강화학습을 통한 초순열의 최소 길이 탐색

(Shortest Super-permutation as the Haruhi Problem: A Reinforcement Learning Approach)

https://github.com/calpidisca/Reinforcement-Learning-Project

120250131 김범수

120250133 박성원

서강대학교

SOGANG UNIVERSITY

CONTENTS

SOGANG UNIVERSITY

프로젝트 주제 및 목표

- 초순열이란, 길이 N으로 만들 수 있는 **모든 순열이 연속된 형태로 한 번씩 등장하는 문자열**이다.

N=3인 경우로 예를 들면, 가능한 순열은 다음의 **3!(=6)**개이다.

({1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2}, {3,2,1})

초순열은 이 6개가 모두 포함된 하나의 문자열이다. 그 중 최소는 다음 9자리의 수열이다.

{123121321}

프로젝트 주제 및 목표

초순열은 $n = 1,2, \dots, 5$에 대해서 알려져 있다.

$$S_1 = 1$$
$$S_2 = 3$$
$$S_3 = 9$$
$$S_4 = 33$$
$$S_5 = 153$$
$$S_6 \leq 872$$

$n \geq 6$부터는 정확한 값은 모르며 upper bound와 lower bound만이 알려져 있다.

$n = 5$만 해도 **120**개의 순열이 생긴다.
이 순열들을 모두 넣되 가능한 짧은 문자열을 만드는 문제는

- 조합적으로 **매우 빠르게 복잡**해지고,
- 일반적인 **최적해는 알려져 있지 않**고,
- 관련 논문에서도 **아직 풀리지 않은 문제**다.

따라서, 완전탐색 대신 **RL** 과 같은 **학습 기반 휴리스틱** 접근이 의미 있게 시도될 수 있다.

프로젝트 주제 및 목표

본 프로젝트의 목표는, 강화 학습을 이용하여
특정 N에서 “모든 순열을 포함하면서 길이는 최소”가 되도록 수열 생성 정책을 학습하여

초순열을 생성하는 방법을 구하는 것이다.

S_n for $n = 3,4,5$ 에 대해 RL를 사용하여 알려진 초순열의 길이에 도달할 수 있는지
혹은 그와 비슷하게 접근가능한지 확인하고자 한다.

S_3 : 123121321
S_4 : 123412314231243121342132413214321
S_5 : 12345123415234125341235412314523142531423514231542312453124351243152431254312134521342513421534-
13542132451324153241352413254132145321435214325143215432 1

SOGANG UNIVERSITY

실험 환경

- Python 3.13.9
- gymnasium>=0.29.0
- stable-baselines3>=2.0.0
- sb3-contrib>=2.0.0
- numpy>=1.24.0
- pandas>=2.0.0
- torch with CUDA 12.4:
- tqdm>=4.65.0
- tensorboard>=2.14.0

실행환경: i7-14700 RTX4070/colab L4 GPU

실험 설정

매 timestep에서 **agent**는 {1,⋯,n} 중 하나의 숫자 혹은 수열**(action)**를 선택해 문자열 뒤에 붙인다.
새로 생긴 문자열의 형태로 reward와 penalty를 계산해서 학습을 진행한다.

$n = 3,4,5$의 경우에 대해 PPO, A2C의 방식으로 학습을 진행해보고
Random policy, Greedy heuristic 방식과 비교한다.

두가지 설계를 사용한다.
Symbolic: 하나의 숫자를 선택하는 설계
마지막에 붙일 하나의 숫자를 고르는 Action
Word_cost: 하나의 순열를 선택하는 설계
ex) $\{1,2,\dots,5\}$으로 구성된 12345324132541325 가 기존의 수열이 있을 때,
하나의 순열 32514을 골라 기존의 수열에 최대로 겹치게 붙임
12345324132541325 + 32514 => 123453241325413251 4

실험 설정_환경 설계

현재까지의 문자열을 관리

새로 만들어진 길이 n의 부분 문자열이 **새로운 순열인지** 판별

모든 순열을 포함했는지, 최대 길이를 초과했는지 체크

현재 길이와 최대길이의 비율을 관리

Symbolic: Action으로 하나의 숫자를 선택하는 설계
Word_cost: Action으로 하나의 순열를 선택하는 설계

Symbolic 은 Action공간이 작으며 가장 최소의 변화 단위로 설계
Word_cost 은 순열을 선택하여 최대한 겹치게되는 방식으로 수열을 변화. 초순열에 대한 기존 연구의 접근 방식에서 따와 설계

실험 설정_상태 설계

Symbolic
[마지막 n-1글자: 나온순열목록 : 최대대비길이비율]

$[A_{\{(T-n+1):T-1\}}: B[0:n!] : r_{T-1}]$

상태의 차원수 :$n-1+n!+1=n!+n$
$A_{0:T}$: T까지의 생성된 수열
$B[i]$: i번째 순열이 나왔는지 표시하는 원핫벡터
$r_T = A_T/(Max\ length)$

Word_cost
[나온순열: 코스트벡터 :길이비율]

$[B[0:n!]: C_T[0:n!] : r_{T-1}]$

상태의 차원수 :$n!+n!+1=2(n!)+1$
B $[i]$: i번째 순열이 나왔는지 표시하는 원핫벡터
$r_T = A_T/(Max\ length)$
$C_T[i]$: i번째 순열이 T에서의 수열에 추가될 때 거리

마지막 수열의 정보가 아닌
직접적으로 거리(코스트)를 상태로 하여
학습에 도움이 되게 설정

환경 및 데이터셋 설명_**행동 설계**

Symbolic: 하나의 숫자를 선택하는 설계
마지막에 붙일 하나의 숫자를 고르는 Action

$i \in \{1,2,\dots,n\}$으로 고르는 것이므로 n차원

Word_cost: 하나의 순열를 선택하는 설계
ex) $\{1,2,\dots,5\}$으로 구성된
12345324132541325 가 기존의 수열이 있을 때,

하나의 순열 32514을 골라 기존의 수열에 최대로 겹치게 붙임

12345324132541325 + 32514 => 12345324132541325514

순열 51432을 골랐을 경우

12345324132541325 + 51432 =>
12345324132541325 1432

$i \in S_{\{1,2,\dots,n\}}$으로 고르는 것이므로 n!차원

환경 및 데이터셋 설명_보상함수 설계

Symbolic

Reward : $r(t) = \alpha \cdot \Delta N_t - \lambda$

최종 보상

$r_{final} = R_{goal}$

$R_{goal} = 50$

$\lambda = 0.1$

$\alpha = 1.0$

ΔN_t: 그 스텝에서 새로 발견된 순열

Word_cost

Reward : $r(t) = \alpha \cdot \Delta N_t - \lambda \cdot$ 증가한 길이

최종 보상

$r_{final} = R_{goal}$

$R_{goal} = 50$

$\lambda = 0.1$

$\alpha = 1.0$

ΔN_t: 그 스텝에서 새로 발견된 순열 수

설계 의도

새 순열을 많이, 빠르게 찾도록 유도

문자열을 짧게 만들수록 유리

목표를 달성하면 큰 보상, 단 길이가 길면 페널티

환경 및 데이터셋 설명_에피소드 종료 조건

성공 조건

모든 순열이 최소 한 번씩 등장 → coverage ratio = 1

실패 조건

최종 문자열 길이 $L_T > n! \cdot \frac{n}{2}$

만약 초과한다면 실패로 간주하고 에피소드 강제 종료

이유 :
무한히 길어지는 시도를 방지
실패 비율로 난이도, 학습 상태를 진단
S_n의 잘 알려진 upper bound는 $n! + (n-1)! + (n-2)! + (n-3)! + n - 3$이다.
$n = 3,4,5$ 정도에서는 충분하다고 판단하였다.

환경 및 데이터셋 설명_Baseline 정책

Random Policy

매 step에서 action을 균일한 확률로 샘플링

Greedy Heuristic

현재 state에서 새 순열을 가장 많이 만들 수 있는 action 선택

역할 :

강화학습 알고리즘과 비교할 기준선
작은 n에서는 baseline과 RL의 성능 차이를 명확하게 보여주기 위함

강화학습 알고리즘 선택

N! 크기의 상태 공간, 액션 공간이 나오며 sequence를 만드는것이라 순서도 영향을 끼치며 최종 길이에 도달해 Goal in점수를 받기까지 행동에 대한 reward가 지연된다.

DQN: 행동/상태 공간이 넓어 학습이 어려울거라 생각해 배제

A2C/PPO: 확률적 탐색으로 넓은 탐, critic의 존재로 지연 보상 해결
특히 PPO는 안정적인 학습의 장점을 주목하였음.
A2C의 경우 불안정한 학습에 우려

Sequence를 만드는 학습이므로 policy 기반 알고리즘이 우위일 것이라 예측

강화학습 알고리즘 선택_Hyperparameter

```
 "ppo": {
      "learning_rate": [3e-4, 1e-4],
      "gamma": [0.99, 0.995],
   },
"a2c": {
      "learning_rate": [7e-4],
      "gamma": [0.99, 0.995],
   },
BASELINE_EPISODES: Dict[str, int] = {
   "random": 100,
   "greedy": 100,
}
```

SEEDS: List[int] = [0, 1, 2, 42, 100, 123, 999]

실험한 n값 $(n = 3, 4, 5, 6)$

```
TIMESTEPS: Dict[str, int] = {
   "ppo": 300_000,
   "a2c": 300_000,
}
```

평가 지표 정리

final_length : 에피소드가 끝났을 때 초순열의 최종 길이
Success : 목표 조건을 만족했는지 (성공: 1, 실패: 0)
coverage_ratio : 덮은 순열의 비율 (#covered perms/n!)
episode_steps : 해당 에피소드에서 실행한 액션(스텝) 수
episode_return : 한 에피소드 동안 받은 총 보상
duplicate_action_ratio : 에피소드 동안 중복된 액션의 비율
new_perms_total : 새롭게 발견한 순열 개수(누적 또는 에피소드 기준)
best_final_length_so_far : 지금까지 관측된 최단 초순열 길이(시작 inf)

가장 짧은 3개 초순열 : 각 설정마다 top-3 초순열과 해당 길이, return 기록

결과 개요

N=3의 경우 너무 간단한 구조(길이 9)로 쉽게 수렴한다.

