Stratégies et performances algorithmiques dans le Mastermind

TIPE de Jacques ARNAULD

N° de candidat : 41615

Thème : Jeux et Sport

Sommaire

- ❑ Introduction
- ❑ Stratégies pour résoudre le Mastermind
- ❑ Modélisation du jeu
- ❑ Fonctionnement et résultats des différents algorithmes
- ❑ Comparaison des algorithmes

Sommaire

- ❑ Introduction
  - Présentation du Mastermind
  - Problématique
- ❑ Stratégies pour résoudre le Mastermind
- ❑ Modélisation du jeu
- ❑ Fonctionnement et résultats des différents algorithmes
- ❑ Comparaison des algorithmes

Présentation du Mastermind

- Jeu de plateau
- Un **codeur** et un **décodeur**
- Jeu classique : **6 couleurs, 4 trous**
- Trouver le code en **au plus 12 coups**
- Couleur bien placée : **fiche rouge**
- Couleur mal placée : **fiche blanche**
- **Code trouvé : c’est gagné !**

Source de l’image : wikipedia.org

Problématique

En analysant les performances des algorithmes dans la résolution du Mastermind, comment peut-on comparer l'efficacité des différentes stratégies ?

Sommaire

- ❑ Introduction
- ❑ Stratégies pour résoudre le Mastermind
  - Force Brute
  - Recherche Heuristique
  - Type Knuth
- ❑ Modélisation du jeu
- ❑ Fonctionnement et résultats des différents algorithmes
- ❑ Comparaison des algorithmes

Stratégies pour résoudre le Mastermind

- **Stratégie Force Brute :**

- Elimination en fonction de la dernière proposition

- **Stratégie Heuristique :**

- Elimination en fonction de toutes les propositions précédentes

- **Stratégie de Knuth :**

- Optimisation du choix de la proposition retenue après une stratégie d'élimination
- Permet de trouver le code en au plus 5 coups

Sommaire

- ❑ Introduction
- ❑ Stratégies pour résoudre le Mastermind
- ❑ Modélisation du jeu
  - Correspondance entre code et 4-liste
  - Création liste complète
  - Fonction score
- ❑ Fonctionnement et résultats des différents algorithmes
- ❑ Comparaison des algorithmes

Modélisation du jeu

- **Correspondance entre un code et une 4-liste d'élément de 0 à 5**

- Exemple : la combinaison Bleu – Rouge – Vert – Bleu sous la forme [2,4,1,2]

- **Création de l'ensemble des 1296 propositions possibles sous forme de liste de codes**

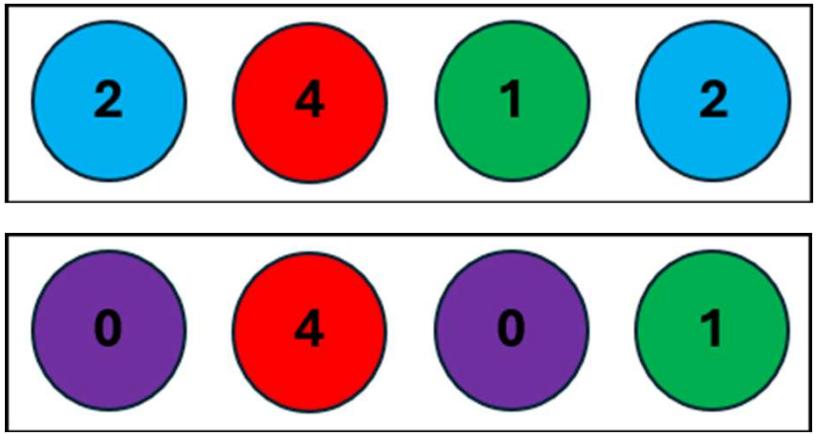
- E = [[0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 0, 2], [0, 0, 0, 3], ..., [5, 5, 5, 4], [5, 5, 5, 5]]

Modélisation du jeu

- **Création de la fonction score prenant en entrée 2 codes et renvoyant des entiers correspondant aux couleurs bien ou mal placées**

- Ensemble des scores possibles : sp = [0, 1, 2, 3, 4, 10, 11, 12, 13, 20, 21, 22, 30, 40]

- Exemple : score([2,4,1,2],[0,4,0,1]) = 11



Sommaire

- ❑ Introduction
- ❑ Stratégies pour résoudre le Mastermind
- ❑ Modélisation du jeu
- ❑ Fonctionnement et résultats des différents algorithmes
  - Algorithme de Force Brute
  - Algorithme de Recherche Heuristique
  - Algorithme Six-Guess de Type Knuth
  - Algorithme Five-Guess de Knuth
- ❑ Comparaison des algorithmes

