산업인공지능 기말과제

60215256 김주원

CONTENS 01

Rule based

problem 1

Method	Total Tardiness	Job Sequence Example
SPT	39,117	[85, 8, 73, 25, ...]
LPT	158,072	[88, 65, 12, 43, ...]
EDD	**2,905**	[8, 16, 25, 0, ...]
LDD	149,857	[7, 18, 80, 28, ...]
Queue	88,106	['0', '1', '2', ...]
Trie	39,117	['850', '80', '730', ...]

최고 성과: EDD (2,905)
타디니스가 SPT/Trie 대비 약 92.6% 더 낮고, LPT 대비 약 98.2% 더 낮음.
중간 성과: SPT/Trie (39,117)
LPT 대비 타디니스가 약 75% 낮음.
마감 기한을 고려하지 않은 한계로 EDD만큼 최적화되지는 못함.
최저 성과: LPT (158,072), LDD (149,857)
긴 작업이나 늦은 마감 기한을 우선 처리하면서 성능이 크게 저하됨.
Queue (88,106)
SPT/Trie 대비 약 125% 높은 타디니스를 기록하며 중간 수준

problem 2

Method	Total Tardiness	Job Sequence Example
SPT	65,889	[91, 35, 8, 55, ...]
LPT	139,406	[77, 7, 86, 63, ...]
EDD	**24,161**	[16, 56, 96, 36, ...]
LDD	142,563	[15, 58, 98, 18, ...]
Queue	94,592	['0', '1', '2', ...]
Trie	65,889	['910', '350', '80', ...]

최고 성과: EDD (24,161)
타디니스가 SPT/Trie 대비 약 63.3% 더 낮고, LPT 대비 약 82.7% 더 낮음.
중간 성과: SPT/Trie (65,889)
EDD보다는 타디니스가 높지만, 작업 시간을 기반으로 효율적으로 처리.
LPT 대비 약 52.7% 낮음.
최저 성과: LPT (139,406), LDD (142,563)
긴 작업이나 늦은 마감 기한 우선 처리로 비효율적.
SPT 대비 약 110% 높은 타디니스.
Queue (94,592)
SPT 대비 약 43.5% 높은 타디니스로 중간 수준.

Rule based

problem 3

Method	Total Tardiness	Job Sequence Example
SPT	58,745	[99, 25, 53, 1, ...]
LPT	133,242	[88, 96, 72, 50, ...]
EDD	**16,463**	[0, 16, 56, 96, ...]
LDD	131,314	[3, 11, 15, 44, ...]
Queue	85,782	['0', '1', '2', ...]
Trie	58,745	['990', '250', '530', ...]

최고 성과: EDD (16,463)
타디니스가 SPT/Trie 대비 약 71.9% 더 낮고, LPT 대비 약 87.6% 더 낮음.
중간 성과: SPT/Trie (58,745)
EDD보다는 높은 타디니스지만, 작업 시간을 고려하여 균형 잡힌 성능.
LPT 대비 약 55.9% 낮음.
최저 성과: LPT (133,242), LDD (131,314)
긴 작업 중심의 스케줄링으로 비효율적.
EDD 대비 약 8배 이상의 타디니스를 기록.
Queue (85,782)
SPT 대비 약 46% 높은 타디니스로 중간 수준.

결론

EDD:
모든 문제에서 가장 낮은 타디니스를 기록.
특히 마감 기한이 중요한 문제에서 강력한 성과를 보여줌.
SPT/Trie:
작업 시간이 중요한 경우에 효과적.
그러나 마감 기한을 고려하지 않아 EDD보다는 성능이 떨어짐.
Due Date가 작업당 하나인 경우 접두어를 Due Date로 계산하여 SPT 스케줄링값과 똑같이 나와 효율적이지 못한 방법임을 알게 됨.
LPT/LDD:
긴 작업 또는 늦은 마감 기한 중심의 접근은 모든 문제에서 비효율적.
Queue:
중간 수준의 성능을 기록하며, 문제 특성에 최적화되지 못함.