N=4의 경우 수렴하긴 하지만 최소값인 33까지 도달하는 경우는 드물다.

N=5,6의 경우에는 수열이 완성되지 않고 최대길이에 도달해 멈추는 경우만 발생한다.

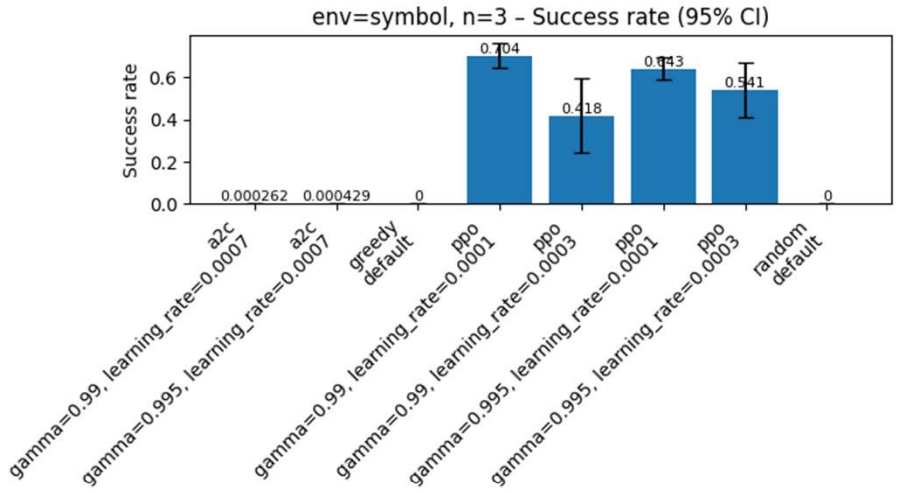
그래서 N=5의 경우에 대해서 learning rate, time_step, λ, α 를 수정하고 끝에 도달하지 못한 경우 감점을 부여하고 다시 실험했다.

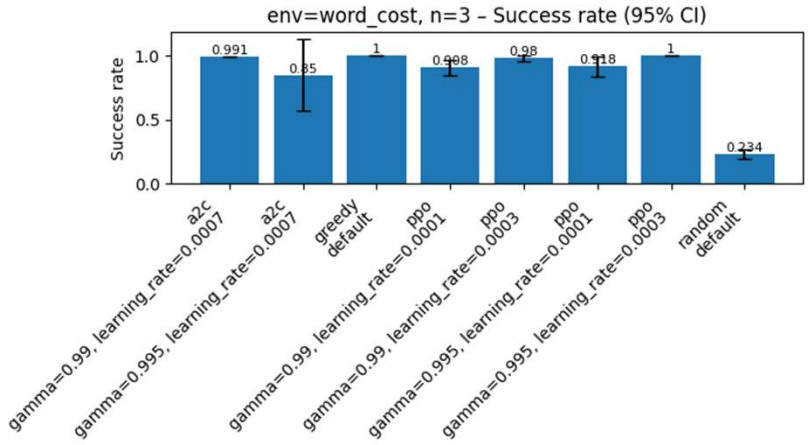
word_cost모델은 학습이 안되었지만 symbolic 은 학습이 되는 모습을 보였다.

이 경우 모델이 길이 감점을 피하기 위해 1234 2341 3412 4123 1234 형태가 반복되는 형태로 학습하게 된다고 판단하여 중복해서 감점 회피를 막기 위해 masking PPO로 테스트해보았다.

Symbolic의 경우 수렴하였기에 그대로 N=6의 경우를 실험했다.

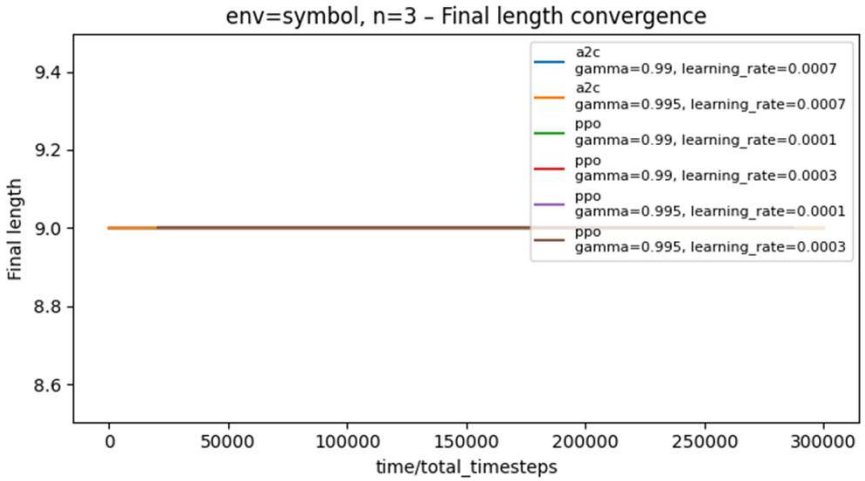
결과_ n=3

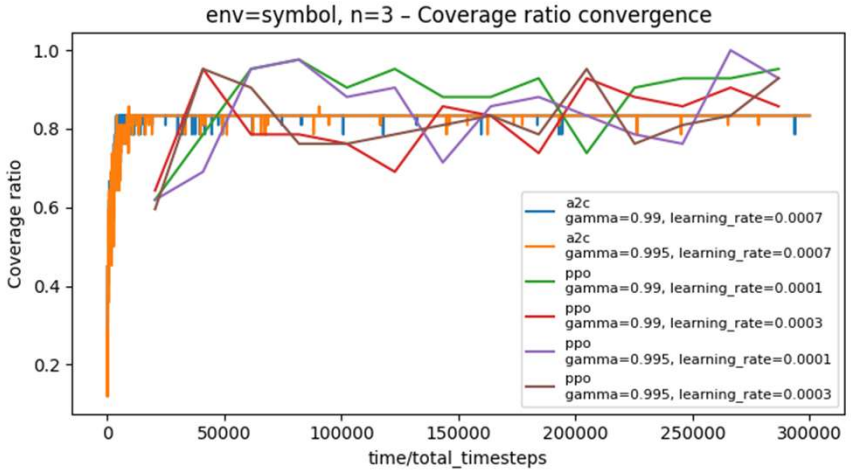


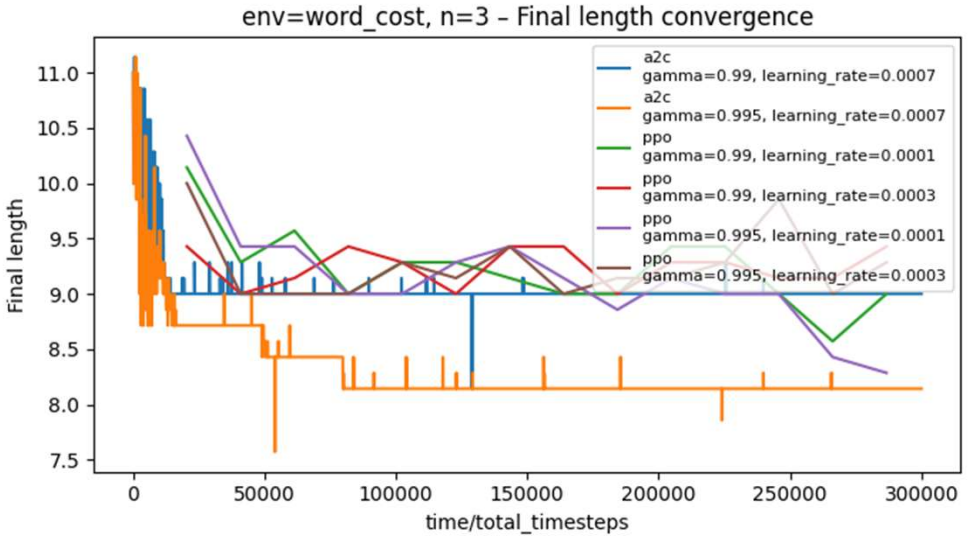


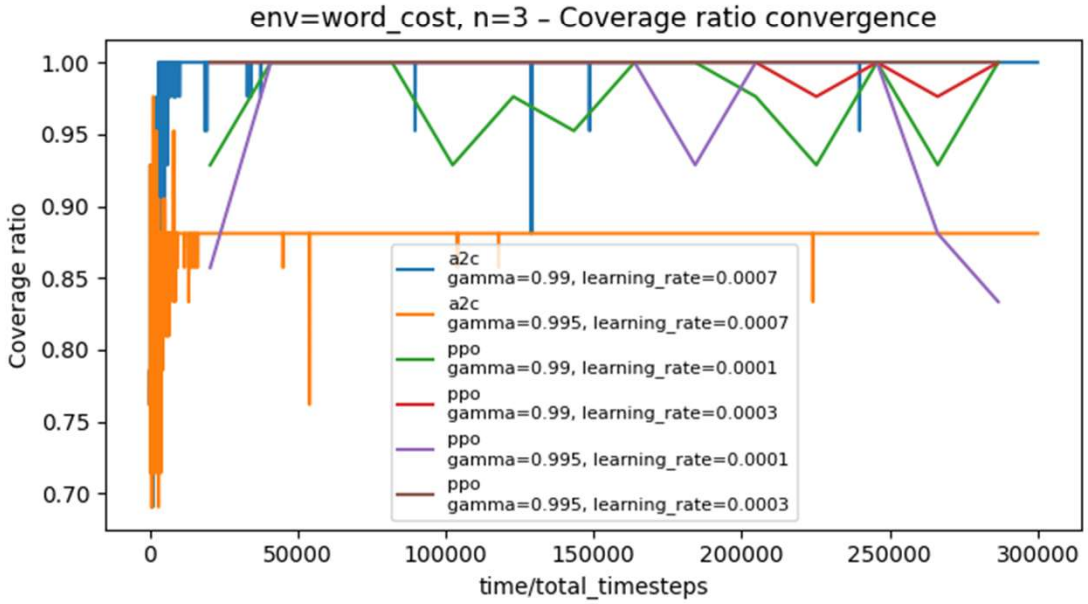
env_type	algo_name	Mean coverage ratio	Mean episode return	Mean episode steps	Mean final length	Success rate
symbol	a2c	0.829	4.089	9	9	0
symbol	a2c	0.829	4.094	9	9	0
symbol	greedy	0	-0.9	9	9	0
symbol	ppo	0.881	39.59	9	9	0.704
symbol	ppo	0.82	24.937	9	9	0.418
symbol	ppo	0.842	36.294	9	9	0.643
symbol	ppo	0.82	31.059	9	9	0.541
symbol	random	0.245	0.57	9	9	0
word_cost	a2c	0.998	46.507	4.029	9.043	0.991
word_cost	a2c	0.88	39.458	6.011	8.298	0.85
word_cost	greedy	1	47	6	9	1
word_cost	ppo	0.973	42.02	4.286	9.224	0.908
word_cost	ppo	0.997	45.714	4.122	9.245	0.98
word_cost	ppo	0.964	42.571	4.653	9.133	0.918
word_cost	ppo	1	46.745	4.306	9.255	1
word_cost	random	0.803	5.906	5.151	10.624	0.234