Algorithme de Force Brute

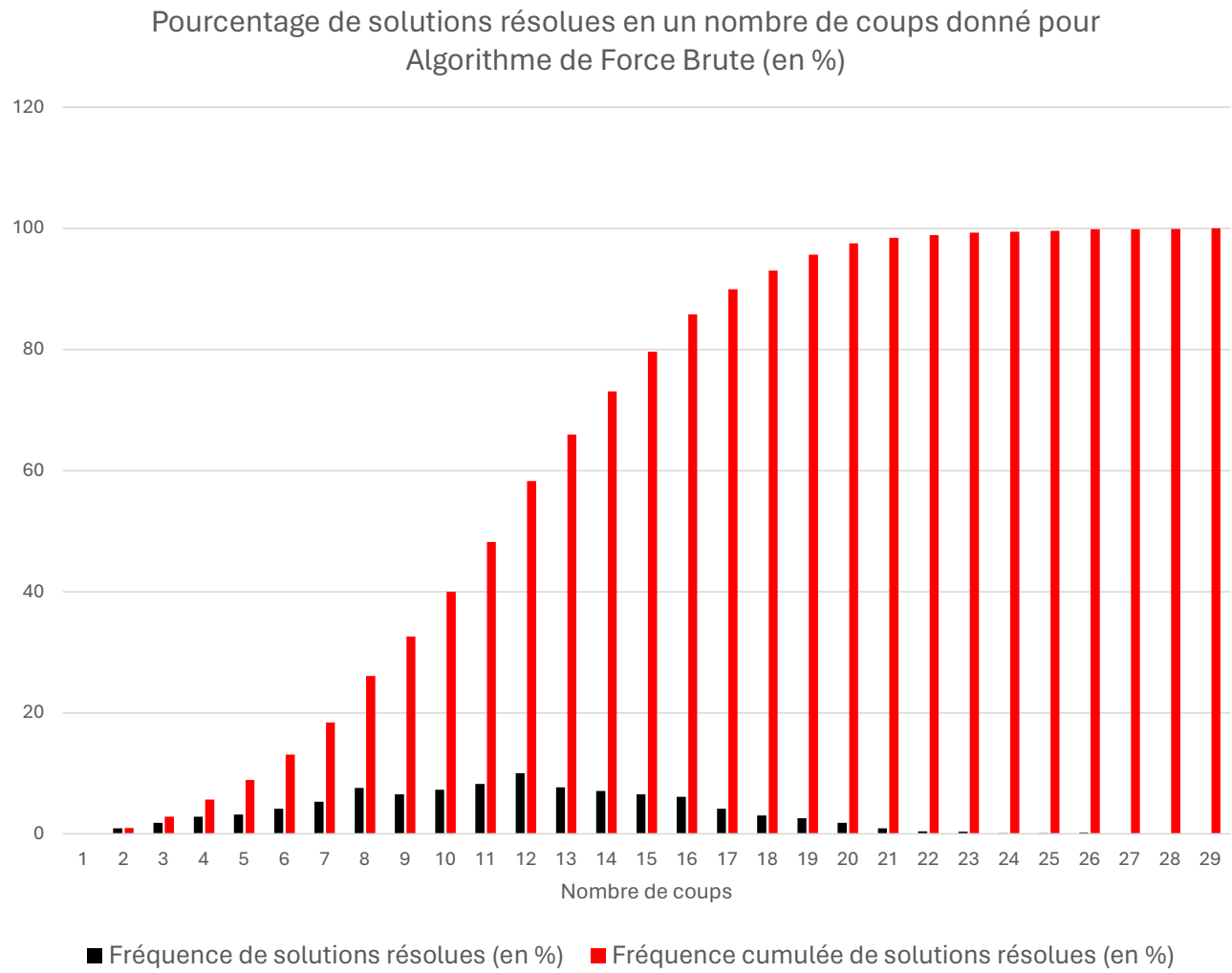
- **Avec en entrée : S (le code secret) et en sortie : n (le nombre de coups)**

1. Choisir une proposition initiale P_0
2. Soit sc_0 = score(S, P_0)
3. n=1
4. Répéter tant que sc_n(= score(S, P_n)) ≠ 40
    5. Choisir comme nouvelle proposition $P_n \in$ E la 1ère vérifiant :

       score(P_n, P_{n-1}) = sc_{n-1}
    6. Augmenter n de 1
7. Renvoyer n

Algorithme de Force Brute

- Maximum **29 coups > 12 coups** → **Inadapté** au jeu du Mastermind
- Résolution de la liste complète en **5,9 secondes**
- Moyenne de **11,7 coups**
- Plus de **90%** des **codes résolus** en au plus **18 coups**
- Proposition Initiale : **[0,0,1,1]**



