T.C.
MARMARA UNIVERSITY
FACULTY of ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CSE4197 Engineering Project I

ANALYSIS AND DESIGN DOCUMENT

Title of the Project

"Autonomous skill acquisition in reinforcement learning using locality based graph theoretic features"

January 5, 2020

Group Members
150115045 Semiha Hazel Çavuş
150115051 Zeynep Kumralbaş

Supervised by
Associate Professor M. Borahan Tümer

1. Introduction

Reinforcement learning (*RL*) is a machine learning technique where an agent takes an action in an environment, moves to the next state and receives rewards or punishments regarding this new state. So, it can learn a satisfactory (hopefully optimal or possibly a near optimal) policy that leads the agent to the goal state in the environment.

As the environment grows too large, converging to a satisfactory policy for regular RL algorithms such as flat Q–learning becomes quickly infeasible. So, splitting up the environment (hence the huge state space) into regions called *subenvironments* based upon the structure of the environment and composing the policy (or policies) of a sequence (or several sequences) of components each learned in one of these regions is an effective solution to RL large state spaces. Learning each component called an *option/skill/macro-action* and learning the main policy (i.e., composing the main policy as one or more sequences) of these skills occur in two different levels or hierarchies of learning. Hence, this process of splitting the environment into subenvironments divides, in fact, learning into hierarchies. In each subenvironment, one or more subtasks are defined depending upon the subgoals selected and each of these subtasks are considered as an individual RL problem to solve in the first hierarchy. Then in the following hierarchy, the skills learned in the former hierarchy are used as the building stones to compose the main policy. This technique is called *Hierarchical Reinforcement Learning* (*HRL*) [1]. In HRL, there are actions as well as skills. A skill representing a specific subpolicy is learned to solve a certain subtask by reaching a subgoal from some initial state within a certain region.

As described above, to construct a skill, the subgoals should be detected. However, the agent does not know the environment completely since it learns the environment episode by episode [2]. Thus, the subgoals may change since the partial graph evolves over time. So, the subgoal changes should be detected again. Subregions can be related as community structures. Further information about this relation will be given in section 1.1.

The focus of the work is on how to improve the time complexity of the detection of subgoals using dynamic community detection algorithms and skill construction. There are works which use dynamic community detection algorithms based on modularity, however there is no work that uses a community detection algorithm based on permanence in RL domain. In our work, permanence and modularity will be used, and it will be our novelty.

1.1 Problem Description and Motivation

In case an environment is represented by a large state space, it is usually a good idea to divide the state space into subregions depending on the topology of the environment.

A memory based agent in RL progressively constructs a partial model of the environment it is in an interaction with in a dynamic fashion where the partial model grows more similar to the actual environment as learning proceeds. Thus, subgoals are also detected in a dynamic manner [1].

In RL, environments may be effectively modeled using graphs [2, 11]. The connection points of subregions can be good candidates to be subgoals by their nature. Hence, subgoals can be considered as bottlenecks of a graph, and they can be isolated from other nodes using a graph property, *betweenness centrality (BC)* [2]. BC values of nodes in a graph are shown in Fig. 1. It defines the bottlenecks very well; however, calculation of BC values is $O(n^3)$ since subgoals might change one episode to another, BC values are calculated from scratch per episode. A possible improvement in computation speed would follow if BC values are computed at longer intervals than each episode; however, this would potentially delay the detection of subgoals while the time complexity of the process would not asymptotically change. The problem of using BC values is that its calculation brings a computational overhead.

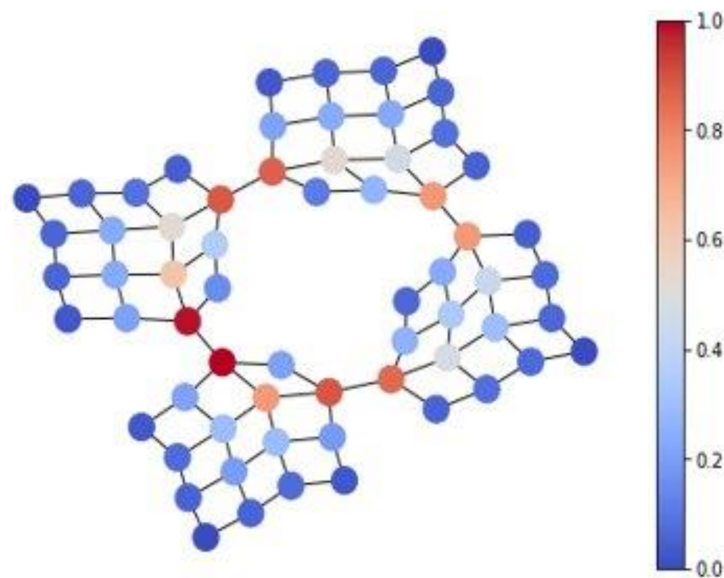

Fig. 1 - An example for subgoal representation. The graph is colored by betweenness centrality values. The red ones have high values since they are subgoals.

Although this problem as explained appears to be a problem within RL domain (improving the time complexity in HRL paradigm) this problem manifests much similarity to a problem within the context of graph theory: detection of communities based on graph properties.

Communities, a common practice in graph theory, can be considered as a candidate to model these subregions; where a community is defined as a group of nodes which in a set exhibit a relatively higher similarity [9]. There are community detection algorithms which run on static graphs [4, 5] and dynamic graphs [6, 7].

The advantage of using communities is that they can be detected dynamically. When a partial graph is updated, there is no need to detect all communities from scratch. In dynamic community detection algorithms, the communities that are affected only from the addition of new nodes or edges are disbanded. Then, the detection algorithms run locally for new nodes and nodes of disbanded communities. The rest of the communities do not change. Hence, all communities are not detected over and over for each episode, only related communities are detected.

The dynamic community detection is a convenient solution for detecting subgoals both from the aspects of accuracy and time complexity.

1.2 Scope of the Project

Our project is based on discrete and deterministic environments. The implementation of the entire HRL algorithm is not in the scope of our project. We will devise a decision logic as a part of the HRL algorithm whether to run the skill construction algorithm observing subgoal states via an analysis of the partial transition graph.