결과_ n=3


env=symbol, n=3 – Final length convergence
Final length
time/total_timesteps
a2c gamma=0.99, learning_rate=0.0007
a2c gamma=0.995, learning_rate=0.0007
ppo gamma=0.99, learning_rate=0.0001
ppo gamma=0.99, learning_rate=0.0003
ppo gamma=0.995, learning_rate=0.0001
ppo gamma=0.995, learning_rate=0.0003


env=symbol, n=3 – Coverage ratio convergence
Coverage ratio
time/total_timesteps
a2c gamma=0.99, learning_rate=0.0007
a2c gamma=0.995, learning_rate=0.0007
ppo gamma=0.99, learning_rate=0.0001
ppo gamma=0.99, learning_rate=0.0003
ppo gamma=0.995, learning_rate=0.0001
ppo gamma=0.995, learning_rate=0.0003

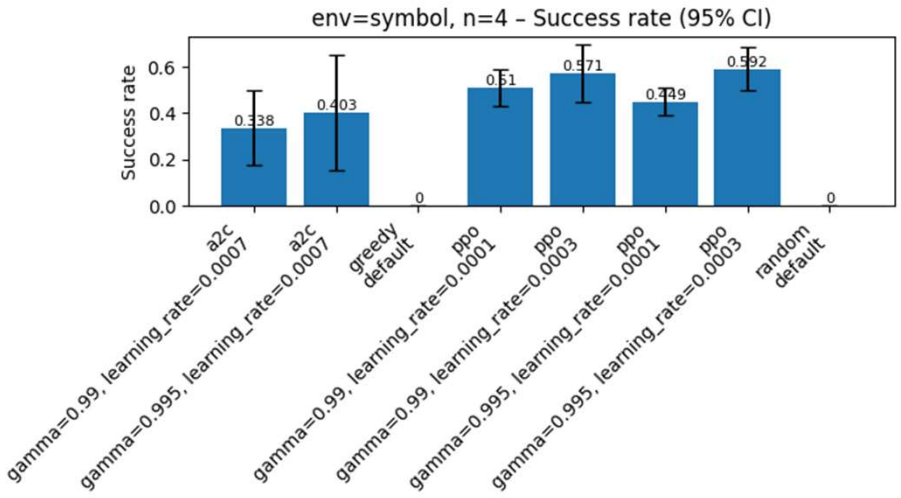

env=word_cost, n=3 – Final length convergence
Final length
time/total_timesteps
a2c gamma=0.99, learning_rate=0.0007
a2c gamma=0.995, learning_rate=0.0007
ppo gamma=0.99, learning_rate=0.0001
ppo gamma=0.99, learning_rate=0.0003
ppo gamma=0.995, learning_rate=0.0001
ppo gamma=0.995, learning_rate=0.0003

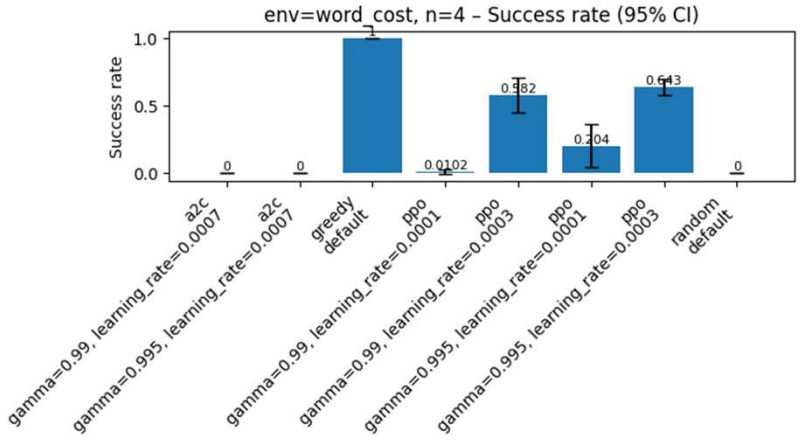

env=word_cost, n=3 – Coverage ratio convergence
Coverage ratio
time/total_timesteps
a2c gamma=0.99, learning_rate=0.0007
a2c gamma=0.995, learning_rate=0.0007
ppo gamma=0.99, learning_rate=0.0001
ppo gamma=0.99, learning_rate=0.0003
ppo gamma=0.995, learning_rate=0.0001
ppo gamma=0.995, learning_rate=0.0003

결과_ n=3

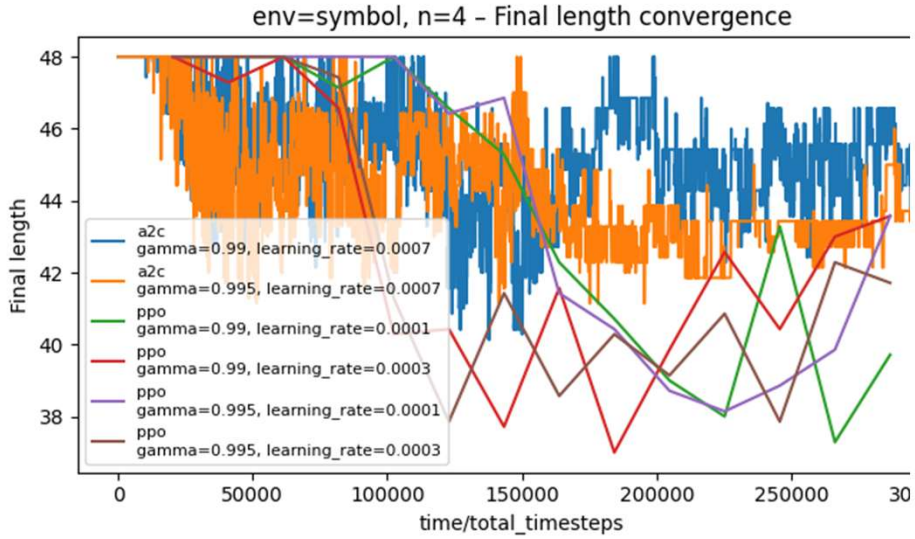
N=3의 경우 symbolic보다 word 기반 방식이 더 성적이 좋았으며
PPO가 A2C보다 높은 경향을 보인다.
성공/실패의 기준이 주어진 최대길이 (N/2)*n! 보단 작으며 모든 순열이 나타날 경우 성공이다.
Word에서 Greedy 방식이 성공률이 매우 높은 것을 보면 단순한 구조에 새로운 순열을 하나하나 넣는것이 높은 전략이라는 뜻이다.
다만 symbolic의 greedy는 매우 낮은 성공률을 보인것을 보아 에피소드가 짧아 문자로 순열을 만드는 것이 학습외지 못한 것으로 추측된다.
커버리지 범위가 a2c의 경우 더 이상 커지지 못하고 있는데 이것은 한번도 성공하지 못해 성공 reward 를 학습하지 못한 것으로 보인다.

