDP methods [1] temporally extended actions (options) are treated as indivisible and unknown units like a black box. There is no attempt in SMDP theory to look inside options, to examine or modify their structure in terms of lower-level actions. Examining the lower-level actions (primitive actions such as right, left) can fasten the learning process since it allows the agent to discover more within a lower-level base while using an option's sub-policy.
When an option is chosen to be updated, all other options that are consistent with executed option are also updated. This way, options are learned faster compared to SMDP. [8]

To improve the learning process, we plan to use Intra-option [8] learning method that finds the sub-policies of options while learning on the big environment proceeds instead of using sub-environments. With this method, it is expected to be more efficient and faster.

4.3 Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining

In this article, the authors try to discover skills incrementally while the agent is interacting with the environment. They presented a method called Skill Chaining. The Skill Chaining approach tries to decompose a continuous problem into chains of skill and then uses an option framework to learn skills. The chaining procedure is done by defining each previous skill's initial states as goal states of current skill. This approach defines a goal skill which indicates how agent can reach the goal state when it arrives at a close proximity to the goal region. Then through the skill discovery process, the initiation set of current skill is considered as the target set of another skill, and a new option is defined to reach this target set. This procedure continues until the start state is reached, where the chain of skills is constructed. They demonstrate experimentally that skill chaining creates appropriate skills and achieves performance improvements in the Pinball domain. They also introduced a Pinball Domain which has a continuous state space. Our approach will be a graph-based skill acquisition approach in the building skill phase used in [6]. Comparisons between graph-based and skill chaining methods are presented in the work of [6]. We will use the same domain as they used for their proposed method. [7]

4.4 Graph-based skill acquisition and transfer Learning for continuous reinforcement learning domains

This article focuses on the curse of dimensionality problem of RL methods hence how they exceed the required learning time. They proposed an option learning framework to autonomously discover high-level skills in continuous reinforcement learning domains. They introduced two novel graph based skill acquisition method, named GSL, and a skill based transfer learning framework, named STL. Skills are discovered using an adaptive community detection algorithm from a connectivity graph which is a model that represents the agent's experience along with the environment's dynamics. The efficiency of these methods is dependent on the structure of the connectivity graph and also the algorithm that is used for community detection. Connectivity graph is a combination of transition and distance graph. They use two different distance graph named K-NN graph [9] and power-law graph [10]. They achieved the best performance by using the combination of transition graph and power-law based distance graph as connectivity graph structure with Louvain algorithm for community detection. In STL, collection of learned skills coming from the source task is used in the target task based on their fitness. They proposed two methods, named Trajectory based Fitness (TF) and Matching based Fitness (MF), to calculate the fitness of a learned skill. [6]
Our approach uses the same continuous RL domain which is the pinball domain used in this article. We will use the same approach that they used for community detection by creating a connectivity graph. However, even though they worked on continuous state space, they used a static community detection algorithm, in our approach community detection will be done dynamically. Our method does not include the transfer learning methods that are proposed in this article.

5 Scope of the Project

Our project is based on a domain which has continuous state space. A ready implementation of HRL and DCD algorithm will be reused. A light to moderate modification is planned for these two algorithms. Both algorithms will be used when the model of the environment is generated as a graph; then this graph will be presented as input to the DCD algorithm directly used in [5].

There is a framework called Pinball Domain. We will use this framework to set up the properties and functionalities of our continuous environment.

This project contains five parts:

1. Representation of the continuous RL domain as a graph with a finite number of nodes
2. State aggregation
3. Discovering skills using the graph generated in part 1 with DCD algorithm
4. Learning with Intra-option Learning method
5. Modifications on SC (skills are constructed in part 2).

We will entirely implement part 1. In part 3, we will use the source code of DynaMo [2] as a DCD algorithm with the input from part 1 to find communities and generate skills. We will implement a state aggregation method to discretize the continuous environment and use it for the learning process with intra-option learning which will also be implemented by us. A light to moderate modification is planned on SC. Modifications on SC are needed because of the change in the environment. This method is presented for only discrete domains. However, if the graph generated can be converted to a discrete environment and used for the SC, we will save a lot on the modifications required.

Influencing Factors (Assumptions) and Constraints

We assume the framework for Pinball Domain has no conflicts with the methods mentioned above.

The verification of the work, i.e., experimentation will most likely require a significant amount of memory space and execution time, we assume that the project will be developed with a computer that has a sufficient CPU and memory.