We will embed this decision logic to the HRL algorithm implemented in [12]. The partial transition graph obtained as the output from [12] is used as an input in our module. The decision of whether to construct skills will be the output from our module and it will be the input for the next part of the HRL. These processes will continue in an iterative manner.

A discrete environment is assumed, and the state space is limited to reasonably complex environments, where reasonable indicates a size within the processing capabilities of the computer. For example, on a computer that has Intel® Core™ i7-8565U CPU @ 1.80GHz 1.99 GHz and 8GB RAM cannot handle solving a grid environment that has one million states in a reasonable time. Our approach works in only two hierarchies, for actions and skills. However,

HRL algorithm can be run in both stationary and non-stationary environments, the stationary environments are considered for our project.

Definitions, Acronyms, and Abbreviations

RL:	Reinforcement Learning
HRL:	Hierarchical Reinforcement Learning
BC:	Betweenness Centrality
NMI:	Normalized Mutual Information
ARI:	Adjusted Rand Index
SCC:	Strongly Connected Component
DFS:	Depth First Search
GSL:	Graph based Skill Learning
STL:	Skill based Transfer Learning
ICD:	Incremental Community Detection Algorithm
DCD:	Dynamic Community Detection
Lattice graph:	It might be considered as a finite part of a regular graph where each vertex has the same number of neighbors.
Modularity:	A metric that represents how strongly are nodes connected to each other in each community.
Permanence:	A vertex-based metric, considers the strength of membership of a vertex to a community [14].
DynaMo:	Dynamic modularity-based community detection algorithm
MaxPerm:	A community detection algorithm based on maximizing permanence, static algorithm
DyPerm:	Dynamic Community Detection by maximizing permanence, dynamic version of MaxPerm algorithm

2. Related Work

There are a lot of works [2,3,9,10,11,15,16,17] for finding subgoal states and construction skills dynamically or statically. The ones based on graph properties are very similar to our work and will be discussed in this section.

2.1 A graph-theoretic approach toward autonomous skill acquisition in reinforcement learning

In this article, authors use a graph-theoretic approach to find subgoals and then to construct skills. The authors state that the visiting frequency of and the probability of transition to the subgoal states are lower than other states. Given this fact, their algorithm is based on frequency-based and graph-partitioning methods. Their RL algorithm builds a transition graph and updates it in every e episodes, with e a parameter of their method. There is also another parameter t_t, which is the minimum required probability for an edge to stay in the transition graph. Then the transition graph is divided into some huge clusters called *strongly connected components (SCCs)* using a linear time algorithm based on depth first search (DFS). Then the states connecting each two SCCs, namely subgoal states, are found considering border states of SCCs as potential candidates of subgoal states. [9]

2.2 Graph based skill acquisition and transfer learning for continuous reinforcement learning domains

In this article, authors mentioned two main problems of RL methods, curse of dimensionality and excess required learning time, and they proposed two novel approaches, which are *Graph based Skill Learning* (GSL) and *Skill based Transfer Learning* (STL). GSL method discovers skills adaptively. HRL is an efficient method since they worked on continuous state space. They detect communities from *connectivity graph* to make state space discrete. Connectivity graph contains both agent's behavior and environment's dynamics. The efficiency of this method depends on both the structure of connectivity graph and qualification of community detection algorithm. They achieved best performance with combination of the transition graph and the distance graph with power-law distribution as the connectivity graph, and Louvain as the community detection algorithm. STL method transfers skills to target task. [10]

2.3 Constructing Temporally Extended Actions through Incremental Community Detection

In this article, authors use graph-based approach to divide the undirected and unweighted state transition graph into communities where nodes in each community exhibit relatively higher similarity. Communities of the transition graph are detected considering a graph property called *modularity* [9]. Modularity is a metric that represents how strongly are nodes connected to each other in each community. Since modularity maximization is an NP-hard problem, the authors employ a heuristic modularity maximization method, the Louvain algorithm [5]. If the graph has m edges in total, the Louvain algorithm runs in $O(m)$. The Louvain algorithm obtains a base partition, then latter partitions are updated depending upon this base partition in the incremental community detection algorithm (ICDA). ICDA is rule-based and updates the communities (partitions) dynamically. Each community is characterized as a macro-state (aka. aggregated), and skills are constructed between each of these macro-states. [11]

3. System Design

3.1 System Model

Subgoal changes should be detected as the agent learns the environment. A module will be implemented that determines whether the subgoal states are changed or not. The RL module, dynamic community detection (D.C.D) module and skill planner can be expressed as a client-server connection. The communication between our modules can be seen in Fig. 2.

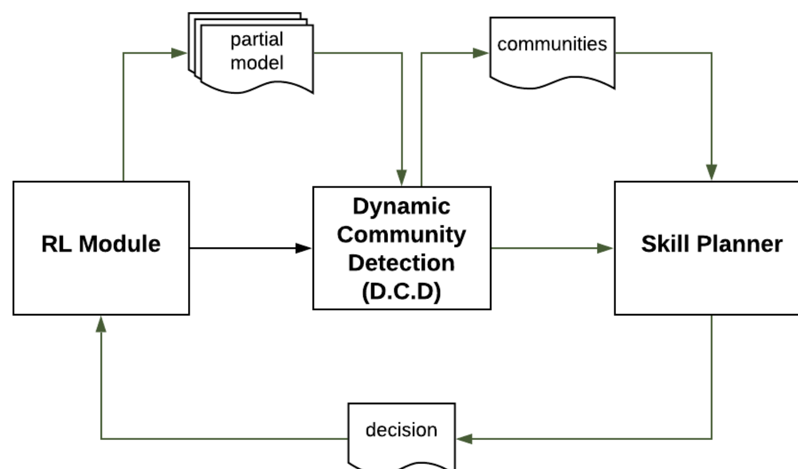

Fig. 2 - Communication between RL module, D.C.D module and Skill Planner

While the agent learns the environment, it sends partial graphs of the environment based on its experience to D.C.D module. D.C.D module will compare the new and the previous partial graphs and determine new communities. Skill planner makes a decision comparing previous and current communities and this decision is forwarded to RL module. This decision may be:

- If there are subgoal changes, construct skills
- If there are not subgoal changes, continue with existing skills.