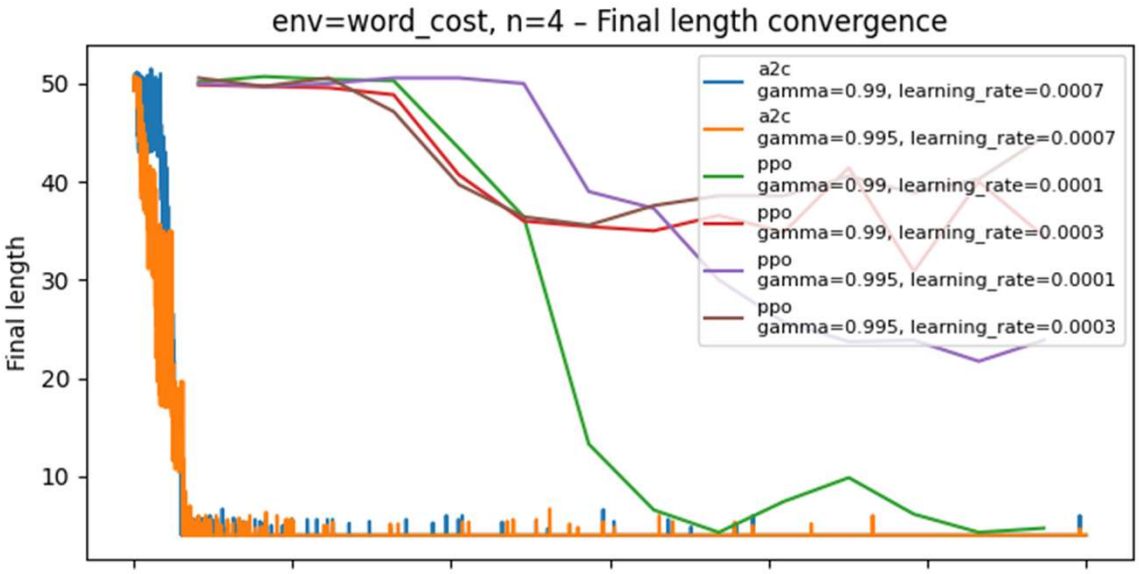
결과_ n=4

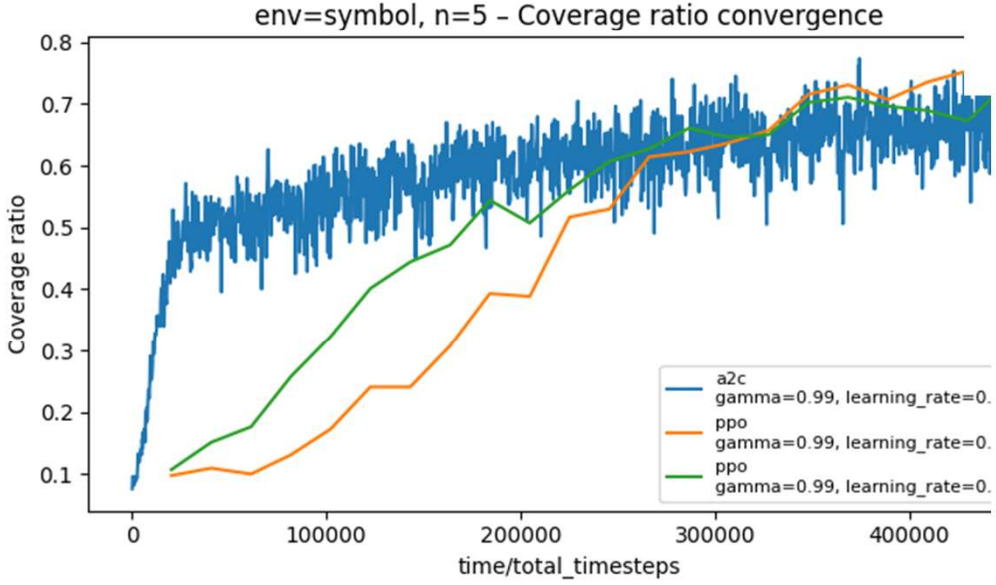


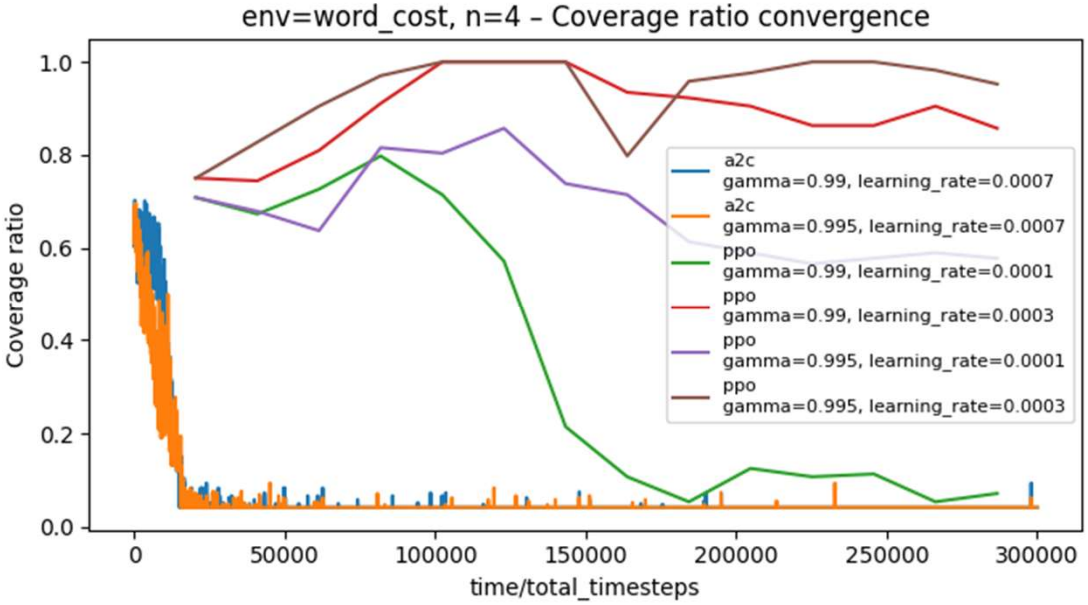


env_type	algo_name	Mean coverage ratio	Mean episode return	Mean episode steps	Mean final length	Success rate
symbol	a2c	0.887	33.645	45.3	45.3	0.338
symbol	a2c	0.89	37.11	44.218	44.218	0.403
symbol	greedy	0	-4.8	48	48	0
symbol	ppo	0.898	42.705	43.663	43.663	0.51
symbol	ppo	0.923	46.465	42.592	42.592	0.571
symbol	ppo	0.855	38.582	43.878	43.878	0.449
symbol	ppo	0.895	46.838	42.337	42.337	0.592
symbol	random	0.16	-0.959	48	48	0
word_cost	a2c	0.07	-4.521	69.446	6.196	0
word_cost	a2c	0.062	-4.121	70.187	5.605	0
word_cost	greedy	1	41	24	33	1
word_cost	ppo	0.36	-15	51.531	24.143	0.01
word_cost	ppo	0.89	10.214	19.939	40.235	0.582
word_cost	ppo	0.676	-11.143	30.235	37.571	0.204
word_cost	ppo	0.937	12.582	16.949	42.051	0.643
word_cost	random	0.679	-34.024	15.037	50.327	0


env=symbol, n=4 – Final length convergence
Final length
time/total_timesteps
a2c
gamma=0.99, learning_rate=0.0007
a2c
gamma=0.995, learning_rate=0.0007
ppo
gamma=0.99, learning_rate=0.0001
ppo
gamma=0.99, learning_rate=0.0003
ppo
gamma=0.995, learning_rate=0.0001
ppo
gamma=0.995, learning_rate=0.0003


env=word_cost, n=4 – Final length convergence
Final length
a2c
gamma=0.99, learning_rate=0.0007
a2c
gamma=0.995, learning_rate=0.0007
ppo
gamma=0.99, learning_rate=0.0001
ppo
gamma=0.99, learning_rate=0.0003
ppo
gamma=0.995, learning_rate=0.0001
ppo
gamma=0.995, learning_rate=0.0003

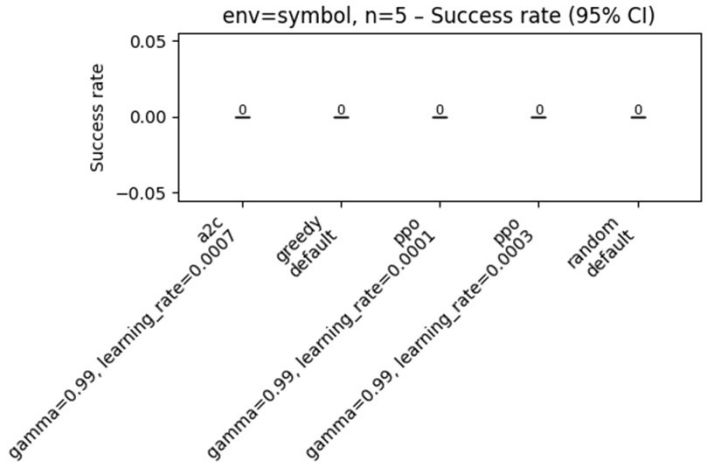

env=symbol, n=5 – Coverage ratio convergence
Coverage ratio
time/total_timesteps
a2c
gamma=0.99, learning_rate=0.
ppo
gamma=0.99, learning_rate=0.
ppo
gamma=0.99, learning_rate=0.

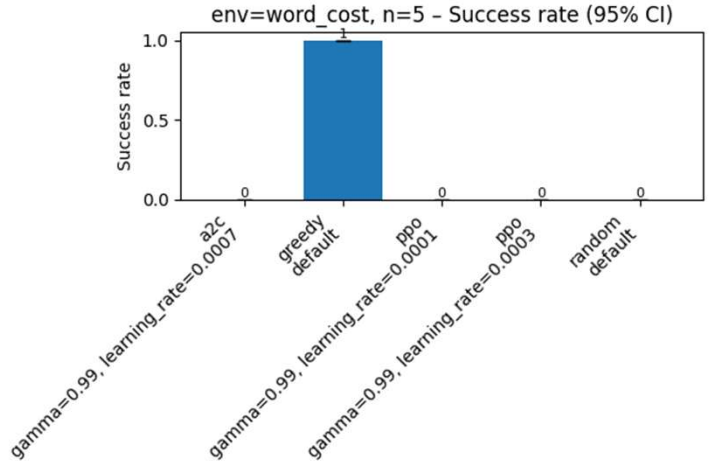

env=word_cost, n=4 – Coverage ratio convergence
Coverage ratio
time/total_timesteps
a2c
gamma=0.99, learning_rate=0.0007
a2c
gamma=0.995, learning_rate=0.0007
ppo
gamma=0.99, learning_rate=0.0001
ppo
gamma=0.99, learning_rate=0.0003
ppo
gamma=0.995, learning_rate=0.0001
ppo
gamma=0.995, learning_rate=0.0003

결과_ n=4

Greedy의 경우는 n=3와 동일하나 a2c의 경우는 정 반대 상황이 되었다.
33길이의 초순열을 만드는데 있어서 9개였을 때보다 복잡해지고 길어져
Word방식은 전반적으로 낮아지고 symbolic의 경우는 크게 변하지는 않았다.
또한 커버리지가 word에선 오히려 낮아지는 경향을 보여
A2C가 잘못된 학습을 했을때(감점을 회피하려 중복되는 순열만 선택할경우) local minimum에 갇히거나 실수를 학습한 영향이 큰것으로 보인다.