6 Methodology and Technical Approach

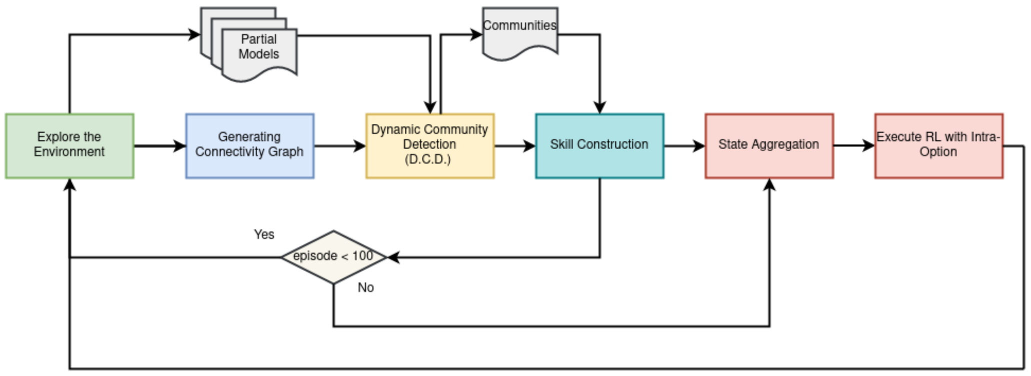


Figure 1: Initial Flow (without SC)

In this project, agent's learning environment is in a continuous state space (contains an infinite number of states) named Pinball Domain introduced in [7]. To be able to show the behaviour of SC described in section 4.1 in such an environment, firstly the continuous environment should be represented as some kind of graph with a finite number of nodes as if it is a discrete environment. This process needs to be done because community detection algorithms need a representation of the environment as a graph that has a finite number of nodes as input. After representation, communities can be detected with a DCD algorithm hence skills can be constructed. The next step after finding the skills is the learning process of how to use them. Finally, SC can be implemented for the continuous domain to examine its behaviour for the learning process.

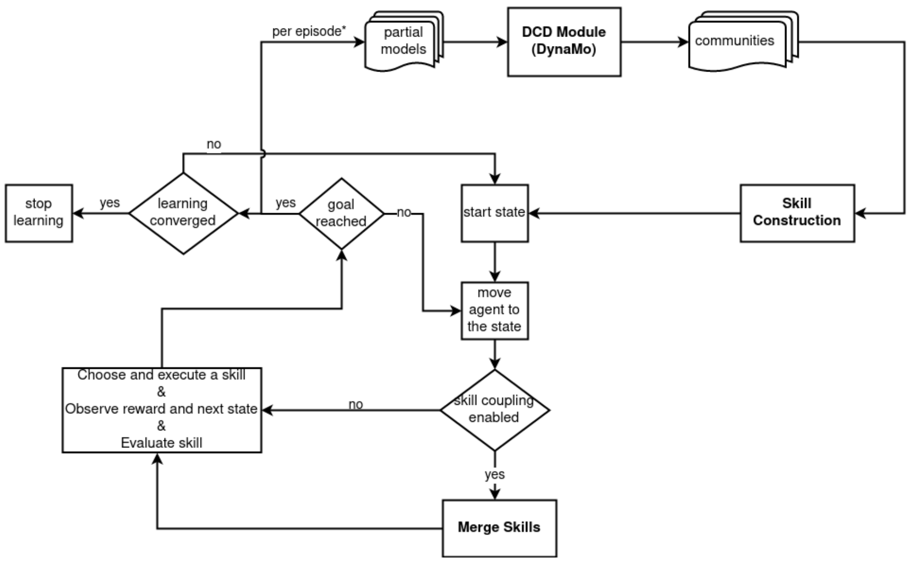


Figure 2: Flow diagram of learning process with SC modified from [5]

Before going any further, challenges of a domain that has continuous state space should be stated so that the methods we will plan to use can be understood: There are several difficulties that are absent or occur less in discrete domains that become important in continuous domains:

- Abstract State: In a continuous state space, the agent may never be at a state previously visited a second time. This is ascribed to the fact that since there is an infinite number of states, the probability of visiting each individual state at the next action of the agent is 0. States of such environments are *non-Markov*; hence, these environments cannot be represented by Markov chains as they are. Markov property assumes that decisions and values are to be a function only of the current state. This is not possible for a continuous state space. [11]. So, for non-Markov states of such an environment, it is a good approach to think the state at each time step as an approximation to a Markov state.

  With state aggregation methods (described in Section 6.4), a state can be represented as an *abstract/representative state* ϕ_n replaces all non-Markov states inside a region n. However, determining that region's size can significantly affect the representation. If the region is very small, then the discrete space of abstract states would be too big and there will be too many representative/abstract states. On the other hand, if the region is too large, then the abstract states of the environment amount to a small number, and the problem of discovering these states would be relatively easy. However, since the area that the abstract states represent in the environment is rather large, it is highly likely that the performance of learning would be low due to the individual distance (i.e., error) between the non-Markov state and its representative state.

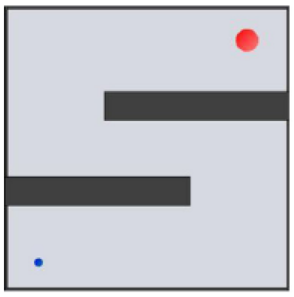
- Initiation sets/Subgals: In discrete domains without function approximation, a policy to reach a subgoal can always be represented exactly; in continuous domains, it may only be possible to represent such a policy locally [7]. This causes the problem of skill learning along with the main policy learning.