This process continues until the agent learns the hopefully optimal or near optimal path.

3.2 Flowchart and/or Pseudo Code of Proposed Algorithms

As described above, RL module, D.C.D module and skill planner are in communication. If skill planner makes a decision of construct skills, skill construction algorithm works in RL module. If skill planner makes a decision of continue with existing skills, RL module uses the existing skills. This flowchart can be seen in the Fig. 3.

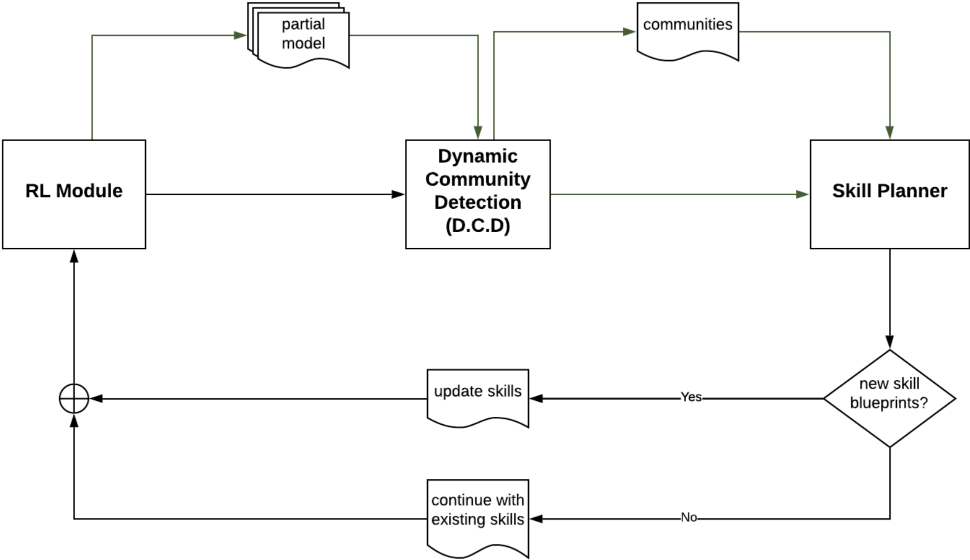

Fig. 3 - Flowchart of Communication of Modules

In our D.C.D module, we plan to use DyPerm [14] which is a dynamic community detection algorithm based on permanence or DynaMo [7] which is also a dynamic community detection algorithm based on modularity or a combination of DyPerm and DynaMo algorithms. Pseudo code of DyPerm can be seen in Algorithm 1, and DynaMo in Algorithm 2.

```
Algorithm 1: NodeAddition
1  Input: Node u to be added; Set of edges E(u, v) of node u;
   Current structure C_t.
2  Output: Updated community structure C_{t+1}
3  if E==null;
4      C_{t+1} ← C_t ∪ {u};
5  else
6      C_{t+1} ← C_t ∪ {u};
7      for each e ∈ E(u, v) do
8          C_{t+1} ← EdgeAddition(e, C_t); //Algorithm 3
9          C_t ← C_{t+1};
10 return C_{t+1};
```

```
Algorithm 2: NodeDeletion
1  Input: Node u to be added; Set of edges E(u,v) of node u;
   Current structure C_t.
2  Output: Updated community structure C_{t+1}
3  if E==null;
4      C_{t+1} ← C_t − ∪{u};
5  else
6      C_{t+1} ← C_t − ∪{u};
7      for each e ∈ E do
8          C_{t+1} ← EdgeDeletion(e, C_t); //Algorithm 6
9          C_t ← C_{t+1};
10 return C_{t+1};
```

```
Algorithm 3: EdgeAddition
1  Input: Edge u, v to be added; Current structure C_t.
2  Output: Updated community structure C_{t+1}
3  C_u ← Comm(G, u);
4  C_v ← Comm(G, v);
5  if C_u == C_v then
6      C_{t+1} ← IntraEdgeAddition(e, C_t) //Algorithm 4
7  else
8      C_{t+1} ← InterEdgeAddition(e, C_t) //Algorithm 5
9  return C_{t+1};
```

```
Algorithm 4: IntraEdgeAddition
1  Input: Graph G (V,E), Edge e(u, v) to be deleted;
         Current structure C_t.
   Output: Updated community structure C_{t+1}
2  C_u ← Comm(G, u);
3  C_v ← Comm(G, v);
4  G(V, E) ← G(V, E) + (u, v)
5  C_{t+1} ← C_t
6  return C_{t+1};
```

```
Algorithm 5: InterEdgeAddition
1  Input: Edge u, v to be added; Current structure C_t.
2  Output: Updated community structure C_{t+1}
3  visited ← null, queue ← null;
4  C_u ← Comm(G, u);
5  C_v ← Comm(G, v);
6  Cu_new ← C_u;
7  Cv_new ← C_v;
8  PermCu_old ← Perm(C_u);
9  PermCv_old ← Perm(C_v);
10 G(V, E) ← G(V, E) ∪ (u, v)
11 Append u to queue;
12 Add head(queue) to visited;
13 while (length(queue) > 0) do
14     C ← current community of head(queue);
15     for each unvisited node i in queue do
16         visited ← visited ∪ head(queue);
17         P_1 = Perm(i) if i stays in Cu_new;
18         P_2 = Perm(i) if i moves to Cv_new;
19         If(P_2 > P_1) do
20             Move i to Cv_new;
21             Remove i from Cu_new;
22     for each node q in queue do
23         Add unvisited internal neighbors of q to queue;
24 if Perm(C_u) + Perm(C_v) > PermCu_old + PermCv_old do
25     C_{t+1} ← C_t − C_u − C_v ∪ Cu_new ∪ Cv_new;
26 return C_{t+1};
```

```
Algorithm 6: EdgeDeletion
1  Input: Graph G (V,E), Edge e(u, v) to be deleted;
         Current structure C_t.
   Output: Updated community structure C_{t+1}
2  C_u ← Comm(G, u);
3  C_v ← Comm(G, v);
4  if C_u == C_v then
5      C_{t+1} ← IntraEdgeDeletion(G, e, C_t) //Algorithm 7
6  else
7      C_{t+1} ← InterEdgeDeletion(G, e, C_t) //Algorithm 8
8  return C_{t+1};
```

```
Algorithm 7: IntraEdgeDeletion
1  Input: Graph G (V,E), Edge u, v to be deleted;
         Current structure C_t.
2  Output: Updated community structure C_{t+1}
3  visited ← null, queue ← null;
4  C_uv ← Comm(G, u)
5  G(V, E) ← G(V, E) - (u, v)
6  Append u to queue;
7  Add head(queue) to visited;
8  while (length(queue) > 0) do
9      for each unvisited node i in queue do
10         C_u ← C_u ∪ head(queue);
11         visited ← visited ∪ head(queue);
12 for each node q in queue do
13     Add unvisited internal neighbors of q to queue;
14         Remove q;
15 Follow step 5 to 13 to obtain C_v;
16 if Perm(C_uv) < Perm(C_u) + Perm(C_v) do
17     C_{t+1} = C_t − C_uv ∪ C_u ∪ C_v;
18 else
19     C_{t+1} = C_t;
20 return C_{t+1};
```

```
Algorithm 8: InterEdgeDeletion
1  Input: Graph G (V,E), Edge e(u, v) to be deleted;
         Current structure C_t.
   Output: Updated community structure C_{t+1}
2  C_u ← Comm(G, u);
3  C_v ← Comm(G, v);
4  G(V, E) ← G(V, E) - (u, v)
5  C_{t+1} ← C_t
6  return C_{t+1};
```