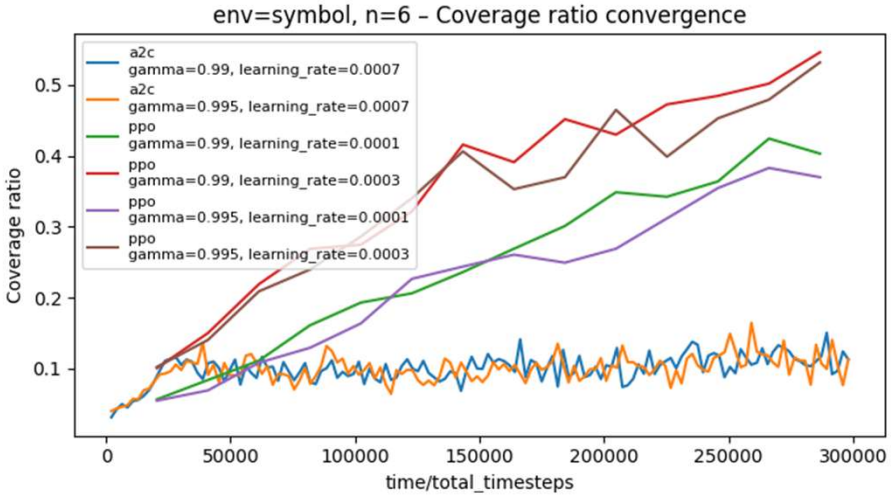
결과_ n=5

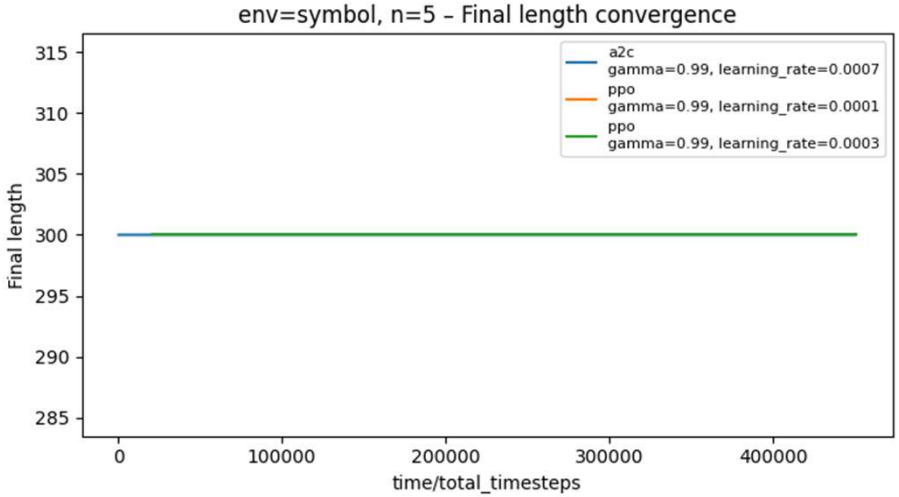


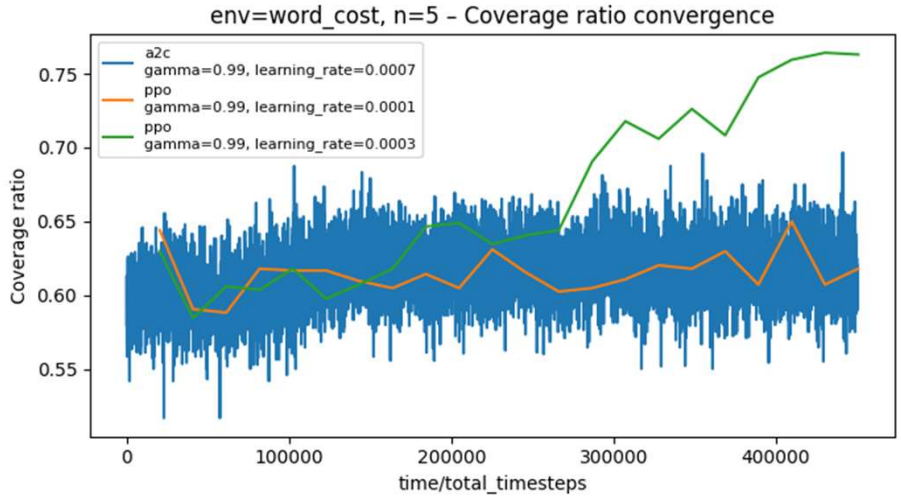


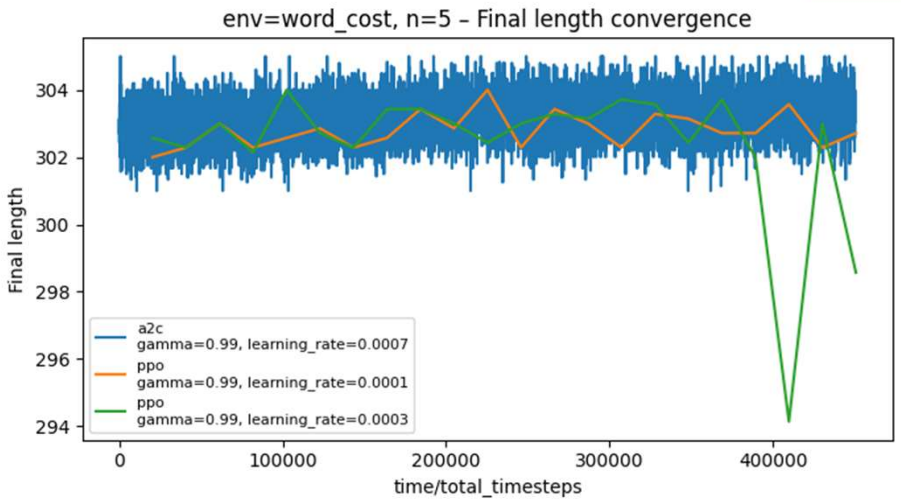
env_type	algo_name	Mean coverage ratio	Mean episode return	Mean episode steps	Mean final length	Success rate
symbol	a2c	0.587	249.273	300	300	0
symbol	greedy	0	-103	300	300	0
symbol	ppo	0.46	173.234	300	300	0
symbol	ppo	0.515	206.253	300	300	0
symbol	random	0.091	-48.586	300	300	0
word_cost	a2c	0.614	207.812	64.301	303.182	0
word_cost	greedy	1	769.4	120	153	1
word_cost	ppo	0.615	208.206	65.701	302.799	0
word_cost	ppo	0.666	239.399	76.929	302.357	0
word_cost	random	0.616	209.067	66.071	302.89	0

결과_ n=5


env=symbol, n=6 – Coverage ratio convergence
a2c
gamma=0.99, learning_rate=0.0007
a2c
gamma=0.995, learning_rate=0.0007
ppo
gamma=0.99, learning_rate=0.0001
ppo
gamma=0.99, learning_rate=0.0003
ppo
gamma=0.995, learning_rate=0.0001
ppo
gamma=0.995, learning_rate=0.0003
Coverage ratio
0.1
0.2
0.3
0.4
0.5
0
50000
100000
150000
200000
250000
300000
time/total_timesteps


env=symbol, n=5 – Final length convergence
a2c
gamma=0.99, learning_rate=0.0007
ppo
gamma=0.99, learning_rate=0.0001
ppo
gamma=0.99, learning_rate=0.0003
Final length
285
290
295
300
305
310
315
0
100000
200000
300000
400000
time/total_timesteps


env=word_cost, n=5 – Coverage ratio convergence
a2c
gamma=0.99, learning_rate=0.0007
ppo
gamma=0.99, learning_rate=0.0001
ppo
gamma=0.99, learning_rate=0.0003
Coverage ratio
0.55
0.60
0.65
0.70
0.75
0
100000
200000
300000
400000
time/total_timesteps


env=word_cost, n=5 – Final length convergence
a2c
gamma=0.99, learning_rate=0.0007
ppo
gamma=0.99, learning_rate=0.0001
ppo
gamma=0.99, learning_rate=0.0003
Final length
294
296
298
300
302
304
0
100000
200000
300000
400000
time/total_timesteps

결과_ n=5

이번 경우는 word의 그리드 방식은 성공률(최대길이 이내에 모든 순열 등장확률)은 1로 매우 높은 것에 비해 다른 모든 방식은 0을 기록했다.

이것은 그리드 방식이 word를 추가하며 최대 이득을 위해 모든 순열을 넣는 방식으로 최소 길이인 153에 도달하였다는 것이다. Symbolic이 도달하지 못한 것은 단어를 완성시키기에는 학습이 짧았던 것으로 추측된다

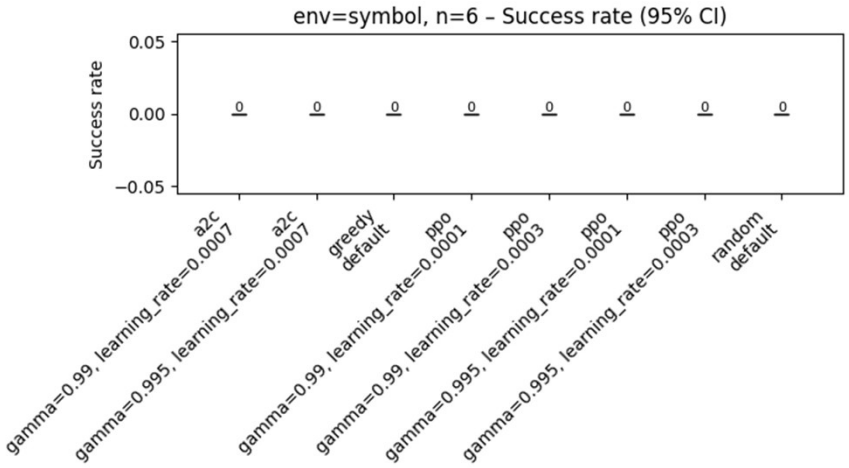
A2C는 학습으로 값이 거의 변하지 않아 너무 학습이 짧은 것일 수 있으며, 현재 문제에 적합하지 않다고 판단하였다.

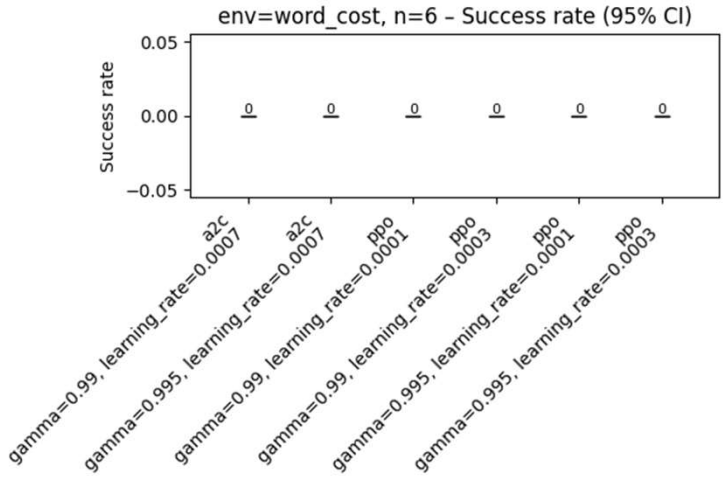
혹은 N=5의 경우 초순열이 유일하지 않은 영향일수도 있겠다는 가능성도 있다.