- Representation: In discrete domains, it is almost perfectly feasible to create a value table (i.e., Q-table) with a dimension of state vs skills/actions for each learned skill/action. In continuous domains with many variables, this is not possible. It requires other methods (function approximation etc.) to represent the skills or actions value functions and update them in each step.

- Characterization: Subgoals are the states that exist in the solution path to the real goal state in the main environment. In discrete domains, subgoals can be detected from the transition graph which is generated by the agent while visiting a finite number of states. By looking at this graph, the likelihood of a state to be a goal state that lies in the solution path to the real goal state can be found. However, in continuous domains, since there are infinitely many states, this process is not so easy and detecting subgoals is one of the challenging problems.

6.1 Pinball Domain

To provide a continuous test domain for RL algorithms, the Pinball domain can be used. Pinball is one of the most challenging environments for RL algorithms because of its dynamic aspect, sharp discontinuities, and extended dynamics control characteristics [6]. It has 4-dimensional continuous test domain which are x, y, x velocity, and y velocity. The goal is to manoeuvre the blue ball into the red hole. The state of the ball is represented with x, y, x', y' since the ball is dynamic [7]. Obstacles in the environment are elastic and this elasticity is useful for the ball to reach the red hole by bouncing. These obstacles can be used to speed up the learning process because of the bouncing effect. There are 5 primitive actions which are incrementing or decrementing x velocity or y velocity and, leaving the velocities unchanged [7].


(a) A simpler pinball domain

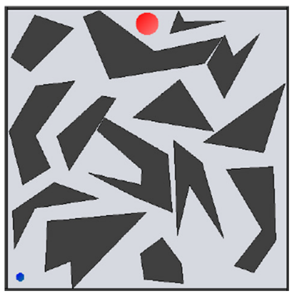

(b) A harder pinball domain

Figure 3: Pinball domain instances

[6]

We will use this domain to test the SC method. This domain is applicable for our problem because of its dynamics and features in the sense of HRL.

6.2 Generating Graphs

When an agent interacts with the environment, its experiences along with the problem domain properties are kept in a graph named Connectivity Graph (CG) [6]. This graph is the combination of two kinds of graph models named transition graph and distance graph. Transition graph shown in Figure 4 is built by the agent experiences via interaction with the environment. Edges are the ways of source state to target state and are weighted according to the probability of taking that action.

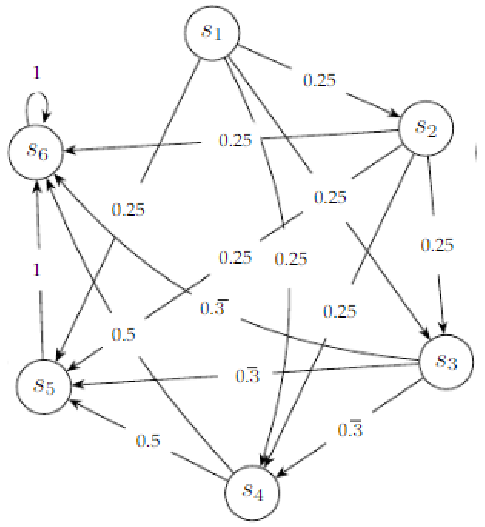


Figure 4: Example transition graph with six states

The problem domain properties will be represented with a distance graph. In the distance graph, there is a node for each visited state, and nodes are connected to each other based on their distance. As an example for distance graphs, K-NN [9] and Power-Law [10] graphs can be given.
Combination of transition graph and distance graph [6] is given below:

$$CG(w_{ij}) = \alpha TG(w_{ij}) + (1-\alpha)DG((w_{ij}) \tag{1}$$

The combination of these two graphs is called connectivity graph. This graph will be the input to DCD algorithm that finds the communities.

6.3 Dynamic Community Detection

Since the environment will be represented as a graph described in Section 6.2 and for sure the partial model that the agent constructs are graph, dynamic community detection algorithms are good candidates to detect subtasks and subgoals. It is dynamic because the graph of the environment changes in each episode until agent experiences most of the actions and visits most of the states. Since the graph is not static, subgoals should be detected in a dynamic manner.

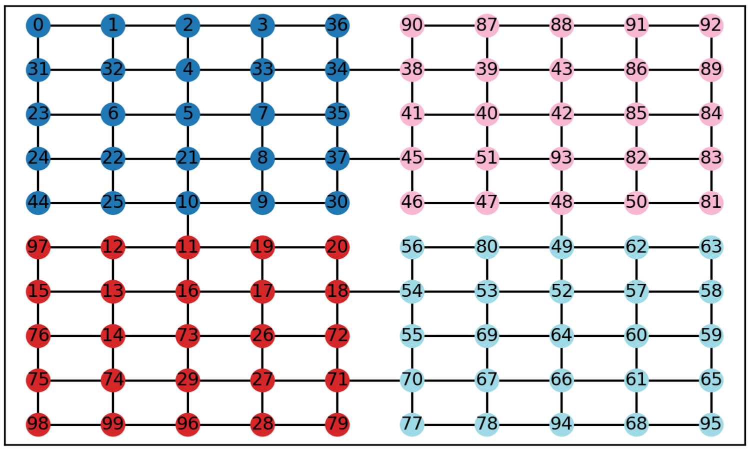


Figure 5: Community detection in 9x9 Grid World (one room) discrete environment with algorithm DynaMo.