Algorithme de Recherche Heuristique

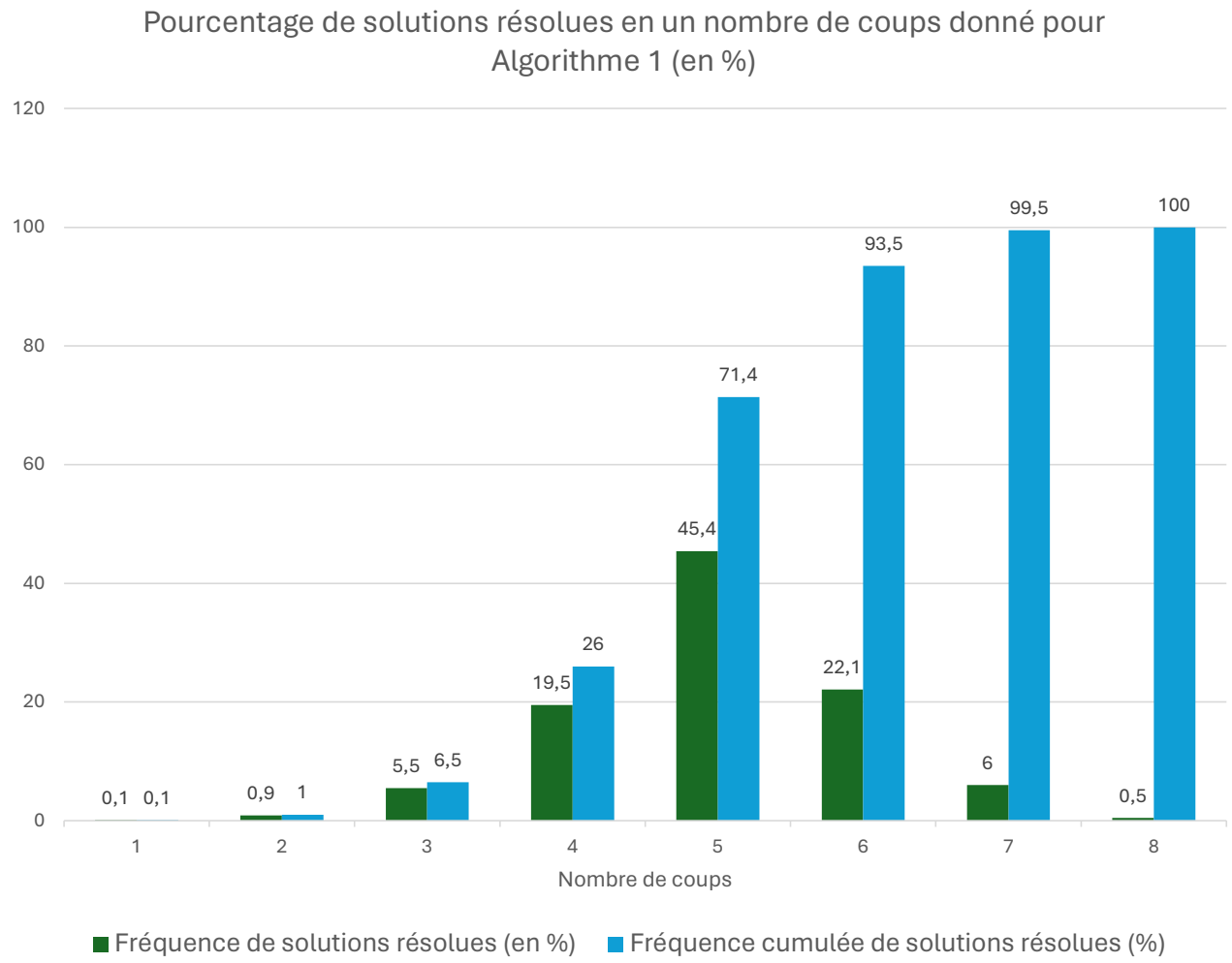
- **Avec en entrée : S (le code secret) et en sortie : n (le nombre de coups)**

1. Choisir une proposition initiale P_0
2. Soit sc_0 = score(S, P_0)
3. n=1
4. Répéter tant que sc_n(= score(S, P_n)) ≠ 40
    5. Choisir comme nouvelle proposition $P_n \in$ E la 1ère vérifiant :

       $\forall \boldsymbol{i} \in [\![\mathbf{0}, \boldsymbol{n-1}]\!]$, **score($\boldsymbol{P_n}$, $\boldsymbol{P_i}$) =** $\boldsymbol{sc_i}$
    6. Augmenter n de 1
7. Renvoyer n