다만 PPO는 상승하는 경향을 보여 추가적으로 파라미터를 조정해 실험한다. 이는 적극적으로 탐사하지 않는것이라 판단해 이하로 수정하였다.

러닝레이트: 2e-4/ seed 0, 42,100,타임스텝:2200000, $\lambda = 0.01$, $\alpha = 5$, truncated 할 시 reward-=100

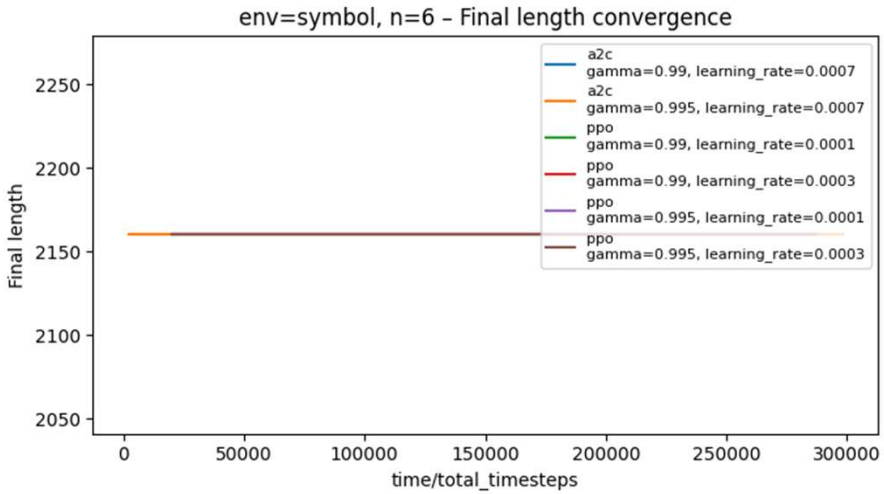
결과_ n=6

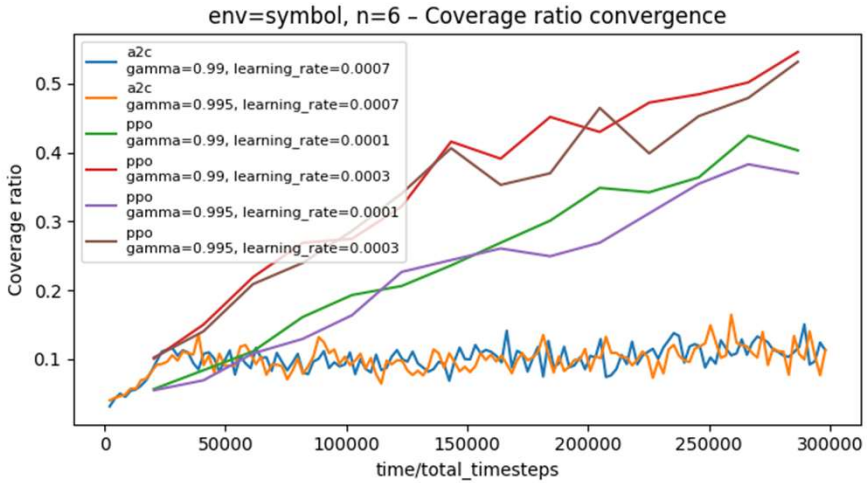


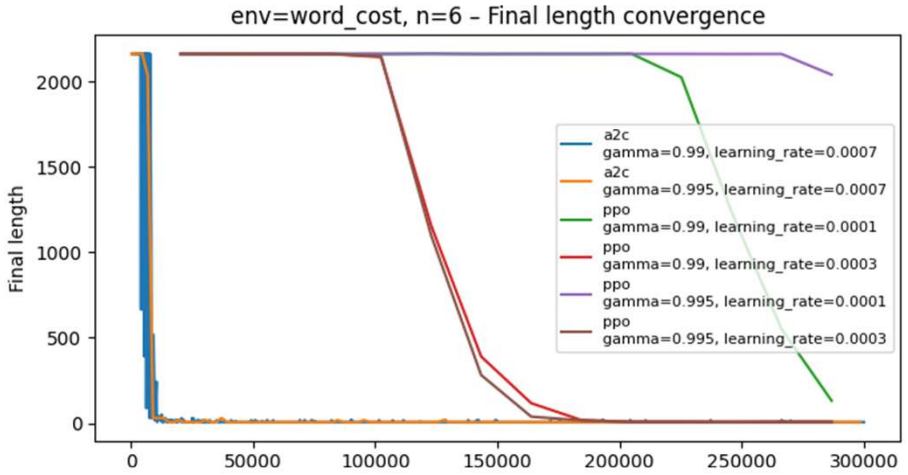


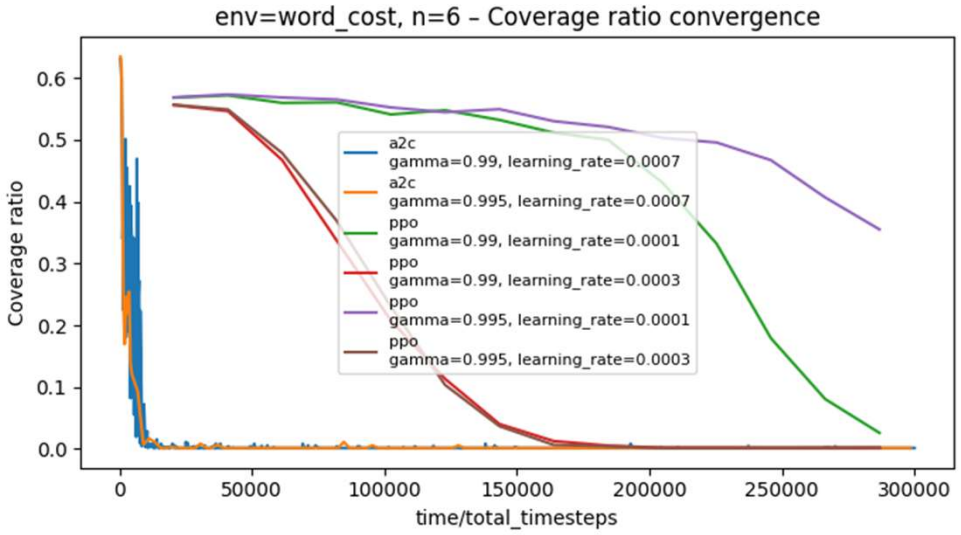
env_type	algo_name	Mean coverage ratio	Mean episode return	Mean episode steps	Mean final length	Success rate
symbol	a2c	0.1	-144.277	2160	2160	0
symbol	a2c	0.098	-145.088	2160	2160	0
symbol	greedy	0	-216	2160	2160	0
symbol	ppo	0.25	-36.092	2160	2160	0
symbol	ppo	0.359	42.582	2160	2160	0
symbol	ppo	0.228	-51.878	2160	2160	0
symbol	ppo	0.341	29.51	2160	2160	0
symbol	random	0.045	-183.477	2160	2160	0
word_cost	a2c	0.033	-167.622	2014.32	191.39	0
word_cost	a2c	0.025	-164.435	2030.34	182.354	0
word_cost	ppo	0.425	-1521.8	789.694	1827.47	0
word_cost	ppo	0.164	-775.867	1526.11	893.837	0
word_cost	ppo	0.515	-1784.34	435.929	2154.82	0
word_cost	ppo	0.167	-756.143	1536.66	876.633	0

결과_ n=6


env=symbol, n=6 – Final length convergence
Final length
time/total_timesteps
a2c gamma=0.99, learning_rate=0.0007
a2c gamma=0.995, learning_rate=0.0007
ppo gamma=0.99, learning_rate=0.0001
ppo gamma=0.99, learning_rate=0.0003
ppo gamma=0.995, learning_rate=0.0001
ppo gamma=0.995, learning_rate=0.0003


env=symbol, n=6 – Coverage ratio convergence
Coverage ratio
time/total_timesteps
a2c gamma=0.99, learning_rate=0.0007
a2c gamma=0.995, learning_rate=0.0007
ppo gamma=0.99, learning_rate=0.0001
ppo gamma=0.99, learning_rate=0.0003
ppo gamma=0.995, learning_rate=0.0001
ppo gamma=0.995, learning_rate=0.0003


env=word_cost, n=6 – Final length convergence
Final length
a2c gamma=0.99, learning_rate=0.0007
a2c gamma=0.995, learning_rate=0.0007
ppo gamma=0.99, learning_rate=0.0001
ppo gamma=0.99, learning_rate=0.0003
ppo gamma=0.995, learning_rate=0.0001
ppo gamma=0.995, learning_rate=0.0003


env=word_cost, n=6 – Coverage ratio convergence
Coverage ratio
time/total_timesteps
a2c gamma=0.99, learning_rate=0.0007
a2c gamma=0.995, learning_rate=0.0007
ppo gamma=0.99, learning_rate=0.0001
ppo gamma=0.99, learning_rate=0.0003
ppo gamma=0.995, learning_rate=0.0001
ppo gamma=0.995, learning_rate=0.0003

결과_ n=6

이 경우 모든 알고리즘이 한번도 성공하지 못했다. 870개 정도의 긴 초순열이라 단순히 greedy로도 접근하지 못한 것이다.