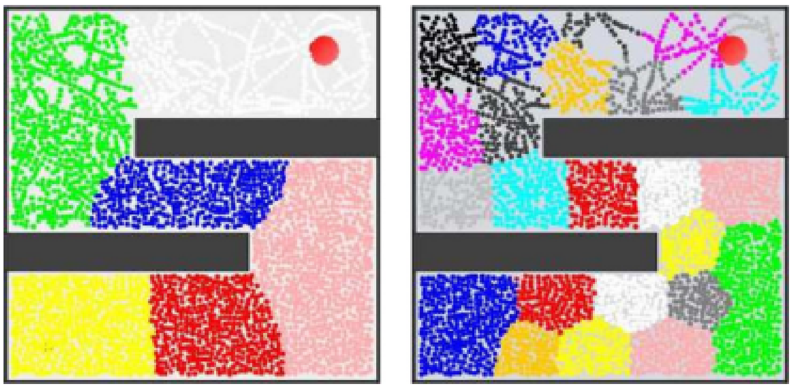


Figure 6: This figure is taken from the work in [6]. Detected communities in Figure 3a obtained by Graph based Skill Learning agent using Louvain [12] algorithm (left) and InfoMap [13] algorithm (right).

An example of community detection in a discrete environment is given above as Figure 5. For continuous environments, the detection of communities can change depending on which algorithm is used. For example in Figure 6, Louvain [12] seems to be a better approach since it detected communities better compared to the InfoMap [13]. We will use the same approach for DCD algorithm as in the work in [5].

6.4 State Aggregation

Function approximation in RL makes it possible to ease learning by approaching the problems partially. State aggregation is a function approximation method that the states are grouped, and thus aggregated together. This process can also be defined as partitioning

the environment into a finite number of regions and then assigning an abstract state ϕ_n for each region n. Each of the state groups/regions has an estimated value [14].
A linear function approximation method is an important function approximation since they assure convergence and they work efficiently with respect to computation. Methods for function approximation are given below which is described in [14] in chapter 9.

6.4.1 Polynomials

In many RL problems, the states are represented using numbers. These problems' function approximations show similarity with applications of regression and interpolation. Regression and interpolation can be easily implemented via polynomials. Therefore, the polynomials hold an important place for finding approximation functions for RL problems. Higher-order polynomials provide better accuracy for complex functions. [14]

6.4.2 Fourier Basis

Fourier Basis is a linear function approximation method which is based on the Fourier series as its name signifies. The Fourier series is practical because of its easy implementation and high performance. It is used for RL problems which the functions to be approximated are unknown. [14]

6.4.3 Coarse Coding

Coarse coding is the representation of a state with overlapping features [14].

6.4.4 Tile Coding

Tile coding is another version of coarse coding for domains with continuous state spaces. It divides the state space into tiles and this division is called tiling. Each tile is processed as the same state.
A uniform grid is an example of a two-dimensional state space and the tiles are represented as squares in this state space. [14]

We will use one of the approaches described above for state aggregation process.

6.5 Skill Learning with Intra-Option

One of the main problems that are faced in this project is skill learning. The problem raises with the question of how to find subpolicies of skills so that they can be used effectively in the big environment. If the environment is discrete, one of the methods to learn subpolicies of skills can be running them in a small RL environment that has one of the subgoals in the real environment as the main goal. Another method can be Intra-Option Learning that learns skills by executing one of them in the big environment without the condition of executing one skill until it terminates. These methods use a Q-Table that includes the values of actions/skills for each state and updates them in every step. However, as mentioned earlier, in continuous state space, agent may never visit the same state as before. So, a state can be represented as abstract state ϕ_n. This way, we can use some kind of Q-table with abstract states to use Intra-option learning for skill learning for the main policy.

To be able to learn the values of subpolicies of skills and main policy, function approximation can be employed as in the work of [6, 7].

Intra-option [8] learning is the method for learning options at the same time while learning on the big environment proceeds instead of using sub-environments. While an option is chosen to be taken from the main policy, other options that are consistent with the chosen one will also be updated in the main policy. This way, in one step, there will be multiple updates for different actions/skills.

We plan to use Intra-option learning method to learn the main policy and the options subpolicies along with function approximation methods described in Section 6.4.

6.6 Skill Coupling (SC)

As mentioned before, oversegmentation is the problem that DCD algorithms faced. SC is one way to overcome this problem introduced in [5] for discrete RL domains. We will implement this method based on the algorithm that is given in the work of [5]. However, since our domain is continuous, there will be modifications needed to be made depending on the process of the project.