Algorithme de Recherche Heuristique

- Maximum **8 coups < 12 coups** → **Adapté** au jeu du Mastermind
- Résolution de la liste complète en **7,2 secondes**
- Moyenne de **5,0 coups**
- Plus de **90%** des **codes résolus** en au plus **6 coups**
- Proposition Initiale : **[0,0,2,2]**



Fonction BestProp

- **Prend en entrée Candidats (une liste de propositions candidates) et renvoie P (une proposition optimale)**

- **Pour Six-Guess :**

- Attribue à chaque proposition de Candidats un poids, puis retient la 1ère proposition de poids minimum

- **Pour Five-Guess :**

- Attribue à chaque proposition de la liste complète un poids, puis retient la 1ère proposition de poids minimum, en priorité appartenant à Candidats

Fonction BestProp

1. Répéter pour chaque P ϵ E (P ϵ Candidats pour Six-Guess)
    2. Répéter pour chaque score possible s ϵ sp
        3. Déterminer le nombre c de candidats obtenant le score s en les comparant avec P
    4. Attribuer à la proposition P un poids égal au plus grand de 14 nombres précédents
5. Choisir comme nouvelle proposition la 1ère proposition ayant le plus petit poids, en priorité appartenant à Candidats (pour Five-Guess)

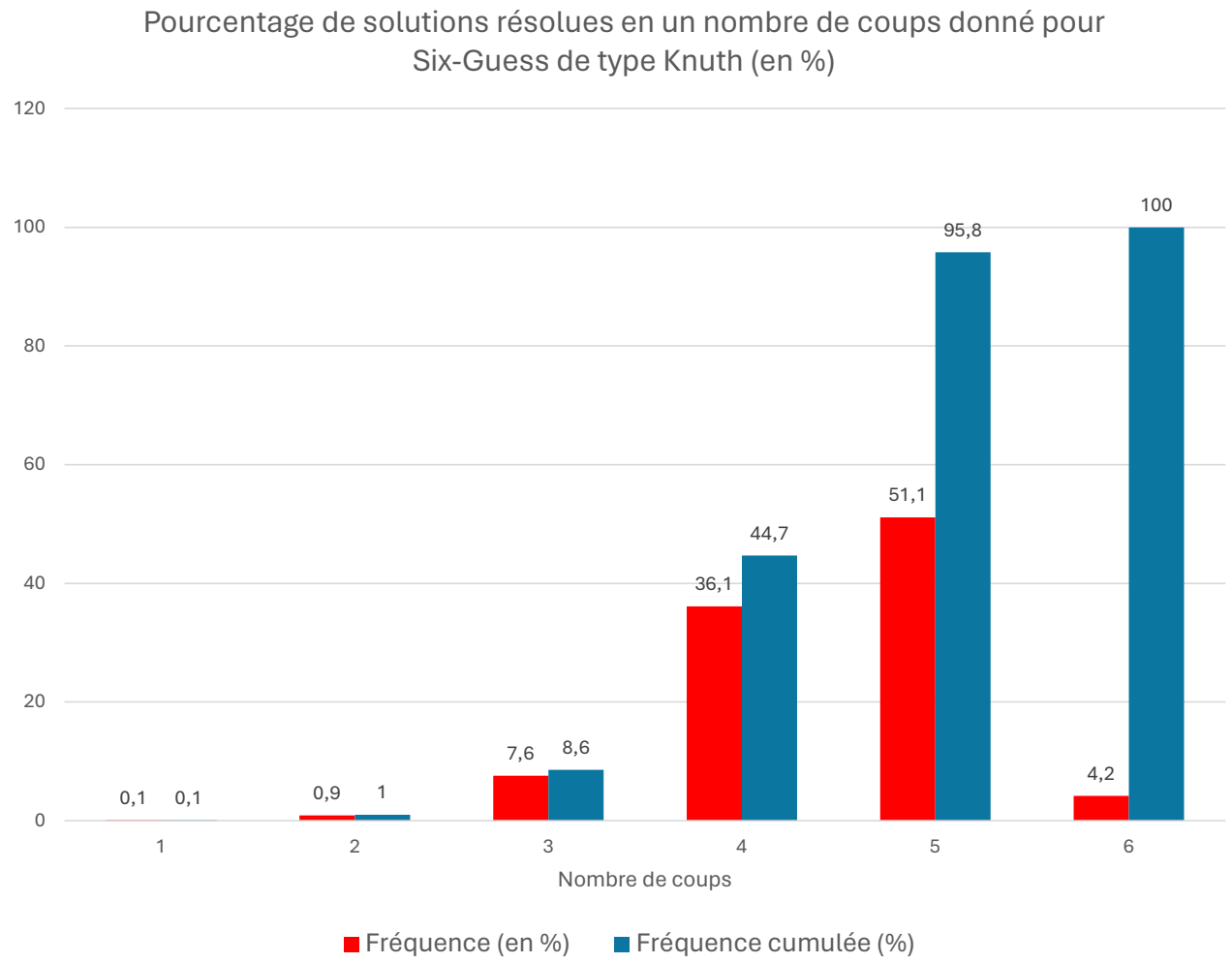
Algorithme de Type Knuth

- **Avec en entrée : S (le code secret) et en sortie : n (le nombre de coups)**

1. Choisir meilleure proposition initiale P avec BestProp en fonction de Candidats = E
2. Calculer sco = score(S,P)
3. n=1
4. Répéter tant que sco ≠ 40
    5. Mettre à jour la liste Candidats en éliminant tous les éléments C vérifiant : score(C,P) ≠ sco
    6. P = BestProp en fonction de Candidats
    7. Calculer sco = score(S,P)
    8. Augmenter n de 1

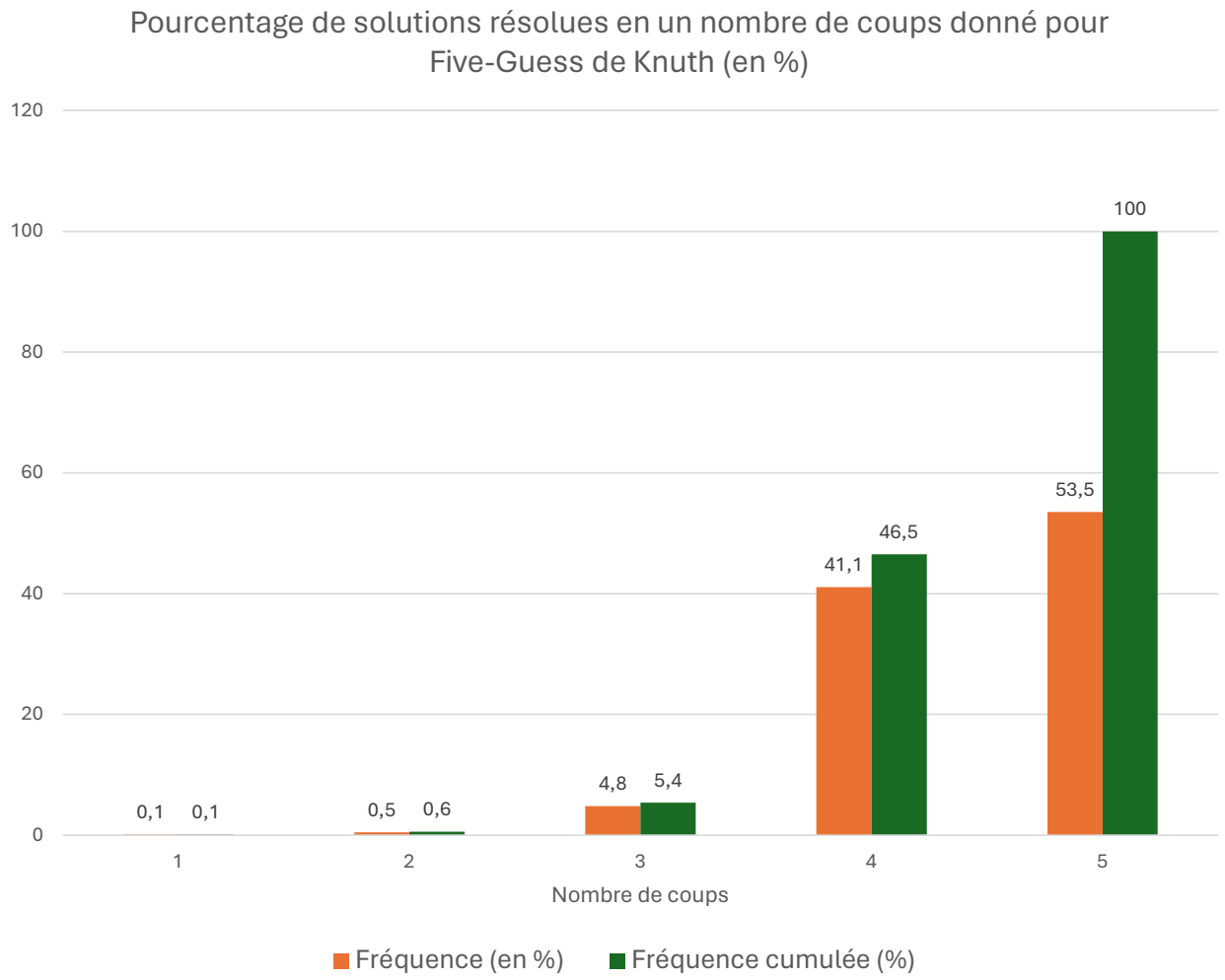
Six-Guess

- Maximum **6 coups < 12 coups** → **Adapté** au jeu du Mastermind
- Résolution de la liste complète en **4 min et 7 s**
- Moyenne de **4,5 coups**
- Plus de **90%** des **codes résolus** en au plus **5 coups**
- Ajout d'un **dictionnaire** → Résolution **16 fois plus rapide**