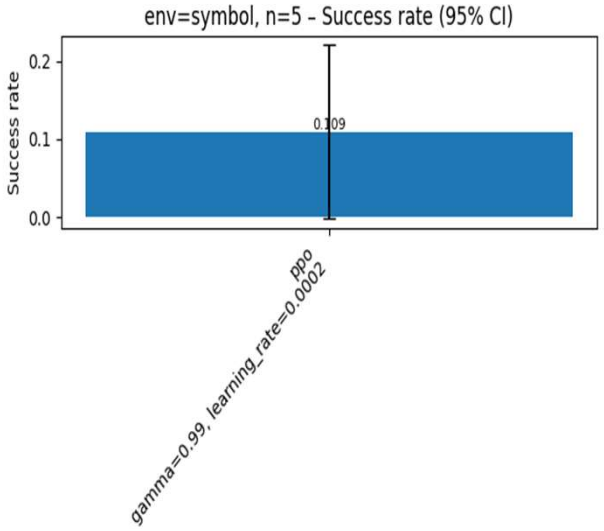
다만 여기서 PPO와 A2C의 차이점이 드러난다.

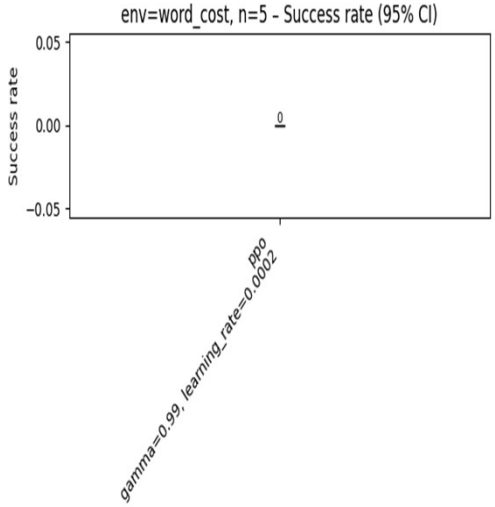
PPO는 커버리지를 넓혀가며 최종길이를 줄이지만 A2C는 커버리지를 줄여가며 길이를 줄인다. 이는 A2C가 그저 동일한 이미 본 순열들(1234, 2341, 3412, 4123)만 반복해서 선택하는 정책을 학습했다고 볼 수 있다. 감점을 피하기 위한 액션만 하게 된다고 볼 수 있다. 이는 n이 커질수록 처음 순환하는 동안 점수를 얻는 구간이 길어져 다른 시도를 안하게된다고 볼 수 있다. 이 경우 성공 리턴을 얻기 매우 힘들어지고 리워드가 낮아지게 된다. 이런 특성은 단어기반에서 더 큰것으로 보인다. PPO의 경우는 symbolic에서는 커버리지가 증가하지만 word 방식에서는 감소한다. 이는 word로 액션하는 행위가 위에 말한 것을 더 하기 쉽게하는것으로도 보이며 symbolic은 탐색공간도 작아 탐사하기 쉬운것이라고 생각된다. 이는 높은 N에서는 symbolic이 선택공간 측면에서도 더 좋을것이라는 걸 예측할수 있다. 다만 최종 길이가 모두 동일한 것은 초순열을 만들기 전에 최대 길이에 도달한것으로 생각된다.

그러므로 Word방식은 동일할 선택을 막는 강제성을 두고 실험할 것이다. 실험(n=5_masking PPO word_cost)

또한 symbolic방식은 길이를 늘리고 리워드를 수정하여 실험한다. 실험 (n=6_리워드 개선 symbolic)

결과_ n=5_reward 개선

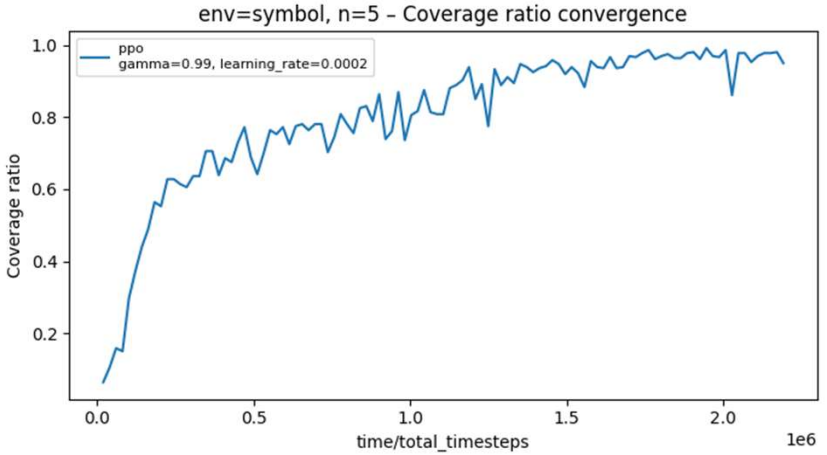


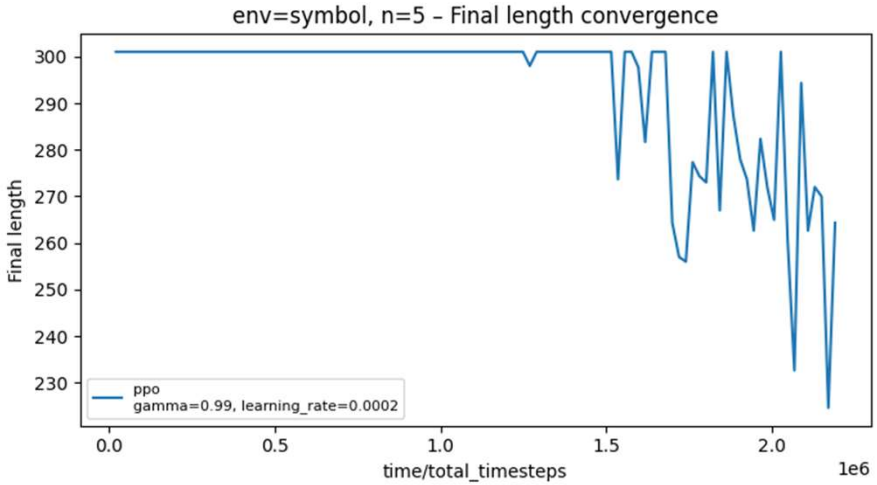


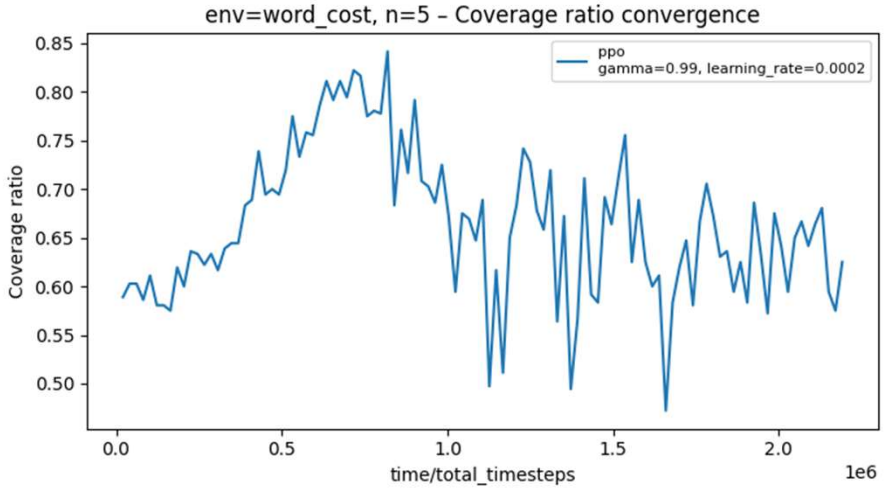
env_type	algo_name	Mean coverage ratio	Mean episode return	Mean episode steps	Mean final length	Success rate
symbol	ppo	0.802	410.835	293.489	293.489	0.109
word_cost	ppo	0.664	244.336	480.966	270.221	0