Algorithm 1 - Pseudo code of DyPerm algorithm

```
Algorithm 1: DynaMo Initialization (Init)
   Input: V^(t+1), E^(t+1), V^(t), E^(t), C^(t).
   Output: △C_1, △C_2.
 1 △E ← A set of edges changed from E^(t) to E^(t+1);
 2 △V_add ← V^(t+1)\V^(t); △V_del ← V^(t)\V^(t+1);
 3 △C_1 ← Ø; △C_2 ← Ø;
 4 for e_ij ∈ △E do
 5     for k ∈ {i, j} do
 6         if k ∈ △V_del then
 7             △C_1 ← △C_1 ∪ {c_k};
 8             for e_kl ∈ E^(t) do
 9                 △C_1 ← △C_1 ∪ {c_l};
10         if k ∈ △V_add then
11             △C_1 ← △C_1 ∪ {c_k};
12             w_max = 0; c ← Ø;
13             for e_kl ∈ E^(t+1) do
14                 △C_1 ← △C_1 ∪ {c_l};
15                 if w_kl > w_max then
16                     w_max = w_kl; c ← {k, l};
17             △C_2 ← △C_2 ∪ {c};
18     if i, j ∉ △V_del ∪ △V_add then
19         if e_ij ∉ E^(t+1) or w^t_ij > w^(t+1)_ij then
20             if c_i = c_j then
21                 △C_1 ← △C_1 ∪ {c_i};
22                 for k ∈ {i, j} do
23                     for e_kl ∈ E^(t) do
24                         △C_1 ← △C_1 ∪ {c_l};
25         if e_ij ∉ E^(t) or w^t_ij < w^(t+1)_ij then
26             if c_i = c_j then
27                 △C_1 ← △C_1 ∪ {c_i}; c ← {i, j};
28                 △C_2 ← △C_2 ∪ {c};
29             else
30                 △w = w^(t+1)_ij − w^t_ij; c_k = c_i ∪ c_j;
31                 α_2 = α_(c_i) + α_(c_j) − α_(c_k); β_2 = β_(c_i) + β_(c_j);
32                 δ_1 = 2m − α_2 − β_2; δ_2 = mα_2 + β_(c_i)β_(c_j);
33                 if 2△w + δ_1 > sqrt(δ_1^2 + 4δ_2) then
34                     △C_1 ← △C_1 ∪ {c_i, c_j}; c ← {i, j};
35                     △C_2 ← △C_2 ∪ {c};
36 return △C_1, △C_2.
```

```
Algorithm 2: DynaMo
   Input: G^(t+1), G^(t), C^(t).
   Output: C^(t+1).
 1 △C_1, △C_2 ← Init(V^(t+1), E^(t+1), V^(t), E^(t), C^(t));
 2 C^(t+1) ← C^(t);
 3 for c_i ∈ △C_1 do
 4     C^(t+1) ← C^(t+1)\{c_i};
 5     for k ∈ c_i do
 6         new singleton community: c_k ← {k};
 7         C^(t+1) ← C^(t+1) ∪ {c_k};
 8 for c = {i, j} ∈ △C_2 do
 9     new two-vertex community: c_k ← {i, j};
10     C^(t+1) ← (C^(t+1)\{c_i, c_j}) ∪ {c_k};
11 C^(t+1) ← Louvain(C^(t+1), G^(t+1));
12 return C^(t+1).
```

Algorithm 2 - Pseudo code of DynaMo algorithm

Our modified HRL algorithm [12] can be seen in Algorithm 3.

Algorithm 1 : HRL

1: Initialize $Q(s,a)$, $Model(s,a)$ and $PQueue$ to empty
2: **loop** forever
3: s = current state, $a = policy(s,Q)$
4: Time-step count, $k = 0$
5: $(s', r, k) = execute(s,a)$
6: $Model(s,a) = (s', a, r, k)$
7: $p = r + \gamma^k max_a Q(s,a) – Q(s,a)$
8: **if** $(p > \theta)$ **then**
9: Insert (s,a) into $PQueue$ with priority p
10: **end if**
11: **for** N times while $PQueue$ is not empty **do**
12: (s,a) = pop from $PQueue$
13: $(s', r) = Model(s,a)$
14: $Q(s,a) = Q(s,a) + \varphi[r + \gamma^k max_{a'} Q(s',a') - Q(s,a)]$
15: **for** all s'', a'' predicted to lead to s **do**
16: r'' = predicted reward from $Model$
17: $p = r'' + \gamma^k max_{a'} Q(s,a) – Q(s'',a'')$
18: **if** $p > \theta$ **then**
19: push (s'', a'') into $PQueue$ with priority p
20: **end if**
21: **end for**
22: **end for**
23: **if** end of the episode **then**
24: send partial graph to community detection algorithm
25: **if** message is "update set of skills" **then**
26: update skills
27: **else**
28: continue
29: **end if**
30: **end if**
31: **end loop**

Algorithm 3 - Pseudo Code of Modified HRL algorithm

Proposed algorithm of Skill Planner in Algorithm 4.

