Sujet Grand Oral NSI : En quoi l

▶ SommaireSauter à une section
Bonjour. Je vais vous parler d'un algorithme qui m'a fait découvrir l'élégance de l'optimisation **informatique** : l'élagage alpha-bêta, appliqué au jeu d'échecs.

Le **sujet** est précis : en quoi l'élagage alpha-bêta illustre-t-il l'optimisation d'une recherche arborescente ? Cette **question** m'intéresse parce qu'elle se trouve au cœur du **programme** de spécialité **NSI** en **terminale** : elle mobilise la notion d'**arbre**, le concept de **complexité** algorithmique, la **récursivité**, et l'idée plus large que toute structure de **données** peut être parcourue de manière naïve ou intelligente.

Aux échecs, le **programme** doit choisir un coup parmi environ trente-cinq possibles à chaque tour. Si on veut anticiper sur plusieurs coups, le nombre d'options explose : c'est ce qu'on appelle l'**explosion combinatoire**. L'algorithme minimax, qui explore exhaustivement toutes les branches d'un **arbre** de jeu, atteint vite ses **limites**. L'élagage alpha-bêta vient résoudre ce problème en coupant intelligemment des branches inutiles, sans changer le résultat final.

Je vais donc procéder en trois étapes. D'abord, je présenterai l'algorithme minimax et son problème de **complexité**. Ensuite, je détaillerai l'idée de l'élagage alpha-bêta et son code Python. Enfin, je quantifierai le gain réel et je discuterai les **limites** de cette optimisation.

Minimax et le problème de la complexité exponentielle

L'algorithme minimax, formalisé par Claude Shannon en 1950, modélise le jeu d'échecs comme un **arbre** de décision. La racine est la position actuelle. Chaque branche représente un coup possible. Le **programme** explore cet **arbre** en profondeur, et à chaque nœud, il calcule la valeur de la position selon une logique simple : je joue pour maximiser mon avantage, mon adversaire joue pour me faire perdre, donc à chaque niveau on alterne entre maximisation et minimisation.

Voici le code Python du minimax classique.

Python Copier

```python
def minimax(position, profondeur, maximise):
    if profondeur == 0 or position.est_terminee():
        return evaluer(position)
    if maximise:
        meilleur = -float('inf')
        for coup in position.coups_possibles():
            position.joue(coup)
            score = minimax(position, profondeur - 1, False)
            position.annule(coup)
            meilleur = max(meilleur, score)
        return meilleur
    else:
        meilleur = +float('inf')
        for coup in position.coups_possibles():
            position.joue(coup)
            score = minimax(position, profondeur - 1, True)
            position.annule(coup)
            meilleur = min(meilleur, score)
        return meilleur
```

Le problème de cet algorithme est sa **complexité**. À chaque nœud, on explore en moyenne trente-cinq fils. Sur quatre niveaux de profondeur, on a trente-cinq puissance quatre, soit environ un million cinq cent mille positions. Sur six niveaux, on dépasse le milliard et demi. La **complexité** s'écrit O(b puissance d), où b est le facteur de branchement et d la profondeur. Cette croissance exponentielle est, en termes de **programme** de spécialité, une fonction qui explose : impossible à gérer sans optimisation.

Pour la perspective historique : il y a environ dix puissance cent vingt parties d'échecs possibles. C'est le nombre de Shannon. Aucun ordinateur ne pourra jamais explorer l'**arbre** entier, même en utilisant toute la puissance de calcul de l'humanité pendant l'âge de l'univers. Le **programme** doit donc se contenter d'une exploration partielle, à profondeur fixée, et c'est ici que l'optimisation devient cruciale : chaque profondeur supplémentaire gagnée se traduit par une amélioration mesurable de la force du **programme**.

L'idée de l'élagage alpha-bêta

L'élagage alpha-bêta repose sur une intuition simple. Si je sais déjà qu'une branche ne peut pas améliorer ma meilleure option connue, il est inutile de l'explorer. On gagne du temps en abandonnant des sous-arbres entiers dès qu'on peut prouver qu'ils ne changeront pas le résultat.

Concrètement, on maintient deux bornes pendant l'exploration. **Alpha** est le meilleur score que les Blancs peuvent garantir sur la branche actuelle. **Beta** est le meilleur score que les Noirs peuvent garantir. Si à un nœud on découvre que beta est inférieur ou égal à alpha, cela signifie que la branche n'intéressera ni les Blancs ni les Noirs : on peut l'abandonner immédiatement. Cette coupure est ce qu'on appelle un **élagage**.

Le code Python correspondant ajoute deux paramètres au minimax classique.