Five-Guess

- Maximum **5 coups < 12 coups** → **Adapté** au jeu du Mastermind
- Résolution de la liste complète en **38 min**
- Moyenne de **4,5 coups**
- 100% des **codes résolus** en au plus **5 coups**



Sommaire

- ❑ Introduction
- ❑ Stratégies pour résoudre le Mastermind
- ❑ Modélisation du jeu
- ❑ Fonctionnement et résultats des différents algorithmes
- ❑ Comparaison des algorithmes
  - Tableau des performances algorithmiques

Comparaison des Algorithmes

	Force Brute	Algo 1	Six-Guess	Five-Guess
Conformité aux règles	Inadapté	Adapté	Adapté	Adapté
Nb de coups maximal	29	8	6	5
Moyenne	11,7	5,0	4,5	4,5
Nb de coups pour +90% de codes résolus	18	6	5	5
Temps de résolution de E	5,9 ± 0,1 s	7,2 ± 0,1 s	4 min 7 s ± 2 s	37 min 55 s ± 49 s
Temps de résolution moyenne / code	4,5 ms Max : 5,5 ms	5,8 ms Max : 27 ms	0,19 s Max : 0,29 s	1,8 s Max : 2,4 s

Merci pour votre attention

Jacques ARNAULD

Annexes

- **Générer un code aléatoire**

```
import random
def code_alea(): # renvoyer une liste de 4 éléments allant de 0 à 5 aléatoirement
    Secret=[]
    for i in range(4): # nombre de trous
        x=random.randint(0,5) # associer une des 6 couleurs à chaque trou aléatoirement
        Secret.append(x) # ajouter le numéro de la couleur à la liste
    return(Secret) # renvoyer une liste de 4 éléments allant de 0 à 5 aléatoirement
```

Annexes

- **Générer l'ensemble des propositions possibles**

```
# renvoyer une liste de l'ensemble des listes allant de [0,0,0,0] à [5,5,5,5]
# nous piocherons dans cette liste les listes dont nous aurons besoin dans les algorithmes
def ens():
    E=[]
    for j0 in range(6):
        for j1 in range(6):
            for j2 in range(6):
                for j3 in range(6):
                    L=[j0,j1,j2,j3]
                    E.append(L)
    return(E)

E=ens()
```

Annexes

- **Fonction score**

```
# Prendre en entrée 2 propositions sous forme de 4-liste
# renvoyer un entier dont le chiffre des dizaines correspond au nombre de boules...
# ...de la même couleur et placées dans le même trou et le chiffre des unités correspond...
# ...au nombre de boules de la même couleur mais placées dans un trou différent
def score(S,P):
    r=0
    for i in range(4): # Pour chaque trou
        if S[i]==P[i]: # Si 2 boules ont la même couleur dans le même n° de trou
            r=r+1      # Ajouter 1 au chiffre des dizaines du score
    b=-r
    for j in range(6): # Pour chaque couleur
        n=0
        m=0
        for k in range(4): # Pour chaque trou
            if S[k]==j:    # Pour le  code secret si la boule d'emplacement k est de couleur j
                n=n+1      # augmenter le compteur n d'une unité
            if P[k]==j:    # Pour la  proposition si la boule d'emplacement k est de couleur j
                m=m+1      # augmenter le compteur m d'une unité
        if n<m:            # S'il y a plus de boules de couleur j dans la prop que dans le code
            b=b+n          # le chiffre des unités est n soustrait au chiffre des dizaines
        else:              # Sinon
            b=b+m          # le chiffre des unités est m soustrait au chiffre des dizaines
    s=10*r+b               # le score est le chiffre des unités additionné au chiffre des dizaines
    return(s)
```