Q-Learning / SARSA

problem 1

학습 알고리즘	총 타디니스	작업 순서
Q-Learning	97897	[56, 66, 83, 32, 60, 82, 53, 71, 49, 68, 15, 52, 4, 57, 74, 65, 47, 90, 68, 15, ...]
SARSA	90764	[1, 61, 43, 8, 52, 97, 51, 84, 29, 99, 51, 84, 29, 99, 51, 84, 29, 99, 51, 84, ...]

SARSA가 Q-Learning보다 약 7.3%
더 나은 총 타디니스를 보였습니다.

problem 2

학습 알고리즘	총 타디니스	작업 순서
Q-Learning	93212	[62, 25, 85, 46, 90, 16, 65, 31, 77, 8, 14, 7, 76, 68, 78, 12, 75, 32, 88, 79, ...]
SARSA	98512	[37, 78, 38, 21, 62, 11, 42, 79, 64, 44, 52, 94, 71, 25, 52, 94, 71, 25, 52, 45, ...]

Q-Learning이 약간 더 나은 결과를 보였지만
두 알고리즘의 성능 차이는 크지 않았습니다.

problem 3

학습 알고리즘	총 타디니스	작업 순서
Q-Learning	93212	[62, 25, 85, 46, 90, 16, 65, 31, 77, 8, 14, 7, 76, 68, 78, 12, 75, 32, 88, 79, ...]
SARSA	98512	[37, 78, 38, 21, 62, 11, 42, 79, 64, 44, 52, 94, 71, 25, 52, 94, 71, 25, 52, 45, ...]

Q-Learning이 약 11% 더 낮은 타디니스를 기록하여
더 효율적인 작업 순서를 생성했습니다.

DQN & GA 융합

GA로 초기 가중치 최적화

```
print("Optimizing initial weights with GA...")
ga = GeneticAlgorithm(population_size=10, input_dim=input_dim, output_dim=output_dim, generations=5)
optimal_agent = ga.evolve(env)
print("GA optimized weights applied.")
```

pseudocode

Input: 환경(env), DQN 네트워크 구조(input_dim, output_dim), GA 설정(population_size, generations, mutation_rate)
Output: GA로 최적화된 DQN 네트워크(optimal_agent)

1. 초기화:
  - population <- 랜덤으로 초기화된 DQN 네트워크 population_size개 생성
  - generations <- 세대 수

2. For generation in 1 to generations:
  a. Evaluate fitness:
    - 각 네트워크 individual ∈ population에 대해:
      - 환경(env)에서 네트워크를 실행하여 total_reward(적합도)를 계산
  b. Select top 2 parents based on fitness scores
  c. Generate new population:
    - Cross over: 부모 두 개체에서 교차하여 자식 생성
    - Mutate: 일정 확률로 자식의 가중치 변이
  d. Update population with new individuals

3. Return: 최적의 네트워크(optimal_agent)

DQN & GA 융합

GA로 하이퍼파라미터 최적화

```
print("Optimizing hyperparameters with GA...")
ga_hyper = GAForHyperparameters(population_size=10, generations=5)
best_hyperparameters = ga_hyper.evolve(env)
print(f"Best Hyperparameters for Problem {idx + 1}: {best_hyperparameters}")
```

pseudocode

Input: 환경(env), 하이퍼파라미터 후보군(learning_rate, gamma, epsilon_decay), GA 설정(population_size, generations)
Output: 최적의 하이퍼파라미터(best_hyperparameters)

1. 초기화:
   - population <- 랜덤 하이퍼파라미터 집합 population_size개 생성
   - generations <- 세대 수

2. For generation in 1 to generations:
   a. Evaluate fitness:
      - 각 하이퍼파라미터 set ∈ population에 대해:
         - 해당 하이퍼파라미터로 DQN 네트워크를 학습
         - 환경(env)에서 네트워크를 실행하여 total_reward(적합도)를 계산
   b. Select top 2 parents based on fitness scores
   c. Generate new population:
      - Cross over: 부모 두 개체에서 교차하여 자식 생성
      - Mutate: 일정 확률로 자식 하이퍼파라미터 변이
   d. Update population with new hyperparameter sets