Algorithm 1 : Skill Planner Algorithm

Input: new communities
Output: decision message

1: Compare previous and current communities
2: **if** communities are changed **then**
3: send skill blueprints and "update skills" message
4: **else**
5: send "no change" message
6: **end if**

Algorithm 4 - Pseudo Code of Skill Planner

3.3 Comparison Metrics

The learning performance of HRL algorithm is measured using a diversity of the following metrics:

- The objective of the agent is to reach the goal state using a hopefully optimal policy (i.e., taking a minimum number of actions). It expresses how rapid the learning occurs as well as revealing convergence and divergence of the learning process. Improving the skill construction planner mechanism is a contribution where skills are constructed only as required. Discovering skills as early as possible has a remarkable effect on learning time. This yields the agent to choose to execute skills, instead of primitive actions, so number of steps to reach a subgoal would be minimized. Also, the total number of episodes learning is achieved within is another comparison metric.
- Another metric to measure the learning performance of HRL is the expected total reward the agent receives after each episode. If the skill construction mechanism builds skills in advance, this increases the expected total reward in each episode.

The metrics to measure the accuracy of detected subgoals are given below:

- Number of true positives, true negatives, false positives and false negatives to calculate precision, recall and F_1 score metrics. These measurement metrics are given in Table 1 and Table 2.

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Actual</td></tr>
<tr><td>Subgoal Change Present</td><td>Subgoal Change Absent</td></tr>
<tr><td rowspan="2">Predicted</td><td>Subgoal Change Detected</td><td>True Positive (TP)</td><td>False Positive (FP)</td></tr>
<tr><td>Subgoal Change not Detected</td><td>False Negative (FN)</td><td>True Negative (TN)</td></tr>
</table>

Table 1 - Subgoal Change Detection Results

Measurement metric	Definition	Ideal Value
Precision	$\frac{TP}{TP+FP}$	1
Recall	$\frac{TP}{TP+FN}$	1
F1 Score	$\frac{2*(Recall*Precision)}{Recall+Precision}$	1

Table 2 - Definitions of Measurement Metrics

The accuracy of detected communities from our D.C.D module will be compared with the ground-truth community structures using NMI and ARI measurement metrics.

3.4 Data Sets or Benchmarks

Two-dimensional lattice graphs are good representatives of the grid world environments. There is a formula of lower bound for modularity of lattices according to given parameters. We implemented a synthetic graph generator to obtain different two-dimensional lattice graphs, since we consider only grid worlds. Then the obtained graphs are converted to RL environments. The agent will try to learn that environments.

HRL will be run in a benchmark problem, four-room grid world, since it is used for expressing the performance of skill construction approach [11].

4. System Architecture

We have three modules: 1. RL module where learning of the agent takes place in; 2. D.C.D module for detecting communities locally and dynamically; 3. Skill Planner which compares previous and current communities, makes inferences and shares these inferences with RL task.

Data flow of RL module is given in Fig. 4.

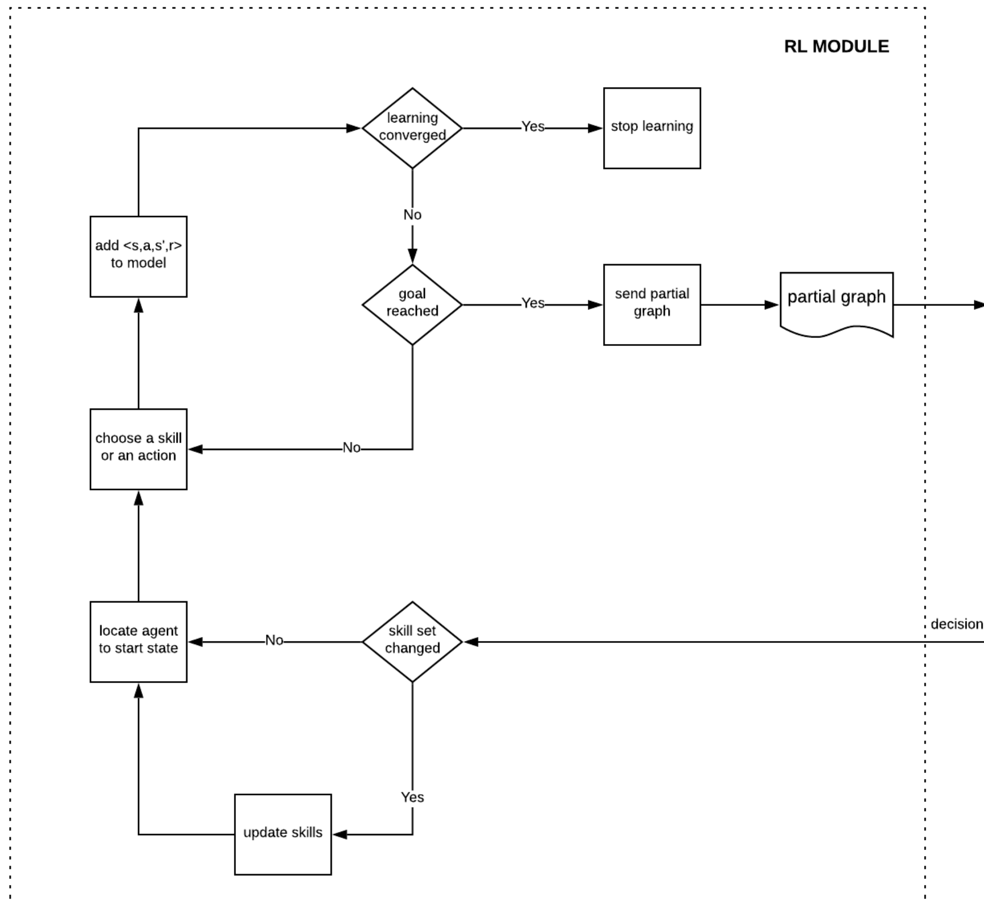

Fig. 4 - Data Flow of RL Module

Control flow of D.C.D module is given in Fig. 5.