7 Professional Considerations

7.1 Methodological considerations/engineering standards

A bar chart is used to show the division of responsibilities and duties among team members and is given in Figure 7, section 8.2. A GANTT chart is used to show the project time line and given in Figure 8, section 8.3.
Source Code Control via Git will be used for the development of the project and the version control.
Existing HRL and DCD implementation [5] is written in Python programming language. For this reason, our project will be written in Python. Also, a graph generator software will be used in Python, since Python has libraries to create graphs and data visualization tools to observe the graphs.
Our project will run in Linux environments since the libraries used in the existing project are convenient for Linux. Other operating systems might also work but there could be unexpected results or problems.

7.2 Realistic Constraints

7.2.1 Economic

The only cost for our project is the cost of an equipment with sufficient performance and capacity since the project works only on digital media. There is no cost for frameworks that will be used since they are open-source and free. However, there may be some additional costs if the project is put into real life because of the necessity of the other equipment.

7.2.2 Environmental

There are no environmental issues if the project is limited to working on digital media only. Although there may be some environmental issues if the project is put into practice in real life. In the latter case, the agent might induce noises to the users. However, there would not be any potential effects on environmental pollution.

7.2.3 Ethical

If the project is used with another application, there may be possible security and privacy issues for the users, only if our project behaves in an unexpected way.

7.2.4 Health and Safety

In the case that the project does not work as expected there might be some safety issues for the users. For instance, if the agent is not trained sufficiently, it may cause danger for the users, especially for infants/children depending on the application and the environment that the project is used.

7.2.5 Sustainability

There are many approaches to solve the RL problems. Our approach works on a continuous state space environment which is represented discretely by using a connectivity graph. Our approach will maintain its sustainability although there may be other approaches that solve the same RL problems more efficiently.

7.2.6 Social

Our project does not cause any social discrimination.

7.3 Legal considerations

Libraries and frameworks that will be used for our implementations are open-source and free. There is no legal issue for this project.

8 Management Plan

8.1 Description of task phases

Phase 1: (Until March) Literature Survey about continuous RL environments and learning methods.
Phase 2: (October to December) Preparing project specification document (PSD).
Phase 3: (November to mid April) Study dynamic community detection algorithms for continuous RL domains.
Phase 4: (November to mid January) Implementation of new continuous environment.
Phase 5: (December to mid January) Preparing analysis and design document (ADD).
Phase 6: (January to mid February) Implementation of Q-Learning/SARSA algorithm.
Phase 7: (February) Test and visualize the functionality of environment.
Phase 8: (February to April) Generating environment connectivity graph and state aggregation process

Phase 9: (March to May) Implementation of Intra-option Learning method and examine whether performance is improved or not.

Phase 10: (April to June) Modification of community detection and SC method for new domain.

Phase 11: (May to June) Test the algorithm with different inputs and observe the results.

