1 Nested Rollout Policy Adaptation

hruighr eufeuf

Algorithm 1: The NRPA playout algorithm

```
playout(state, policy) sequence ← []
while true do
    if state is terminal then
        return (score (state), sequence)
    end
    z ← 0.0
    for m in possible moves for state do
        z ← z + exp(policy [code(m)])
    end
    move ← choose a move with probability exp(policy [code(move)]) / z
    state ← play(state, move)
    sequence ← sequence + move
end
```

Algorithm 2: The NRPA adapt algorithm

```
adapt(policy, sequence)
polp ← policy
state ← root
for move in sequence do
    polp [code [move]] ← polp [code(move)] + α
    z ← 0.0
    for m in possible moves for state do
        z ← exp(policy [code(m)])
    end
    for m in possible moves for state do
        polp [code(m)] ← polp [code(m)] - α * exp(policy [code(m)]) / z
    end
    state ← play(state, move)
end
policy ← polp
```

Algorithm 3: The NRPA algorithm

```
NRPA(level, policy)
if level == 0 then
    return playout(root, policy)
end
bestScore ← −∞
for N iterations do
    (result, new) ← NRPA(level- 1, policy)
    if result ≥ bestScore then
        bestScore ← result
        seq ← new
    end
    policy ← adapt(policy, seq)
end
return bestScore, seq
```

2 Adapt the NRPA to stochastic problems

2.1 the Nested Rollout Policy Adaptation (NRPA) in a stochastic context

As seen in the section **??**, the NRPA learns a policy while finding the best sequence of move possible. Using a policy, it generates a sequence and set its score to equal the score obtained when reaching a final state in the playout. It then stores the sequence with the highest score and adapt its policy so it tends to generate this sequence more often. In a stochastic context, this strategy is problematic on different levels:

fill the ref ence to nr definition section

1. a sequence of moves can obtain a high score by fluke.

2. a given move may be illegal in certain state.

3. a state may never be encountered again due to its low probability.

The first point may be easy to fix as a given sequence can be used in a new playout. It is then possible to compute several playout for a unique sequence allowing us to compute an average score. A mean score is preferable when dealing with stochastic problem as it tends to capture the best possible outcome. Using average score has already been used with success in algorithm such as Upper Confidence bound applied to Trees (UCT)[1].

The second point is more delicate. Given a sequence $S = \{(s1, a1), (s2, a2), \dots\}$which starts by using action, or move, $a1$ on state $s1$ to obtain state $s2$. In a stochastic context, the probability to have $s2$ when applying $a1$ to $s1$ may not equal 1; therefore, any sequence of state and actions may not be reproducible. Even if we consider only the actions such as $S = \{a1, a2, a3, \dots\}$ then we may obtain states in which some actions are illegal. If such a case occur then, not only the sequence is not reproducible but it is also not usable. The idea of sequence must be changed entirely if we want to use it in a stochastic context.

The third point concerns the concept of policy. A policy gives a probability distribution over actions for a given state. In a stochastic problem, the number of states can be quite large and a hundred playout might never see the same states. Therefore, learning probabilities over actions for each state may never give any results as each state could be seen one and one time only. This type of policy is not an option as it is unlikely to give any knowledge to the algorithm.

These three points are, for us, the main concerns about NRPA in a stochastic concern. We tried to address them by designing the Stochastic Nested Rollout Policy Adaptation (SNRPA).

2.2 The Stochastic Nested Rollout Policy Adaptation

Algorithm 4: The SNRPA playout algorithm

```
playout(state, policy)
sequence ← generateSequence(policy)
sumScore ← 0
for i ← 0 to nbPlayout do
    copyState ← state
    while copyState is not terminal do
        code ← choose the first legal code in sequence
        move ← decode(code)
        copyState ← play(copyState, move)
    end
    sumScore ← sumScore + score(copyState)
end
score ← sumScore/ nbPlayout
return (score, sequence)
```

Algorithm 5: The SNRPA generate sequence

```
generateSequence(policy)
sequence ← []
for i ← 0 to numberCode do
    z ← 0.0
    for code in possible codes do
        if code is not in sequence then
            z ← z + exp(policy [code])
        end
    end
    code ← choose a code not in sequence with probability exp(policy [code]) / z
    sequence ← sequence + code
end
return sequence
```

fioueyhuifehf

Algorithm 6: The SNRPA adapt algorithm

```
adapt(policy, sequence)
polp ← policy
for codeIndex ← 0 to sequence.size() - 2 do
    polp [sequence [codeIndex]] ← polp [sequence [codeIndex]] + α
    z ← 0.0
    for codeIndexBis ← codeIndex to sequence.size() do
        z ← exp(policy [sequence [codeIndexBis]])
    end
    for codeIndexBis ← codeIndex to sequence.size() do
        polp [sequence [codeIndexBis]] ← polp [sequence [codeIndexBis]] −α * exp(policy [sequence [codeIndexBis]]) / z
    end
end
policy ← polp
```

Algorithm 7: The SNRPA algorithm

```
SNRPA(level, policy)
if level == 0 then
    return playout(root, policy)
end
bestScore ← −∞
for N iterations do
    (result, new) ← SNRPA(level- 1, policy)
    if result ≥ bestScore then
        bestScore ← result
        seq ← new
    end
    policy ← adapt(policy, seq)
end
return bestScore, seq
```

References

[1] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *Proceedings of the 17th European Conference on Machine Learning*. ECML'06. Berlin, Germany: Springer-Verlag, 2006, pp. 282–293. ISBN: 354045375X. DOI: 10.1007/11871842_29. URL: https://doi.org/10.1007/11871842_29.