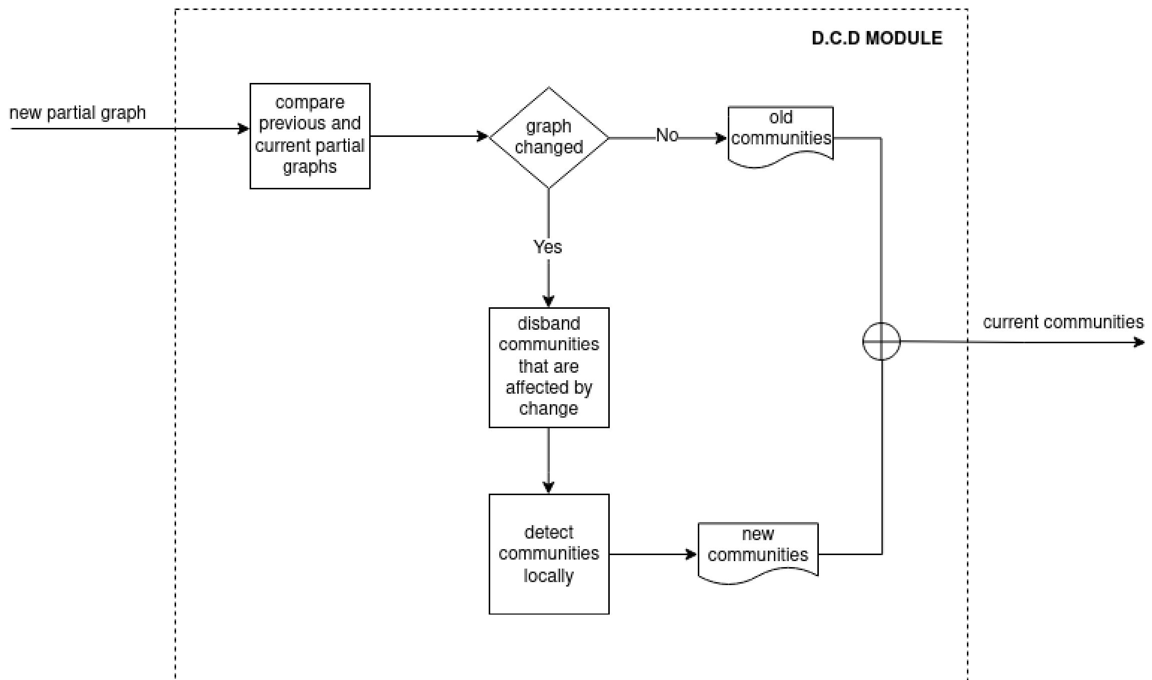

Fig. 5 - Control Flow of D.C.D module

Control flow of Skill Planner is given in Fig. 6.

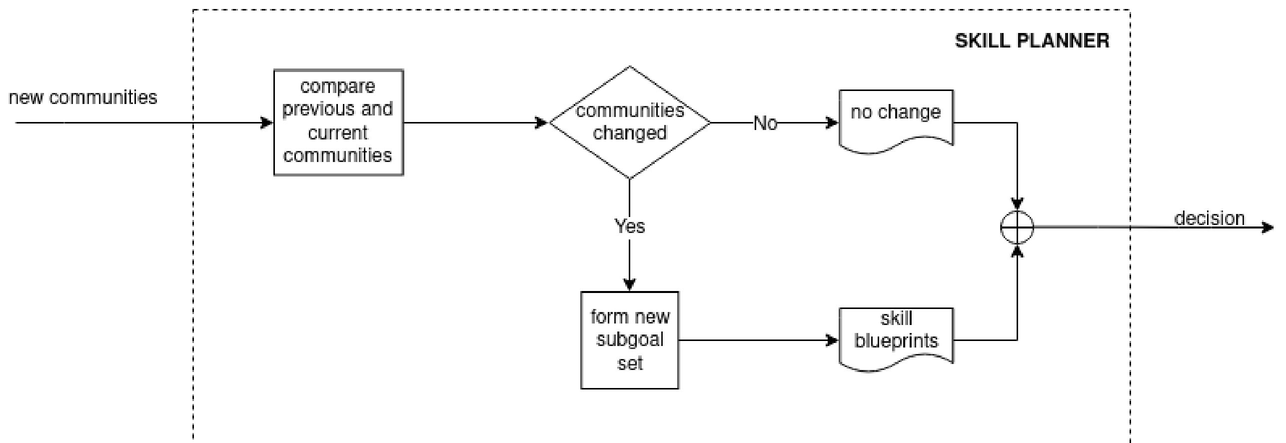

Fig. 6 - Control Flow of Skill Planner

5. Experimental Study

In order to analyze how community detection algorithms, behave on lattice graphs, two experiments are made using static algorithms in section 5.1 and 5.2.

5.1 Community Detection with modularity, Louvain algorithm [4]

Experimental Setup: Louvain algorithm which maximizes modularity was run on some two-dimensional lattice graphs in order to examine how it forms the communities.

Experimental Results: A lattice graph that has at most 9x9 nodes in each subregion and 3x3 subregion in Figure 7. The detected communities with Louvain algorithm are in Figure 8.

