Rapport IAT

Résolution du MasterMind par Meta-heuristiques et recherche locale

11 Février 2021

Auteurs : Sandrine Roux - Léonard Grynfogel

4TC2

Table des matières

1 Abstract

Le Mastermind est un jeu de société courant, jouable par deux joueurs et dont le but est de craquer un code.
Selon les créateurs du jeu, une réponse en cinq essais est meilleure que la moyenne, et une réponse en deux essais ou moins doit être attribué à de la chance.
Contrairement à certains jeux qui peuvent être résolus informatiquement, tel que le jeu d'Hex ou les dames, le mastermind ne peut être résolu dans le nombre minimum de coup (3). Nous allons cependant dans ce document essayer de s'en rapprocher un maximum en comparant deux approches : l'approche génétique et l'approche linéaire.

2 Introduction

Bien que le mastermind paraissent déconnecté de notre formation initialement, il a en réalité été inventé par Mordecai Meirowitz, un expert en télécommunications et a contribué à la fondation de la cyber-sécurité, c'est ainsi un peu plus qu'un jeu qui a même servi pendant de nombreuses années comme entrainement par l'armée australienne. Le concept est tout simple, chacun des deux joueurs incarnera un rôle, le codificateur ou le décodeur. Le but pour le décodeur est de deviner par déductions successives la couleur et la position d'un nombre de pions préalablement choisis et caché par l'encodeur. Dans le jeu initial, il y a 4 positions de pions et 6 couleurs possibles. A chaque tentative du décodeur, le codificateur lui indique le nombre de pions ayant la bonne couleur et étant au bon endroit par un nombre de pin noir et le nombre de pions ayant la bonne couleur mais étant mal placé par un certain nombre

FIGURE 1: Version original du mastermind

de pin blanc. Si le code est craqué en moins de 12 essais, le décodeur est gagnant et les rôles s'inversent, sinon c'est l'encodeur qui a gagné. Dans ce document nous présenterons un algorithme de résolution basé sur le sujet n°2 d'IAT.

3 Création du modèle du Mastermind

Nous représenterons dans ce document chaque code du mastermind par une suite de lettres correspondant chacune à une couleur.
Ainsi,le code **Bleu - Orange - Rouge - Violet** deviendra $[B, O, R, V]$ Chaque essai se verra par la suite attribué un score de MP (Mal placé) et BP (Bien Placé) ainsi qu'un score général étant la somme des MP et BP, les BP étant pondérés par deux, puisque plus importants.
Ainsi :

$$S(e) = 2 * BP + MP$$

avec e l'essai en question.

Un code pouvant être le bon sera donc un code ayant les exacts mêmes scores lorsqu'il est comparé avec chacun des essais que les scores obtenus durant la partie.

Nous choisirons donc le code avec l'écart minimum de score lorsqu'on le compare avec l'ensemble des essais précédents.
La fonction que l'on cherchera à minimiser, appelé par la suite fitness sera la suivante :

$$f(c, E) = \sum_{i=1}^{n}(|MP(E(i) - MP(c)| + 2 * |BP(E(i) - BP(c)|)$$

avec c le code dont l'on calcul le fitness,
E l'ensemble des précédents essais

4 Problématique et approche

L'objectif est donc de trouver le bon code. Plus simplement, il s'agit en réalité de trouver le code parmis la totalité des codes possibles qui a la plus grande probabilité d'être le bon.
Si nous choisissions en effet de manière aléatoire nous mettrions jusqu'à N^C possibilités avec N le nombre de positions et C le nombre de couleurs. Ce n'est donc pas une solutions viable.
Viens ensuite la possibilité d'éliminer à chaque tour tous les codes non éligibles et d'en choisir un aléatoire. Même si cette solution parait à la fois simple et efficace elle nécessite surtout un grand temps de calcul qui reste envisageable lors d'un jeu avec 4 positions et 6 couleurs mais le serai sans doute moins pour des valeurs plus élevés, ayant une complexité proportionnel au nombre total de possibilités.
Nous traiterons cependant cette possibilité à la fin de ce rapport. La solution qui elle aussi envisageable est une solution lié à l'intelligence artificielle et plus précisement aux méta-heuristique. Il s'agit des algorithmes génétiques. Le principe est de copier la nature en créant une population de solutions possibles que l'on fera évoluer au fur et à mesure de l'avancée de notre problématique. C'est cette approche que nous allons traiter ici.