Annexes

- **Jeu Mastermind Homme vs Ordinateur**

```
def demande_proposition():
    p = input("Proposition ? ") # taper la proposition que l'on souhaite avec un espace entre chaque chiffre
    return [int(x) for x in p.split(" ")] # renvoyer cette proposition sous forme de 4-liste

def jeu():
    S=code_alea()  # le code secret est aléatoire
    n=0            # nb de coups
    t=1
    P=[]
    while P!=S and n!=12: # tant que la dernière proposition n'est pas le code et que le nb de coups est <=12
        n+=1              # le nombre augmente de 1 à chaque boucle
        t=n
        P= demande_proposition() # On choisit la dernière proposition P
        assert len(P)==4         # s'assurer que P est une 4-liste de nombres entre 0 et 5
        for i in range(4):
            assert P[i]>=0 and P[i]<=5
        blanc=score(S,P)%10     # le nb de drapeaux blancs est le chiffre des unités du score entre S et P
        rouge=score(S,P)//10    # le nb de drapeaux rouges est le chiffre des dizaines du score entre S et P
        if P!=S:                # si P n'est pas S indiquer le nb de drapeaux rouges et blancs
            print('Vous avez ',rouge,' drapeau(x) rouge(s) et ',blanc,' drapeau(x) blanc(s)')
        else:                  # sinon indiquer en quel nombre de tentative on a gagner
            n=12
            print('Vous avez gagné en', t,'tentatives!')
    print('La solution était ',S) # si on perd indiquer le code secret
```

Annexes

- **Algorithme de Force Brute**

```
# trouver le code secret à partir de la proposition précédente et renvoyer le nombre de coups pour le trouver
def force_brute(S):
    n=1 # n correspond au nombre de coups
    P=[0,0,1,1] # P est ici la proposition initiale
    sc=[score(S,P)] # sc est la liste contenant le score entre la propositon initiale et le code secret
    if sc[0]==40:
        return(n)  # si P est le code secret renvoyer n=1
    else:
        while sc[n-1]!=40: # tant que P n'est pas le code secret
            for i in range(len(E)):
                if score(E[i],P)==sc[-1]: # si le score entre un élément de la liste complète avec P
                    P=E[i] # est le même que le score entre P et la solution affecter P à cet élément
                    sc.append(score(S,P)) # ajouter le score de P avec le code secret à sc
                    n+=1 # porter le nbre de coup à une unité de plus
    return(n) # renvoyer le nombre de coups n pour trouver le code secret
```

Annexes

- **Algorithme de Recherche Heuristique**

```
# trouver le code secret à partir des propositions précédentes et renvoyer le nombre de coups pour le trouver
def algo1(S):
    n=1 # n correspond au nombre de coups
    P=[0,0,1,1] # P est ici la proposition initiale
    L=[P] # L est la liste où seront stockées les propositions
    sc=[score(S,P)] # sc est la liste contenant le score entre la propositon initiale et le code secret
    if sc[0]==40:
        return(n) # si P est le code secret renvoyer n=1
    else:
        while sc[n-1]!=40: # tant que P n'est pas le code secret
            for i in range(len(E)): # pour parcourir la liste l'ensemble des solutions possible
                j=0
                while j<=n-1 and score(L[j],E[i])==sc[j]: #si le score entre les P et une liste a le même
                    j=j+1  #score que les P avec avec le code secret
                if j==n:
                    P=E[i] # affecter à P cette liste
                    L.append(P) # ajouter la proposition P à L
                    sc.append(score(S,P)) # ajouter le score de P avec le code secret à sc
                    n+=1
    return(n) # renvoyer le nombre de coups n pour trouver le code secret
```

Annexes

- **Algorithme choisissant la meilleure proposition parmi les candidats (pour Knuth et Five-Guess)**

```
sco=[0,1,2,3,4,10,11,12,13,20,21,22,30,40]


def BestProp(C): # Renvoie la meilleure proposition pour une liste de candidats donnéee
    n=0
    D={}
    L=[]
    P=[]
    for i in range(len(C)): # pour les propositions dans la liste des candidats
        for j in sco: # pour les scores dans la listes des scores
            for k in range(len(C)):  # pour les propositions dans la liste des candidats
                if (i,k) not in D: # si le score entre la liste C[i] et C[k] n'est pas dans le dictionnaire
                    D[(i,k)]=score(C[i],C[k]) # calculer ce score et le stocké dans D
                    D[(k,i)]=D[(i,k)] # le score étant symétrique, affecter ce score à (k,i)
                if D[(i,k)]==j:
                    n+=1 # compte nb de liste de candidats ayant le même score qu'une liste candidat donnée
            L.append(n) # faire correspondre à chaque score ce nombre de liste de candidat
            n=0
        P.append(max(L)) # score ayant le plus grand nb de solutions est le poids correspondant à un candidat
        L=[]
    ppp=min(P) # prendre le plus petit poids de la liste des candidats
    for i in range(len(C)):
        if P[i]==ppp: # correspond au premier candidat qui a le plus petit poids
            return(C[i]) # renvoyer ce candidat
```

```
def BestProp2(C):
    n=0
    L=[]
    P=[]
    D={}
    for i in range(len(E)):
        for j in sco:
            for k in range(len(C)):
                if (i,k) not in D:
                    D[(i,k)]=score(E[i],C[k])
                if D[(i,k)]==j:
                    n+=1
            L.append(n)
            n=0
        P.append(max(L))
        L=[]
    ppp=min(P)
    for i in range(len(E)):
        if P[i]==ppp:
            if E[i] in C:
                return E[i]
    for i in range(len(E)):
        if P[i]==ppp:
            return E[i]
```