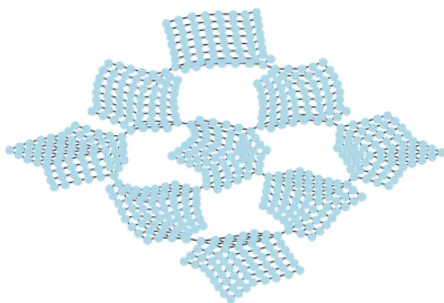

Fig. 7 - Lattice graph

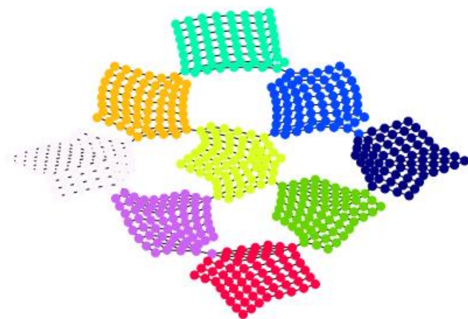

Fig. 8 - Lattice graph with detected communities

Discussions: Lattice graph is created considering the environment has 9 subregions. We expect the community detection algorithm to find these 9 subregions as forming 9 communities. The Louvain algorithm detected each subregion as a community as we expected.

5.2 Community Detection with permanence, MaxPerm algorithm [13]

Experimental Setup: MaxPerm algorithm was run on some two-dimensional lattice graphs in order to examine how it forms the communities.

Experimental Results: A lattice graph that has at most 9x9 nodes in each subregion and 3x3 subregion in Fig. 8. The detected communities with Permanence algorithm are in Fig. 9.

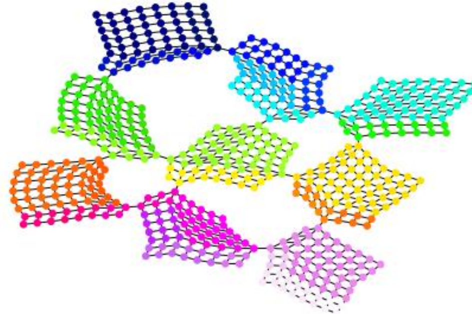

Fig. 9 - Lattice graph with detected communities

Discussions: The environment is considered to have 9 subregions. We expect the community detection algorithm to find these 9 subregions as forming 9 communities. But MaxPerm algorithm found 2 or 3 communities in each subregion, detected different communities from the Louvain algorithm. The result was not as we expected.

5.3 HRL experiment

Experimental Setup: Skill planner will be implemented. HRL will be tested on the different size of environments using modules of our project. The results will be evaluated based on our comparison metrics that are explained in section 3.3.

6. Tasks Accomplished

6.1 Current State of the Project

In current state of the project, these tasks were accomplished:

- Deprecated functionalities of HRL implementation were rewritten in Python.
- The problem in the DynaMo code was solved by contacting its author.
- Synthetic graph generator is implemented in Python. We obtain different graphs giving maximum number of nodes for rows and columns of subregions and number of subregions as parameters to the synthetic graph generator.

- The static version of the permanence MaxPerm code was obtained from its author, since it is not available on the internet. MaxPerm algorithm is tested on some of the graphs that are generated from the synthetic graph generator to understand how it works.
- There is an ongoing work on modifying the dynamic version of the permanence, DyPerm code.

6.2 Task Log

Our meetings are held in our advisor's office and lasted for 1.5-2 hours. Kutalmış Coşkun who is the author of [12] also attends occasionally our meetings.

Meeting 1 - 3.10.2019: In our first meeting, we talked about our project and decided to make more detailed literature survey about both community detection algorithms and subgoal detection algorithms in HRL.

Meeting 2 - 10.10.2019: We focused on researching and understanding the community detection algorithms [6, 7] over the past week. We shared the articles we found with our advisor and Kutalmış Coşkun.

Meeting 3 - 17.10.2019: We focused on researching and understanding the subgoal detection algorithms over the past week. We found an article that is very close to our approach which detects subgoals using a community detection algorithm based on modularity [11]. We thought we were going to come up with a novel solution, but this approach is thought and published already. We talked about how we can add a novelty to our project. After the discussion, we decided to search different graph properties in order to strengthen the detection of subgoals.

Meeting 4 - 24.10.2019: We found an article that proposes a new graph property to find communities more accurate but could not be able to make detailed study on the article [13]. We shared this article with our advisor, and he asked us to understand the article better. We found an article about relation between modularity and lattices since we wanted to generate lattices in a systematic manner. In the article, there is a formula of lower bound for modularity of lattices according to given parameters. We tried to determine parameters for our lattice structure.

Meeting 5 - 31.10.2019: We showed our PSD to our advisor and he shared his opinions and gave advices to us.

Meeting 6 - 7.11.2019: We completed synthetic graph generator and showed this module to our advisor.

Meeting 7 - 21.11.2019: The article that proposes a new graph property, permanence, was analyzed. This graph property is used to detect community structures. The proposed MaxPerm and DyPerm algorithms were discussed. It was decided to ask for the code of MaxPerm from its author, since it is not available on the internet [13].

Meeting 8 - 28.11.2019: The code of DyPerm is publicly available. While we were waiting the code of MaxPerm, we tried to run the DyPerm code and made some observations. But the code did not work. There was a missing file to run the code. We could not figure out what kind of information the file contained, and documentation was not well enough. We shared our experiences with our advisor. We decided to modify the code in accordance of our problem.

Meeting 9 - 12.12.2019: The code of MaxPerm was obtained. We run the code on our graphs. The results were not as we expected. Thus, we decided to work more on the permanence.

Meeting 10 - 26.12.2019: We showed our graduation project presentation to our advisor. We got some feedbacks from our advisor.

6.3 Task Plan with Milestones

- Generate/modify decision logic
  Using RL task and D.C.D modules, a decision logic is derived.
- Implement a decision module for skill acquisition
  A module will be implemented using the derived decision logic.

- Test the algorithm with different inputs
  The system will be tested with different inputs. If results are not satisfactory, the decision logic will be modified.

These tasks will be proceeded in an iterative manner. The Gannt chart for the project time line is given in Figure 10.