3. Return: 최적의 하이퍼파라미터(best_hyperparameters)

DQN & GA 융합

GA로 최적화된 가중치와 하이퍼파라미터를 사용한 DQN 학습

pseudocode

Input: GA 최적화된 가중치(optimal_agent), 하이퍼파라미터(best_hyperparameters), DQN 네트워크 구조(input_dim, output_dim)
Output: 학습된 GA+DQN 네트워크(policy_net)

1. 초기화:
  - policy_net <- optimal_agent (GA로 최적화된 초기 가중치)
  - target_net <- policy_net 복사
  - 하이퍼파라미터(best_hyperparameters) 설정

2. For episode in 1 to NUM_EPISODES:
  a. Reset 환경:
    - state <- env.reset()
  b. While not done:
    - ε-greedy로 액션(action) 선택:
      - With probability ε, random action
      - Else, policy_net(state)에서 Q-value가 가장 높은 액션 선택
    - Step 환경:
      - next_state, reward, done <- env.step(action)
    - 경험 리플레이 버퍼에 추가:
      - replay_buffer.push(state, action, reward, next_state, done)
    - 학습:
      - 버퍼에서 미니배치 샘플링
      - Q-value 업데이트:
        Q_current = policy_net(state)
        Q_target = reward + gamma * max(Q_target_net(next_state)) (if not done)
        손실 계산: MSE(Q_current, Q_target)
        손실 역전파 및 최적화

  c. ε 감소 (ε <- max(ε * epsilon_decay, 최소값))
  d. 매 k 에피소드마다 target_net 업데이트:
    - target_net <- policy_net 복사
3. Return: 학습된 policy_net

DQN & GA 융합

Problem별 GA로 선택한 하이퍼파라미터

pseudocode

Input: GA 최적화된 가중치(optimal_agent), 하이퍼파라미터 (best_hyperparameters), DQN 네트워크 구조(input_dim, output_dim)
Output: 학습된 GA+DQN 네트워크(policy_net)

1. 초기화:
   - policy_net <- optimal_agent (GA로 최적화된 초기 가중치)
   - target_net <- policy_net 복사
   - 하이퍼파라미터(best_hyperparameters) 설정

2. For episode in 1 to NUM_EPISODES:
   a. Reset 환경:
      - state <- env.reset()
   b. While not done:
      - ε-greedy로 액션(action) 선택:
        - With probability ε, random action
        - Else, policy_net(state)에서 Q-value가 가장 높은 액션 선택
      - Step 환경:
        - next_state, reward, done <- env.step(action)
      - 경험 리플레이 버퍼에 추가:
        - replay_buffer.push(state, action, reward, next_state, done)
      - 학습:
        - 버퍼에서 미니배치 샘플링
        - Q-value 업데이트:
          Q_current = policy_net(state)
          Q_target = reward + gamma * max(Q_target_net(next_state)) (if not done)
          손실 계산: MSE(Q_current, Q_target)
          손실 역전파 및 최적화

   c. ε 감소 (ε <- max(ε * epsilon_decay, 최소값))
   d. 매 k 에피소드마다 target_net 업데이트:
      - target_net <- policy_net 복사
3. Return: 학습된 policy_net

DQN과 DQN&GA 융합코드 반복

런타임 이슈로 인한 problem1 5회 반복 (episode = 7000)

GA+DQN

에피소드	1회	2회	3회	4회	5회	평균	표준편차
1000	93354	92023	95783	84203	90746	91221.8	4428.69
2000	92649	80314	85308	90722	97125	89223.6	6266.58
3000	92589	91503	94151	98987	94312	94308.4	2623.89
4000	89540	93781	91997	88110	92214	91128.4	2115.49
5000	91209	93244	97571	91433	94933	93678.0	2590.47
6000	94756	93242	89836	90132	93286	92250.4	2015.11
7000	90416	92201	96145	93875	94175	93362.4	1974.58