5 Algorithme génétique

5.1 Principe

Notre approche ici va être de créer une population initiale de code aléatoire.
A la fin de chaque tour nous allons faire se développer cette population en fonction de la réponse au code proposé precedemment.
Nous allons choisir les codes ayant le plus de chances d'être le bon code, puis nous allons faire se développer la population à partir de ses codes.
Lorsque la population sera développé nous allons sélectionner les codes pouvant être les bons puis redévelopper jusqu'à avoir un certain nombre de code éligible qui constitueront notre nouvelle population dans laquelle nous choisirons le futur code à proposer de manière aléatoire.

5.2 Développement de la population

5.2.1 Principe

Afin de générer une nouvelle population à partir de l'ancienne, nous commmençons par calculer le fitness de chaque membre de la population. Nous normalisons ensuite chaque valeur entre 0 et 1. Nous sélectionnons ensuite aléatoirement des couples de codes, ceux avec une valeur de fitness basse ayant plus de chance d'être sélectionnée. Chaque couple de code peut à chances égales générer un nouveau code des deux manières suivantes :

1. Croisement en un point : un index est tiré aléatoirement, inférieur au nombre de positions, toutes les couleurs des positions inférieurs à cet index seront héritées d'un parent et toutes celles des positions supérieurs de l'autre parent.
2. Croisement en deux points : deux index sont tirés, à l'intérieur de ces indexs les couleurs sont celles d'un parent, à l'extérieur ce sont celles de l'autre parent.

Nous effectuons ensuite 3 mutations de la population, à chances égales de 2% :

1. Mutation : Une couleur est changée à un index aléatoire du code.

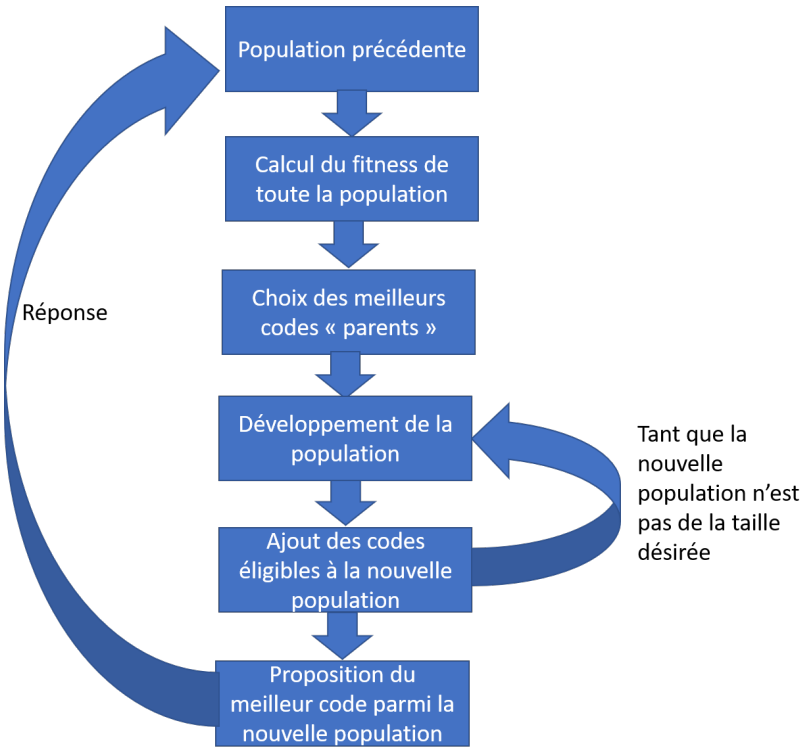

FIGURE 2: Algorithme Génétique pour le mastermind

2. Inversion : Un index est choisi aléatoirement, le code est inversé entre les deux côtés de l'index
3. permutation : Deux couleurs d'un code sont inversés.