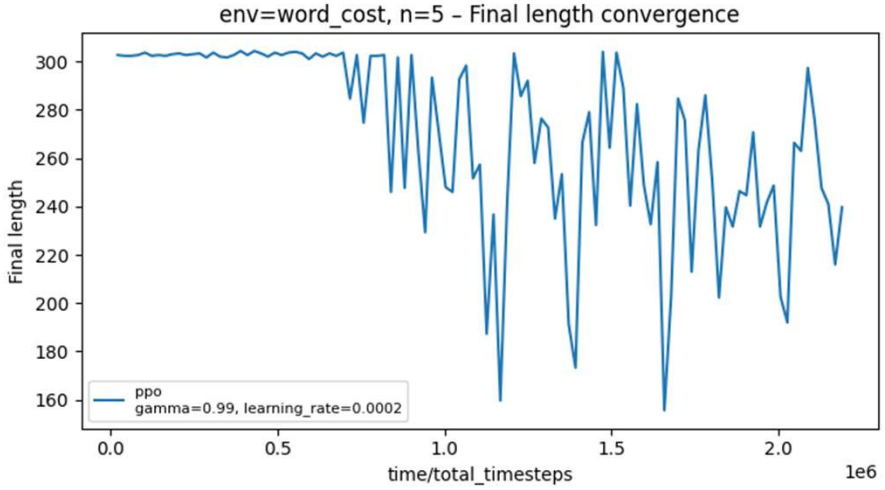
SOGANG UNIVERSITY

결과_ n=5_reward 개선





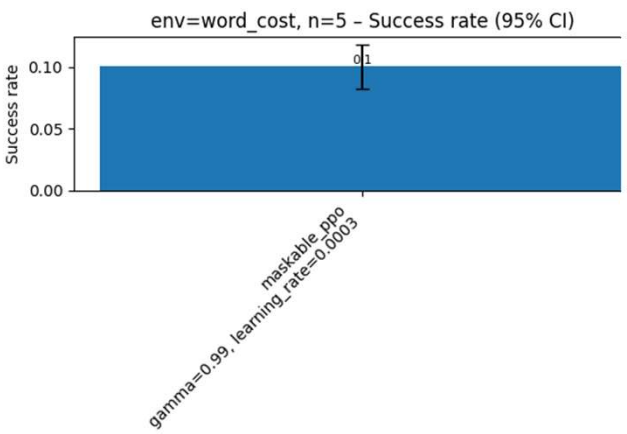


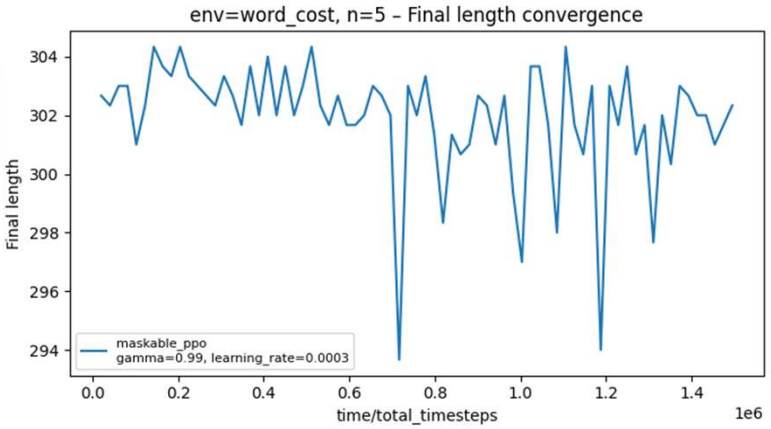


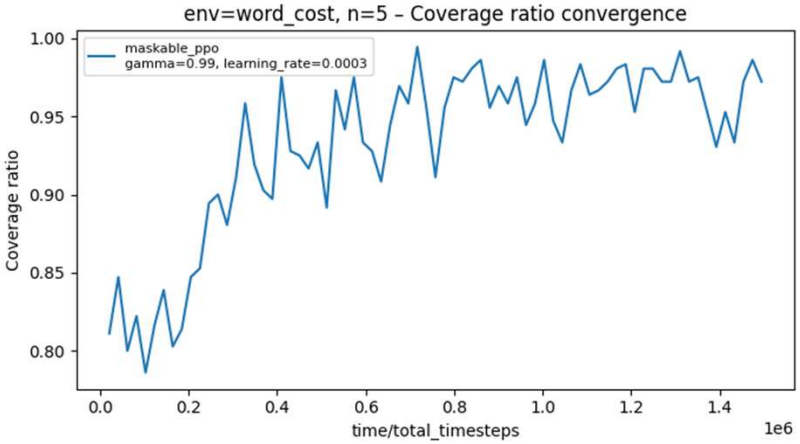
결과_ n=5_reward 개선

스텝을 늘리고 truncted에 심한 감점을 두었으나 word 방식에서 여전히 동일하게 반복하는 듯한 그래프 형태를 보여준다. Symbolic의 경우 커버리지를 늘려가며 길이를 줄여 성공 사례도 존재한다. 이 실험에서는 word 방식은 동일한 선택을 방지해야한다는 것을 알 수 있다.

결과_ n=5_masking PPO word_cost







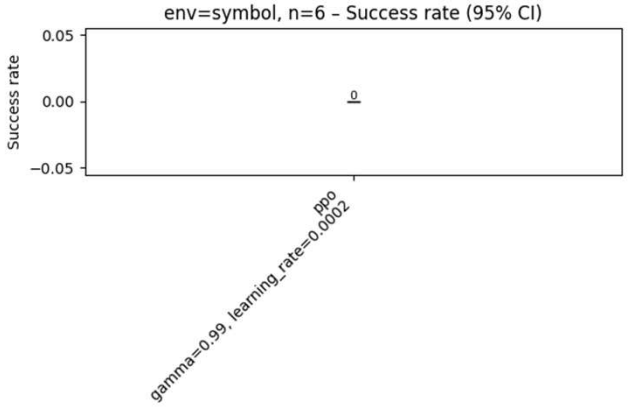
env_type	algo_name	Mean coverage ratio	Mean episode return	Mean episode steps	Mean final length	Success rate
word_cost	maskable_ppo	0.932	429.231	64.539	301.904	0.1

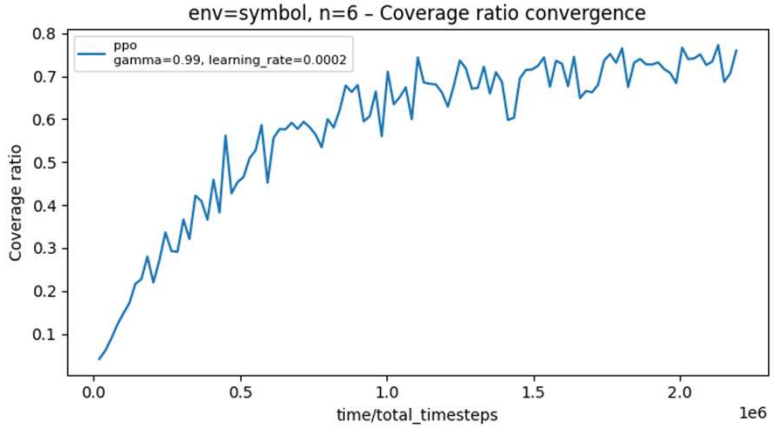
SOGANG UNIVERSITY

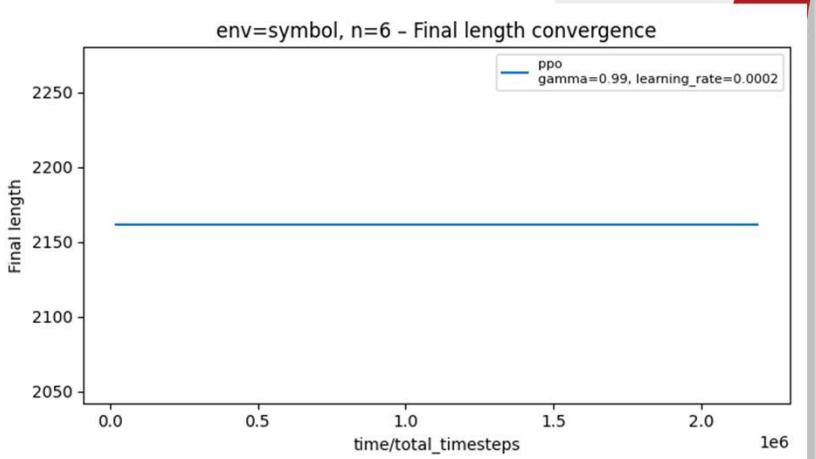
결과_ n=5_masking PPO word_cost

이 경우 symbolic처럼 학습이 되며 커버리지가 줄지않고 성공까지 하는 모습을 보인다.
다만 action space가 커서인지 learning rate가 높아서 인지 학습에 굉장히 불안정한 모습을 보인다. 이는 word방식 추가에 있어서 동일한 word를 막아도 N=5일때부터 초순열이 유일하지 않기 때문인 것일 수도 있으며 실제 선행 연구에서도 단어간의 거리차이가 미미하지만 최종적으로 도달해야하는 거리가 늘어날 수 있는 TSP문제라는 것을 고려하면 이 초순열 문제가 가지고 있는 본질적인 어려움일 수 있다.
다만 Symbolic방식이 유효할 가능성이 있으므로 다음 실험을 진행하였다.

결과_ n=6_리워드 개선 symbolic







env_type	algo_name	Mean coverage ratio	Mean episode return	Mean episode steps	Mean final length	Success rate
symbol	ppo	0.582	1974.93	2161	2161	0

SOGANG UNIVERSITY

결과_ n=6_리워드 개선 symbolic

이 경우 커버리지도 상승하며 긍정적인 추세를 보이나 최대 872길이의 초순열이며 순열이 모두 6!=720 가지이므로 겹치기엔 time_step이 너무 짧았을 수도 있다. 최대길이가 모두 동일한 것은 모든 시도가 최대 길이에 도달하여 에피소드가 종료된것으로 보이므로 2200000 횟수의 time_step을 설정하였으나 n=6에 대해서는 너무 짧은 학습이었다는 판단이다.

결과_ 논의 :

초반에는 greedy 방식이 효과적인 것을 보아 충분히 작은 n은 매우 단순하다.

N이 작을 경우는 word기반으로 초순열을 구성하는 것이 빠르며 효과적이지만 N이 커지고 선택공간이 넓어지며 바로 앞에 이득이 너무 커지는 경향이 있어 N이 큰 경우 그리 효과적이지 못하다는 결론을 낼 수 있다.

Symbolic의 경우는 action이 그리 빠르게 커지지 않으며 탐사가 쉬워 비교적 높은 n에 대해서도 어느정도 효과를 냈지만 n=6의 경우가 되어도 학습이 매우 커지고 무거워진다는 것이 아쉬운 점이었다.

N이 10이나 8만 되어도 학습에 필요한 시간이 매우 늘어날 것을 예측되어 n이 크지 않더라도 실제 최소 초순열 길이에 도달하기까지는 매우 긴 학습이 필요해 보인다. 또한 이런 두번 선택해서는 안돼는 환경에서는 maskedppo와 같은 추가적인 reward 제한 방식이 효과적임을 확인하였다.

한계점

상태 표현: n! 크기의 state를 사용하는 점

학습 예산: RTX4070으로 학습이 매우 오래걸린다.

목표 미달: 최소길이까지 도달하게 하지 못함

충분한 시간을 들여 매우 길게 학습하여 확인하지 못한점이 이 프로젝트의 한계점이다.

향후 연구 방향

RNN, Transformer 등을 사용한 시퀀스 인코딩을 하여 state을 줄이기

작은 n에 대해 학습한 정보를 사용하여 n+1에 적용할 수 있는지 가능성

더 효과적인 Reward 설계

더 긴 timesteps, 더 많은 seed, 분산 학습을 사용

임의의 초순열을 길이를 줄이는 액션으로 최소 초순열로 만들기

결론

작지 않은 n에 대해서는 일반적인 강화학습만으로는 한계가 뚜렷하며

Word방식은 학습 효율은 좋지만 local minimum에 빠질 가능성이 크고

Symbolic은 추가적인 masking은 필요없지만 학습이 비교적느리다.

이런 순열을 만드는 문제에서는 A2C는 그리 좋은 접근이 아니며 PPO가 더 나은 성능을 보여주었다.

Thank you for your attention

(Shortest Super-permutation as the Haruhi Problem: A Reinforcement Learning Approach)

120250131 김범수(실험설계)

120250133 박성원 (PPT작성 및 자료조사)

서강대학교

SOGANG UNIVERSITY