Python Copier

```python
def alpha_beta(position, profondeur, alpha, beta, maximise):
    if profondeur == 0 or position.est_terminee():
        return evaluer(position)
    if maximise:
        meilleur = -float('inf')
        for coup in position.coups_possibles():
            position.joue(coup)
            score = alpha_beta(position, profondeur - 1, alpha, beta, False)
            position.annule(coup)
            meilleur = max(meilleur, score)
            alpha = max(alpha, meilleur)
            if beta <= alpha:
                break
        return meilleur
    else:
        meilleur = +float('inf')
        for coup in position.coups_possibles():
            position.joue(coup)
            score = alpha_beta(position, profondeur - 1, alpha, beta, True)
            position.annule(coup)
            meilleur = min(meilleur, score)
            beta = min(beta, meilleur)
            if beta <= alpha:
                break
        return meilleur
```

Les deux instructions clés sont `if beta <= alpha: break`. Cette condition, qu'on appelle la **coupure**, est le cœur de l'optimisation. Quand elle se déclenche, la boucle s'arrête immédiatement : on n'explore pas les coups restants, on remonte le résultat actuel au nœud parent. Toute la subtilité tient dans le fait que cette coupure ne change pas la valeur retournée par l'algorithme : on prouve mathématiquement que les branches coupées ne contiennent pas de meilleure réponse.

Pour le **jury**, je peux dessiner un petit **arbre** à trois niveaux et montrer en direct quelle branche est coupée et pourquoi. C'est l'exercice qui marque les esprits : visualiser une **complexité** qui se transforme en quelques traits de crayon.

Quantifier le gain et discuter les limites

L'élagage alpha-bêta réduit la **complexité** de O(b puissance d) à O(b puissance d sur deux), dans le meilleur cas. Cette formule est due à Donald Knuth en 1975. Concrètement, pour une profondeur de six, on passe de trente-cinq puissance six, soit environ un milliard huit cents millions, à trente-cinq puissance trois, soit environ quarante-deux mille. Le gain est d'un facteur quarante mille. C'est ce qui permet à des **programmes** comme Stockfish d'explorer des profondeurs de vingt à trente coups en quelques secondes, là où le minimax pur s'arrêterait à cinq ou six.

Cependant, ce gain n'est atteint que dans le meilleur cas, c'est-à-dire quand les coups sont explorés dans le meilleur ordre possible. Dans le pire cas, c'est-à-dire quand on explore d'abord les coups les moins prometteurs, l'élagage alpha-bêta dégénère en minimax pur : aucune coupure n'a lieu. C'est pour cela que les **programmes** modernes investissent énormément dans des heuristiques de tri des coups : table de transposition, killer heuristic, history heuristic. Toutes ces techniques cherchent à présenter les coups dans le bon ordre pour maximiser les coupures.

Au-delà de la performance brute, l'élagage alpha-bêta illustre un principe fondamental de l'**informatique** : optimiser ne consiste pas à chercher toutes les solutions, mais à éliminer intelligemment les mauvaises. C'est exactement la philosophie qu'on retrouve dans d'autres algorithmes d'optimisation au **programme** de spécialité, comme la programmation dynamique ou la mémoïsation.

Les **limites** de cette approche restent réelles. Premièrement, même avec alpha-bêta, on ne peut pas atteindre des profondeurs supérieures à trente coups, ce qui est très loin de la fin de partie. Deuxièmement, la qualité de l'évaluation des positions atteintes à la profondeur limite dépend toujours d'une fonction écrite par des humains, donc imparfaite. Troisièmement, et c'est la rupture la plus profonde, l'élagage alpha-bêta reste un raisonnement déductif : on explore un **arbre** prédéfini. Or AlphaZero, en deux mille dix-sept, a montré qu'on pouvait obtenir de meilleurs résultats en abandonnant l'**arbre** entier au profit d'un réseau de neurones qui apprend à reconnaître les positions sans les énumérer.

Conclusion

Pour répondre à ma **question** initiale, l'élagage alpha-bêta illustre l'optimisation d'une recherche arborescente de la manière la plus claire possible : il prend un problème de **complexité** exponentielle et le rend tractable en exploitant la structure du problème, sans changer le résultat. Il transforme l'impossible en faisable, en quelques lignes de code.

Cette idée d'éliminer intelligemment plutôt que d'énumérer exhaustivement est, à mon sens, l'un des plus beaux **enjeux** de l'**informatique** moderne. Elle apparaît dans la compression de données, dans les **systèmes** de recherche sur le web, dans les solveurs de contraintes. Les échecs ont été le terrain d'expérimentation historique de cette idée, et alpha-bêta en reste la formulation la plus élégante.

L'ouverture naturelle, c'est de se demander si l'approche par apprentissage automatique d'AlphaZero rend obsolète l'élagage alpha-bêta. Ma réponse est non. Depuis deux mille vingt, Stockfish intègre lui-même un réseau de neurones pour l'évaluation, mais conserve alpha-bêta pour la recherche. Les deux paradigmes se complètent, et c'est probablement dans cette hybridation que se trouve l'avenir des moteurs.

Je vous remercie et je suis prêt à répondre à vos **questions**.