Ces trois mutations permettent de générer des codes et de diversifier la population.
La mutation permet en effet de trouver une solution si nous avons 3 couleurs sur 4 ou que la population n'est plus assez diversifiée.
L'inversion permet de réduire potentiellement le nombre de MP en maximisant celui de BP
La permutation permet aussi de réduire le nombre de MP mais de manière plus discrète.

5.2.2 Bilan

Après nos tests, et par une volonté d'optimiser le temps de traitement nous avons décidés de choisir une population de 150 individus, l'augmentation du nombre d'individus n'améliorait pas nos résultats (-0,02 essais en moyenne pour une population de 500) alors que la réduction les influencait (+0,3 essais pour une population de 75 individus).
Nous choisissons aussi de fixer un nombre d'individus éligible limite à 60. Lorsque ce nombre sera obtenu, nous completeront simplement avec des codes de la population actuelle.
Afin de limiter le temps de calcul, nous limitons le nombre de générations de population afin de remplir le quota d'individus éligible à 50. Au delà de cela, nous prenons

simplement les individus déjà présent, cela permet de limiter les calculs inutiles surtout lors des derniers essais où il ne peut pas y avoir 60 individus éligibles.

6 Résultats

Après les différentes optimisations de notre algorithme, nous avons obtenus les résultats suivants pour un jeu classique, c'est à dire avec 6 couleurs et 4 emplacements possibles :

Temps				Essai			
Moyenne	Ecart Type	Max	Min	Moyenne	Ecart Type	Max	Min
2,52	0,67	4,82	0,570	4,54	0,83	7	2

Nous pouvons donc voir que nous avons des résultats très satisfaisant puisque nous sommes en dessous de la barre des 5 essais, cependant, nous sommes encore loin du seuil des 3 essais.
Au niveau du temps, nous avons un temps assez court de résolution et surtout avec un écart type très faible.
Cela permet de valider le fait que notre modèle nous permet aussi de résoudre des masterminds avec des ensembles plus grands. En effet, la taille de la population étant toujours la même, nous n'avons pas de grands écarts de temps ni d'essais ce qui fait que notre modèle est assez fiable et les expériences répétables.
Cela garantit ainsi qu'avec des ensembles plus larges nous n'aurons pas des durées de traitements qui divergeront et bloqueront le traitement. Afin d'améliorer ces résultats, il serait intéressant de se pencher sur le choix du premier code. Dans cet étude, ce choix est aléatoire mais il pourrait être intéressant de constater l'influence de ce choix.

Nous allons maintenant confronter cet approche avec l'approche linéaire dont nous traitions dans la problématique.

7 Approche linéaire

7.1 Principe

L'approche ici, est de générer en début de partie l'entièreté des codes possibles. Nous allons ensuite, en fonction de chaque retour, supprimer les codes qui ne peuvent pas être la solution, donc avec une fitness différente de 0. C'est donc une solution sans mémoire, nous agissons à chaque tour seulement en fonction du dernier résultat. La problématique va ensuite être dans le choix du code parmi ceux restant. En effet, le choix d'un code aléatoire n'est pas forcément une solution optimale dans des ensembles aussi grands. Une des solutions de choix possible est de sélectionner le code qui viendra à minimiser au coup suivant la taille de l'ensemble.