DQN

에피소드	1회	2회	3회	4회	5회	평균	표준편차
1000	92702	93447	91126	96749	93351	93475.0	2062.15
2000	87065	91459	94249	84772	93247	90158.4	3780.17
3000	91153	94650	94518	95177	87209	92541.4	3221.87
4000	92956	89514	91304	92044	88042	90772.0	1970.22
5000	91715	97411	94155	89845	96161	93857.4	2922.94
6000	89199	93585	93585	89685	93066	91824.0	1994.77
7000	93716	94624	96145	92569	89857	93382.2	2131.86

분석 요약

1000번째 에피소드

GA+DQN 평균: 91221.8, 표준편차: 4428.69

DQN-only 평균: 93475.0, 표준편차: 2062.15

GA+DQN이 DQN-only보다 표준편차가 크지만 평균 값은 더 낮음.

2000번째 에피소드

GA+DQN 평균: 89223.6, 표준편차: 6266.58

DQN-only 평균: 90158.4, 표준편차: 3780.17

GA+DQN은 평균 Total Tardiness가 더 낮음.

3000~6000번째 에피소드

에피소드 수가 증가하면서 GA+DQN의 안정성이 개선되고 있음(표준편차 감소).

DQN-only는 3000~5000 에피소드에서 평균 Tardiness 값이 오히려 증가하는 경향을 보임.

7000번째 에피소드

GA+DQN 평균: 93362.4, 표준편차: 1974.58

DQN-only 평균: 93382.2, 표준편차: 2131.86

GA+DQN이 표준편차에서 우위를 보이며 결과 안정성이 더 높음.

결론

GA+DQN은 학습 초기에는 DQN-only 대비 편차가 컸으나, 점차 안정성을 확보하며 평균 Tardiness 값도 더 낮은 결과를 보임.

DQN-only는 학습 초반 성능이 우세했으나, 7000번째 에피소드에서는 GA+DQN과 유사한 평균 성능을 보였으나 안정성은 다소 낮음.

모든 문제 결과 비교

problem 1

	Method	Total Tardiness
1	SPT	39117.0
2	LPT	158072.0
3	EDD	2905.0
4	LDD	149857.0
5	Queue	88106.0
6	Trie	39117.0

problem 2

	Method	Total Tardiness
1	SPT	65889.0
2	LPT	139406.0
3	EDD	24161.0
4	LDD	142563.0
5	Queue	94592.0
6	Trie	65889.0

problem 3

	Method	Total Tardiness
1	SPT	58745.0
2	LPT	133242.0
3	EDD	16463.0
4	LDD	131314.0
5	Queue	85782.0
6	Trie	58745.0

종합 비교 및 분석

SPT와 EDD가 대부분의 문제에서 가장 낮은 타디니스 값을 보여줌.
EDD는 문제 3에서 가장 좋은 결과(타디니스 16,463)를 보였음.

GA+DQN은 규칙 기반 알고리즘(SPT, EDD)을 제외하고 Q-Learning, SARSA, DQN보다 성능이 우수한 경우가 많았음.
특히 문제 3에서는 DQN보다 7,000 낮은 타디니스를 기록함.

SARSA는 세 문제에서 Q-Learning보다 성능이 약간 더 나았음.
하지만, GA 기반의 알고리즘에는 미치지 못함.

Queue 기반 스케줄링은 전반적으로 높은 타디니스를 기록했으며 효율성이 낮았음.

Trie 기반 알고리즘은 SPT와 비슷한 성능을 보이며, 문제 1과 문제 3에서 동일한 최적의 결과를 기록함.

GA+DQN은 문제 2와 문제 3에서 DQN 대비 개선된 성능을 보여,
GA로 초기 가중치 최적화 및 하이퍼파라미터 조정의 효과를 입증함.