8.2 Division of responsibilities and duties among team members

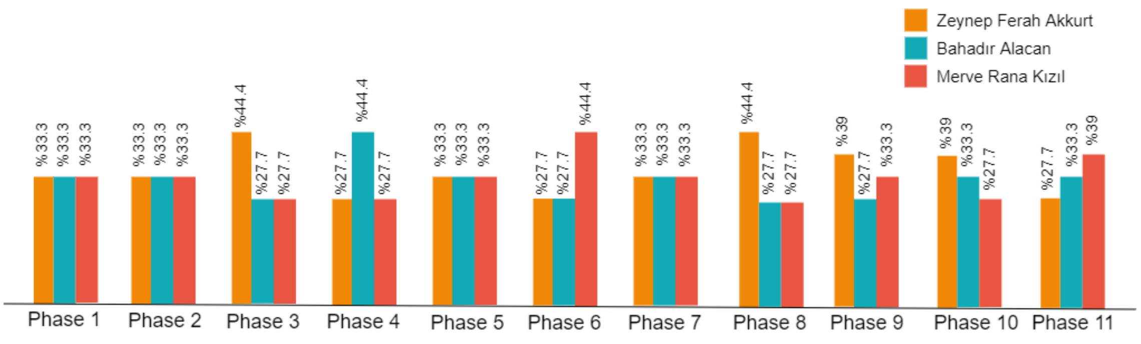


Figure 7: Work Sharing Among Team Members Chart

8.3 Time line with milestones

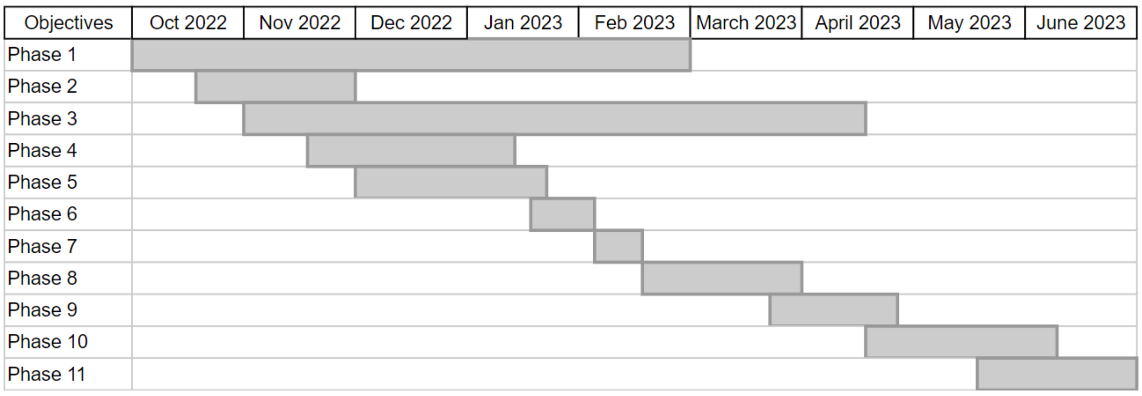


Figure 8: GANTT Chart for the Project Time Line

9 Success Factors and Risk Management

9.1 Success Factors

Measurability Success for the objectives that are given in Section 3 can be listed as follows:

- **Success Factor for Objective #1: Representing the continuous RL domain**

  Domain will be represented with the generated connectivity graph described in Section 6.2. This graph will help us to understand the number of abstract states and give an idea about the environment. We will visualize this graph with the help of the library ”networkx” in Python. From the graph, we will have the information of the number of abstract states and their transitions.

  In this phase, the main aim is to have a graph with a finite number of states with the combination of distance and transition graphs. In a way, the real environment needs to be compressed into an environment which has a finite number of states (aka *state aggregation* mentioned in Section 6). After partitioning the environment into a finite number of regions and assigning an abstract state for each region making up the environment, all non-Markov states within a particular region are represented by one and the same abstract state. Hence, the continuous environment is transformed into one with a finite number of (abstract) states and the Markov models can now be employed to apply RL algorithms. This process has two parameters: 1) a *compression ratio*, γ given in Equation (2), defined as the rate of the number N of the non-Markov states visited that are the members of the continuous state space (i.e., each node in transition graph is a non-Markov state) over the number N_A of the abstract states that represent these non-Markov states and 2) a *representation error*, ϵ_r, that specifies the average Euclidean distance between the non-Markov states visited by the agent and its representative abstract states.

  $$\gamma = \frac{N}{N_A} \tag{2}$$

  where N is the number of the non-Markov states visited throughout the learning process and N_A the number of the abstract states used to represent the non-Markov states visited.

  $$\epsilon_r = \frac{1}{N} \sum_{n=1}^{N_A} \sum_{k=1}^{S_n} \sqrt{[(x_k - x_n)^2 + (y_k - y_n)^2]} \tag{3}$$

  $$N = \sum_{n=1}^{N} S_n \tag{4}$$

  In Equation (3),

  - S_n is the number of non-Markov states visited in region n and represented by the abstract state ϕ_n

- N_A is the number of abstract states ϕ_n
- N is the total number of non-Markov states
- x_k and y_k is the coordinate of the original (i.e., non-Markov) state s_k (a member of the continuous state space) observed at the k^{th} move of the agent in the domain
- x_n and y_n is the representative (i.e., abstract) state ϕ_n coordinate for the current non-Markov state s_k.

We will calculate the Euclidean distance between each non-Markov state s_k in the region n and its representative/abstract state ϕ_n. This operation will be done for each abstract state in the domain until all non-Markov state visited are considered. At the end, we will find the total distance of the learning process by summing up the individual Euclidean distances. We then divide the total distance by N and get the average value. This will be our error metric. Detailed equation give in 3.

The aim is to obtain a compression ratio as much as possible, and obtaining a total representation error as less as possible. So, the aim can be expressed with the maximization of the value z given in Equation (5), which means maximizing the ratio between the compression ratio and the representation error.

$$z = \frac{\gamma}{\epsilon_r} \tag{5}$$

- **Success Factor for Objective #2: Detecting communities and constructing skills**

Modularity [15], denoted by M and given in Equation (6), is a metric that measures how strongly nodes are connected in each community and it is strictly between -1 and 1 [16]. Therefore it is an important feature to characterize the community structure of a graph. If the number of intra-community edges is no better than random, we will get $M = 0$. Values approaching $M = 1$, indicate a strong community structure.

$$M = \frac{1}{2m} \sum_{i,j} \left[A_{i,j} - \rho \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{6}$$

In Equation (6), $A_{i,j}$ is the number of the edges between nodes i and j, ρ is the resolution parameter ($\rho > 0$), k_i is the degree of node i, k_j is the degree of node j,

$\delta(c_i, c_j)$ is the Kronecker delta (1 if nodes i and j belong to the same community, 0 otherwise) and $2m$ is the total number of edges in the network.

Community detection algorithms use the modularity function as an objective function to maximize. Maximizing modularity function is an NP-hard problem, so true communities may not always be found correctly.