7.2 Choix du code

Pour faire cela, nous allons calculer pour chaque code et chaque réponse possible de l'encodeur la taille de l'ensemble obtenu dans ce cas.
Le logarithme de cette valeur, rapporté à la taille de l'ensemble des codes actuellement possibles nous donne une valeur, que nous appellerons I.

$$I(c,r) = log(\frac{E(c,r)}{E})$$

avec E(c,r) la taille de l'ensemble si l'on propose le code c et qu'on obtient la réponse r E la taille de l'ensemble des solutions possibles.

information. Nous pouvons ainsi pour chaque code éligible calculer un vecteur contenant toutes les valeurs de I en fonction de toutes les réponses possibles de l'encodeur. Deux solutions s'offrent à nous :

1. Minimiser les risques : prendre le code qui a la valeur de I maximum la plus faible
2. Optimiser les risques : prendre le code qui a la plus basse valeur moyenne de I

7.2.1 Minimiser les risques

Cette approche a pour objectif de ne jamais atteindre un trop haut nombre d'essai. En effet, nous prenons la solution qui dans le pire des cas (i.e. la réponse menant au plus grand ensemble possible) à la taille d'ensemble la plus petite.
Cela est une approche qui ne parait pas amener de performances, si la pire solution mène à une taille d'ensemble pas trop grande cela veut dire qu'il n'y a pas de solution avec ce code menant à une taille de l'ensemble très petite, c'est à dire que les solutions sont assez centrées et groupées.

7.3 Optimiser les risques

Cette approche a pour conséquence de prendre plus de risques que la précédente. Cependant, elle parait plus optimisé. On va ainsi calculer pour chaque code la moyenne des valeurs I selon toutes les réponses. c'est pour cela que nous appliquons le logarithme lors du calcul de I, afin de ne pas se retrouver avec la même moyenne pour tous les codes. Cette approche parait plus sensé mais risque de conduire à un écart type plus important.

7.4 Résultats

Après avoir mené des expérimentations dans les mêmes conditions que pour l'algorithme génétique, nous avons obtenu les résultats suivants

Type	Temps				Essai			
Linéaire	Moyenne	Ecart Type	Max	Min	Moyenne	Ecart Type	Max	Min
Minimiser	2,82	2,2	23,64	0,004	4,49	0,93	7	1
Optimiser	3,14	2,31	20,69	0,004	4,35	0,76	7	1
GA	2,52	0,67	4,82	0,570	4,54	0,83	7	2

Nous pouvons ainsi constater que les résultats sont comparables, voir un peu meilleur au niveau du nombre d'essai pour l'approche linéaire.
On peut cependant constater que au niveau du temps, l'approche linéaire est bien inférieur à l'approche méta-heuristique. Bien qu'ayant des moyennes proches, on peut constater grâce à l'écart type et surtout au maximum que cette approche à tendance à faire diverger les modèles. Bien que pour cette taille il semble adapter, pour des modèles plus grands, le temps de traitement sera infiniment plus long, en effet, il croit de manière exponentielle avec la taille du modèle puisque l'on compare à chaque code, l'entièreté des codes possibles et cela plusieurs dizaines de fois par tentative. On a donc un modèle qui, même s'il parait intéressant au niveau des résultats des essais, n'est pas pérenne au niveau du temps.

8 Conclusion

Pour conclure sur ce projet, nous pouvons constater que l'approche que abordons avec l'algorithme génétique donne des résultats très convenables avec une taille commune du modèle, comparables avec les algorithmes plus "brutaux". Cependant, là où se situe sa vraie force est dans son évolution avec la taille du modèle, en effet le temps de traitement sera bien moindre que les autres algorithmes. Ce modèle parviendra donc à jouer dans des temps raisonnables à l'échelle humaines contre de très gros modèle là où les algorithmes linéaires n'en seront pas capables. Pour se rapprocher encore du seuil des trois essais, il faudrait se pencher sur le choix du premier code ainsi que l'optimisation des différents paramètres.