문제 별 선택 알고리즘

문제 1

선택 알고리즘: EDD
결과

EDD의 총 타디니스: 2,905
가장 낮은 타디니스를 기록했으며, 다른 알고리즘에 비해 압도적으로 효율적.
이유

특성: 문제 1에서는 작업의 Due Date 중심으로 작업을 정렬했을 때 최적의 결과를 얻을 수 있는 구조를 가짐.
EDD 알고리즘의 원리:
마감 기한이 가까운 작업을 우선적으로 처리하여 전체 지연 시간을 최소화.
다른 알고리즘과의 비교:
SPT (39,117): 작업 처리 시간 중심으로 정렬하였으나, 마감 기한 고려가 부족하여 효율성이 떨어짐.
GA+DQN/DQN (94,175): 강화 학습 기반 알고리즘이지만, 초기 학습 및 하이퍼파라미터 조정에서 최적화 부족.
SARSA (90,764), Q-Learning (97,897)도 EDD의 단순한 규칙 기반 접근을 넘어서지 못함.

문제 2

선택 알고리즘: SPT결과

SPT의 총 타디니스: 65,889
다른 알고리즘에 비해 가장 낮은 타디니스를 기록.
이유

특성: 문제 2는 작업의 처리 시간이 짧은 작업을 우선적으로 수행했을 때, 전체적인 지연을 최소화할 수 있는 구조.
SPT 알고리즘의 원리:
처리 시간이 짧은 작업을 우선적으로 배치하여 시스템 전체의 평균 처리 시간을 단축.
다른 알고리즘과의 비교:
EDD (24,161)도 좋은 결과를 보였지만, 문제 2의 작업 특성상 처리 시간 기반 최적화(SPT)가 효과적.
GA+DQN (94,175)는 DQN보다 성능이 낮음.
LDD (142,563), LPT (139,406)는 작업 마감 기한 및 긴 처리 시간 중심으로 정렬하여 최적화에 실패.
Trie 알고리즘은 SPT와 같은 성능을 보여, SPT의 규칙 기반 접근이 문제 2에 적합함을 입증.

CONTENS 06

문제 별 선택 알고리즘

문제 3

선택 알고리즘: GA+DQN
결과

GA+DQN의 총 타디니스: 87,305
DQN보다 낮은 타디니스를 기록하며 규칙 기반 알고리즘(SPT, EDD)을 제외한 학습 알고리즘 중 최고의 성능.
이유

특성: 문제 3은 작업 수와 제약 조건이 복잡해 단순한 규칙 기반 알고리즘으로는 최적의 스케줄링을 찾기 어려운 구조.
GA+DQN 알고리즘의 원리:
GA가 DQN의 초기 가중치를 최적화하여 학습 과정을 빠르고 효율적으로 수행.
DQN 경험을 바탕으로 최적화된 작업 순서를 찾아냄.
다른 알고리즘과의 비교:
SPT (58,745) 및 EDD (16,463)는 규칙 기반 접근으로 우수한 성능을 보였으나, 학습 알고리즘이 복잡한 문제 해결에 더 적합.
SARSA (98,512)와 **Q-Learning (93,212)보다 성능이 개선.
LPT (133,242), LDD (131,314)는 문제 3의 작업 특성에 부합하지 않아 높은 타디니스 기록.

Insight

☑ GA의 초기 가중치 최적화 효과

GA를 통해 초기 가중치를 최적화한 GA+DQN은 초기 학습 성능이 더 높았습니다.

이는 GA가 DQN 학습의 초기 조건을 개선하여 더 빠른 수렴을 가능하게 한 것으로 보입니다.

☑ GA의 하이퍼파라미터 탐색 효과

GA를 활용한 하이퍼파라미터 탐색(learning_rate, gamma, epsilon_decay)은 DQN Only의 고정된 하이퍼파라미터보다 학습 효율을 높이는 데 기여한 것으로 보입니다.

특히, gamma와 epsilon_decay의 적절한 설정은 탐험Exploration과 Exploitation의 균형을 맞추어 타디니스 감소에 기여한 것으로 보입니다.

☑ DQN Only의 불안정성

에피소드 7000에서 DQN Only의 총 타디니스는 오히려 증가하는 경우가 있었습니다. 이는 학습 중 발생하는 Q-Value Divergence나 불안정한 학습의 징후일 수 있습니다.

반면, GA+DQN은 꾸준한 성능을 보이며 최종 타디니스가 더 낮은 값에 도달합니다.