Since modularity is an effective metric in identifying communities, it is widely used in community detection algorithms. In Figure 9 the relation of community structure and modularity is visualized using different partitions on the same graph.

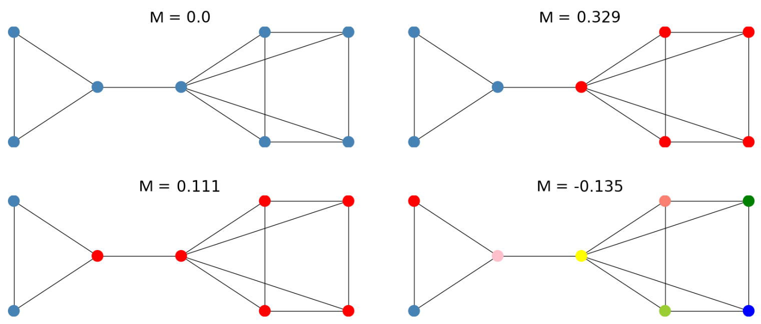


Figure 9: This figure is taken from the work in [5]. Different modularities for a graph. Communities that nodes belong to are indicated by colors.

To be able to understand detected communities/subgoals better, graph will be visualized with colors (one color for each community, an example shown in Figure 5, 6,9) so that we can see how many communities are detected. From this, it can be seen if there is an unnecessary community or not. When we say unnecessary community, we mean that the oversegmentation problem mentioned in Section 2 (a problem which divides a community into more communities) This way, we can conclude that the algorithm that is used, does not work as efficient as we want since it constructs too many colored areas or less than it should be.

- **Success Factor for Objective #3: Learning Skills and Improving skill learning process with Intra-Option Learning**

  Skill learning can be detected by looking inside of a skill and examine its subpolicy. We will trace each action/option that the agent takes. To be able to trace the performance of learning, we will have log output files that consist of the interactions of the agent with the environment along with state-action values, rewards and etc. This way, by looking at the log files and the communities we can observe the process and can make a conclusion about the performance. That way, learning process can be followed in case of an error or interruption.

A learning curve can be defined as the representation in graph form of the rate of learning something over time or repeated experiences. Our project requires lots of experiments with different inputs. To be able to understand this process, learning curves can be used. The learning curve is a visual representation of how long it takes to acquire new skills or knowledge. An example can be seen in Figure 10. Our learning curve will have the number of episodes in the x-axis and number of steps in the y-axis per episode (Definition of metrics is given under the Figure 10). The performance of the learning process will be compared with the learning in continuous RL domains such as in [6]. It is expected to be at least as good as their work or even better.

- **Success Factor for Objective #4: Evaluating Performance of Skill Coupling in Continuous Domains**

  Behaviour of Skill Coupling (SC) will be evaluating based on its performance in terms of metrics defined below in continuous state spaces. We will use graphical charts (learning curves described above and shown in Figure 10) to visualize the metrics. There will be several comparisons in this project with the previous works [5–7].

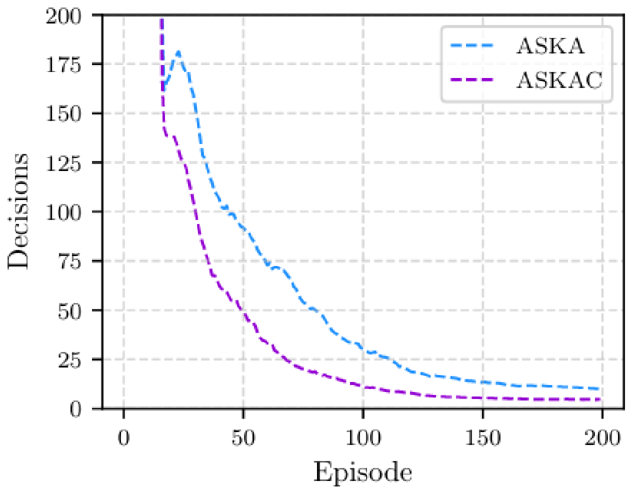


Figure 10: An example learning curve taken from [5]

Metrics can be listed as follows:

  – Number of Steps: Number of actions that agent takes per episode from initial state to goal state.

- Number of Skills: Number of skills/options that agent takes per episode from initial state to goal state after constructing skills.
- Number of Decisions: Number of decisions that agent takes per episode from initial state to goal state. Decisions can be a primitive action or a skill
- Accumulated Reward: Total reward gained after one episode.

Note that number of steps includes only primitive actions, if a skill is chosen, then taken steps are counted as many primitive action as the skill has inside. However, in number of decisions a primitive action or an option counted as one.

Above graph shows that the method ASKAC has reached a better learning in terms of number of decision. As the number of episode increases, number of decision decreases. This proves that the method ASKAC has improved the learning process by decreasing the amount of decisions that the agent makes.

