1 Nested Monte Carlo pour la sélection de features

Les méthodes de Monte-Carlo imbriquées sont des variantes des méthodes de Monte-Carlo qui utilisent des appels imbriqués pour biaiser les playouts. Nested Monte-Carlo Search [Caz09] est un algorithme de Monte-Carlo imbriquée qui a obtenu de bons résultats pour plusieurs problèmes d'optimisation.

Une variante des méthodes de Monte-Carlo imbriquées à été développée par Chris Rozin et porte le nom de Nested Rollout Policy Adaptation (NRPA) [Ros11]. Cette variation permet d'apprendre en ligne une politique de playout. L'algorithme repose sur deux composantes majeures : une structure imbriquée introduite dans [Caz09] et une politique adaptative de playout. La politique adaptative de rollout est alors une politique qui est paramétrisée par un poids sur chaque action. Les actions sont ainsi choisies pendant les playouts en utilisant cette politique. L'échantillonage de Gibbs est utilisé pour la mise à jour de la politique. Chaque action est codée par un index sur la politique qui retourne son poids.

NRPA possède cependant un problème d'évaluation. En effet, la meilleure séquence renvoyée par l'algorithme peut être biaisé vers la meilleure séquence du niveau 1 dans le cas où une bonne politique n'obtiendrait pas de bons résultats au niveau 0. Or, cette politique peut être meilleure ce qui souligne une restriction dans l'exploration au niveau 0. Pour résoudre ce problème une autre variante de l'algorithme a été introduite qui va générer P séquences au lieu d'une : Stabilized NRPA [CST20]. L'idée derrière l'algorithme est que la génération de ces plusieurs séquences puisse diversifier l'exploration au niveau 1.
Les auteurs ont ainsi noté que cela améliorait la stabilité de la convergence de NRPA.

Afin de réaliser la sélection de nos features, on passe par une application de l'algorithme NRPA stabilisée qui a obtenu de meilleurs résultats de manière générale pour notre problème. On considérera comme un *état*, un triplet avec : une liste d'indices à ajouter, une liste d'indice ajoutés et une liste d'indice non ajoutés.

Le pseudo code pour la version stabilisée de NRPA est le suivant :

Algorithm 1 The Stabilized NRPA algorithm.

```
1: StabilizedNRPA (level, policy)
2: if level == 0 then
3:     return playout(root, policy)
4: else if level == 1 then
5:     bestScore ← −∞
6:     for 1,..,P do
7:         (result, new) ← StabilizedNRPA(level-1, policy)
8:         if result ≥ bestScore then
9:             bestScore ← result
10:            seq ← new
11:        end if
12:    end for
13:    return (bestScore, seq)
14: else
15:    bestScore ← ∞
16:    for 1,..,N do
17:        (result, new) ← StabilizedNRPA(level-1, policy)
18:        if result ≥ bestScore then
19:            bestScore ← result
20:            seq ← new
21:        end if
22:        policy ← Adapt(policy, seq)
23:    end for
24:    return (bestScore, seq)
25: end if
```

- Dans notre cas, la valeur **bestScore** correspondra ainsi à une accuracy et **seq** sera une liste d'indice de features sélectionnées.

- Dans notre cas, l'état **root** sera un triplet avec une liste d'indices à ajouter qui correspond à tous les indices de features de l'espace utilisé, une liste d'indices ajoutés qui est vide et une liste d'indices non ajoutés qui est également vide.

Le pseudo-code de la fonction playout qui joue une partie aléatoire en utilisant la politique associée est le suivant :

Algorithm 2 L'algorithme de playout.

```
1: Playout (state, policy)
2: sequence ← []
3: while True do
4:     if state is terminal then
5:         return (score(state), sequence)
6:     end if
7:     z ← 0.0
8:     for m in possible moves for state do
9:         z ← z + exp(policy[code(m)])
10:    end for
11:    choose a move with probability exp(policy[code(m)]) / z
12:    state ← play(state, move)
13:    sequence ← sequence + move
14: end while
```

- Dans notre cas, la fonction **score** sera calculée en effectuant une validation croisée à 3 blocs avec les features sélectionnées et en renvoyant la métrique définie par l'utilisateur pour le pipeline.

- Dans notre cas, la fonction **code** renverra un indice pour l'action m sur un vecteur de réel de taille $k = 2 * |F_{normalized}|$ correspondant aux poids de la politique.

Enfin, le pseudo code de la fonction *adapt* qui adapte la politique est :

Algorithm 3 L'algorithme Adapt.

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