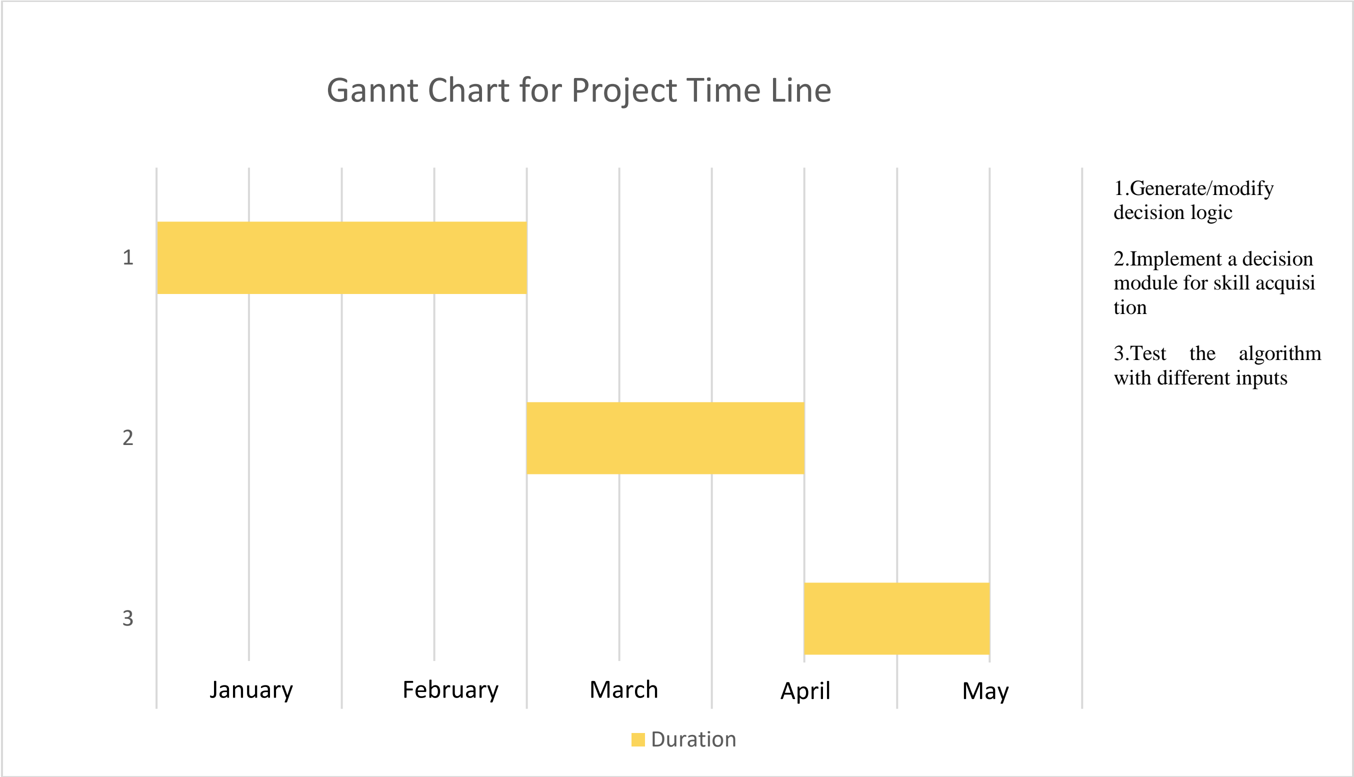

Figure 10 - Gannt Chart for Project Time Line

7. References

[1] Sutton, Richard S., Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." Artificial intelligence 112.1-2 (1999): 181-211.

[2] Şimşek, Özgür, and Andrew G. Barto. "Skill characterization based on betweenness." Advances in neural information processing systems. 2009.

[3] Davoodabadi, Marzieh, and Hamid Beigy. "A new method for discovering subgoals and constructing options in reinforcement learning." IICAI. 2011.

[4] BLONDEL, Vincent D., et al. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008, 2008.10: P10008.

[5] TRAAG, Vincent A.; WALTMAN, Ludo; VAN ECK, Nees Jan. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports*, 2019, 9.

[6] CORDEIRO, Mário; SARMENTO, Rui Portocarrero; GAMA, João. Dynamic community detection in evolving networks using locality modularity optimization. *Social Network Analysis and Mining*, 2016, 6.1: 15.

[7] ZHUANG, Di; CHANG, J. Morris; LI, Mingchen. DynaMo: Dynamic Modularity-based Community Detection in Evolving Social Networks. *arXiv preprint arXiv:1709.08350*, 2017.

[8] NEWMAN, Mark EJ; GIRVAN, Michelle. Finding and evaluating community structure in networks. *Physical review E*, 2004, 69.2: 026113.

[9] KAZEMITABAR, Seed Jalal; TAGHIZADEH, Nasrin; BEIGY, Hamid. A graph-theoretic approach toward autonomous skill acquisition in reinforcement learning. *Evolving Systems*, 2018, 9.3: 227-244.

[10] SHOELEH, Farzaneh; ASADPOUR, Masoud. Graph based skill acquisition and transfer learning for continuous reinforcement learning domains. *Pattern Recognition Letters*, 2017, 87: 104-116.

[11] XU, Xiao; YANG, Mei; LI, Ge. Constructing Temporally Extended Actions through Incremental Community Detection. *Computational intelligence and neuroscience*, 2018, 2018.

[12] Coşkun, Kutalmış, Aslancı, Ezdin, 2017, Using Hierarchies in Reinforcement Learning Framework with Non-Stationary Environments. B.S. thesis. Marmara University.

[13] CHAKRABORTY, Tanmoy, et al. On the permanence of vertices in network communities. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014. p. 1396-1405.

[14] AGARWAL, Prerna, et al. DyPerm: Maximizing permanence for dynamic community detection. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, Cham, 2018. p. 437-449.

[15] MENDONCA, Matheus RF; ZIVIANI, Artur; BARRETO, AndrÉ. Graph-Based Skill Acquisition For Reinforcement Learning. *ACM Computing Surveys (CSUR)*, 2019, 52.1: 6.

[16] DING, X. I. A. O.; LI, Yi-tong; CHUAN, S. H. I. Autonomic discovery of subgoals in hierarchical reinforcement learning. *The Journal of China Universities of Posts and Telecommunications*, 2014, 21.5: 94-104.

[17] MACHADO, Marios C.; BELLEMARE, Marc G.; BOWLING, Michael. A laplacian framework for option discovery in reinforcement learning. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017. p. 2295-2304.