Learning process has two important features that needs to be examined by looking at the learning curve; speed of decrease of the number of steps and convergence rate. If the number of steps/decisions has decreased way more faster than the other algorithm, this shows us that it learns faster. However it does not mean it will converge faster. That is why we need to examine its convergence rate. If the convergence is faster, the algorithm finds a solution faster, however it does not need to be the optimal one. Convergence means number of decisions is no longer improving and stays almost the same as previous episode. An example can be seen in Figure 10, for ASKA convergence starts after episode 160, for ASKAC convergence starts after approximately around 140). ASKAC is a better algorithm in terms of convergence rate and speed of decrease of the number of decisions.

9.2 Risk Management

There are some possible risks that may be encountered throughout the project. These are listed below.

Risk #1:

The HRL implementation [1] that will be used may have libraries or functions that are deprecated.

- Resolution to Risk #1: The existing project should be examined before being used.

If there are deprecated libraries or functions, they should be modified with their new versions.

Risk #2:

Pinball domain does not exist in any RL libraries. Because of that we plan to use an open-source code for the implementation of the environment. It is possible that some functionalities may be missed or not available anymore.

- Resolution to Risk #2: Unavailable algorithms and missed functionalities are requested from the authors or they are implemented during the project progress.

Risk #3:

The performance of the method ASKAC is highly dependent on the community detection and skill learning method since we use a continuous RL domain. There is a possibility that method can excess the required learning time.

- Resolution to Risk #3: Based on the testing results, community detection algorithm can be changed along with other parameters.

10 Benefits and Impact of the Project

If the project is successful, SC method will be an effective method for HRL problems in continuous state spaces in terms of time. Researchers working on HRL in a real world problems will benefit from the result of the project.

10.1 Scientific Impact

This project aims to enlighten a proposed method with other introduced novel approaches for other researchers who work in this area. We discuss several challenging problems in the area of HRL. We believe it will bring out new ideas to most of the researchers with this study. So, it has a potential to be an scientific paper.

10.2 Economic/Commercial/Social Impact

The outcome of this project will be an academic study for other researchers in the area of HRL. It can improve the education level about Skill acquisition and coupling in continuous domains.

10.3 Potential Impact on New Projects

We expect this project to have a pioneer for continuous state spaces and how to speed up the learning process in such an environment. If the project reaches its goal, we believe it will make an effect on the searches on continuous RL domains with Skills to improve learning along with its complexity.

10.4 Impact on National Security

This project is based on academic research, therefore the impact of a national security does not apply for this project.

References

[1] R. S. Sutton, D. Precup, S. Singh, Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning, Artificial intelligence 112 (1999) 181–211.

[2] D. Zhuang, J. M. Chang, M. Li, Dynamo: Dynamic community detection by incrementally maximizing modularity, IEEE Transactions on Knowledge and Data Engineering 33 (2019) 1934–1945.

[3] R. Aktunc, I. H. Toroslu, M. Ozer, H. Davulcu, A dynamic modularity based community detection algorithm for large-scale networks: Dslm, in: Proceedings of the 2015 IEEE/ACM international conference on advances in social networks analysis and mining 2015, pp. 1177–1183.

[4] M. Cordeiro, R. P. Sarmento, J. Gama, Dynamic community detection in evolving networks using locality modularity optimization, Social Network Analysis and Mining 6 (2016) 1–20.

[5] S. H. Çavuş, Z. Kumralbaş, K. Coşkun, B. Tümer, Autonomous acquisition of arbitrarily complex skills using locality based graph theoretic features: A syntactic approach to hierarchical reinforcement learning, under evaluation for publication at Evolving Systems (2022).

[6] F. Shoeleh, M. Asadpour, Graph based skill acquisition and transfer learning for continuous reinforcement learning domains, Pattern Recognition Letters 87 (2017) 104–116.

[7] G. Konidaris, A. Barto, Skill discovery in continuous reinforcement learning domains using skill chaining, Advances in neural information processing systems 22 (2009).

[8] R. S. Sutton, D. Precup, S. Singh, Intra-option learning about temporally abstract actions., in: ICML, volume 98, pp. 556–564.

[9] G. L. Miller, S.-H. Teng, W. Thurston, S. A. Vavasis, Separators for sphere-packings and nearest neighbor graphs, Journal of the ACM (JACM) 44 (1997) 1–29.

[10] W. Aiello, F. Chung, L. Lu, A random graph model for massive graphs, in: Proceedings of the thirty-second annual ACM symposium on Theory of computing, pp. 171–180.

[11] R. S. Sutton, A. G. Barto, et al., Introduction to reinforcement learning (1998).

[12] V. A. Traag, J. Bruggeman, Community detection in networks with positive and negative links, Physical Review E 80 (2009) 036115.

[13] L. Bohlin, D. Edler, A. Lancichinetti, M. Rosvall, Community detection and visualization of networks with the map equation framework, in: Measuring scholarly impact, Springer, 2014, pp. 3–34.

[14] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[15] M. E. Newman, M. Girvan, Finding and evaluating community structure in networks, Physical review E 69 (2004) 026113.

[16] V. D. Blondel, J. L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech.-Theory Exp. 2008 (2008) P10008.