Annexes

- **Algorithmes Knuth et Five-Guess**

```python
# Trouver le code secret en réduisant au maximum le nombre de candidats et...
# ... en choisissant la proposition la plus adaptée
def six_guess(S):
    C=E  # la liste des candidats initiale est la liste complète
    n=1 # on initialise le nombre de coups
    P=[0,0,1,1] # le proposition de la liste complète ayant le plus petit poids
    sc=score(S, P) # le score entre la solution et la proposition initiale
    if sc==40:
        return(n) # renvoyer 1 si la solution est la proposition initiale
    while sc!=40: # tant qu'on a pas trouver la solution
        L=[]
        for i in C: # pour les propositions dans la liste des candidats
            if score(P,i)==sc:
                L.append(i) # Ajouter à L candidats ayant le même score que la prop précédente avec la solution
        P=BestProp(L) # affecter à P la meilleure propostion parmi ces candidats
        C=L # pour l'éventuelle prochaine boucle affecter à C la liste des candidats
        sc=score(S,P) # calculer le score entre la nouvelle propositon et la solution
        n+=1 # augmenter le nombre de coups d'une unité
    return(n)
```

```python
def fiveguess(S):
    C=E
    n=1
    P=[0,0,1,1]
    sc=score(S, P)
    if sc==40:
        return(n)
    while sc!=40:
        L=[]
        for i in C:
            if score(P,i)==sc:
                L.append(i)
        P=BestProp2(L)
        C=L
        sc=score(S,P)
        n+=1
    return(n)
```

Annexes

- **Génerer statistiques d'un algorithme**

```
def stat_fb():
    L=[]
    M=[]
    n=1
    s=0
    for i in E:
        L.append(force_brute(i)) # Crée une liste correspondant au nbre de coups pour résoudre chaque solution
    while sum(M)!=len(E): # Tant que toutes les solutions ne sont pas résolues en un nbre de coup donné
        for i in L:
            if i==n:
                s+=1 # compte le nombre de solutions résolues en un nbre de coups
        M.append(s)
        s=0
        n+=1
    return(M)

#permet de voir le nombre de listes de la liste complète résolues en un nombre de coups donné
```

Annexes

- **Calcul de Moyenne et de Fréquences pour les stats**

```python
# Prend en entrée une liste de stats et renvoie la moyenne de nombre...
# ... de coups nécessaire pour trouver la solution sous forme flottants
def moyenne(R):
    n=0
    r=0
    for i in range(len(R)):
        n+= (i+1)*R[i]
        r+= R[i]
    return(round(n/r,2))

# Prend en entrée une liste de stats et renvoie la fréquence et la fréquence cumulée...
# ... (en %) pour chaque nombre de coups nécessaire sous forme de couple de flottants
def frequence(R):
    F=[]
    Fc=[]
    n=0
    for i in range(len(R)):
        f=(R[i]/len(E))*100
        F.append(round(f,1))
        n+=F[i]
        Fc.append(round(n,2))
    return(F,Fc)
```

Annexes

- **Calculer le temps d'exécution des algorithmes**

```
# prends en entrée un algorithme et un nombre repésentant le nombre de répétition de la résolution ...
# ... de la liste complète par cet algorithme
# renvoie le temps de résolution de la liste complète, le temps moyen de résolution par code, ...
# ... l'écart-type pour la résolution de la liste complète et le code prenant le plus de temps
def temps(f,n):
    L=[]
    F=[]
    M=[]
    for i in range(n):  # répeter n fois le programme suivant
        J=[]            # J est une liste vide
        for j in E:     # pour chaque code de la liste complète
            debut = time.time()
            f(j)
            fin = time.time()
            J.append((fin-debut))  # ajouter à J le temps d'exécution de chaque code
        L.append(J)                # après chaque résolution de E, ajouter J à la liste L
    for j in range(len(E)):      # pour chaque indice de code de la liste complète
        s=0                      # créer un compteur
        for i in range(n):       # pour le nombre de boucle
            s += L[i][j]         # sommer les temps d'exécution de chaque code
        F.append(s/n)            # F est une liste de la moyenne de temps d'exécution de chaque code
    maxi=max(F)                  # maxi est le temps d'exécution le plus long
    listecomp=sum(F)             # le temps d'exécution de la liste complète est la somme des codes
    for i in L:                  # pour chaque liste dans L
        M.append(abs(listecomp-sum(i)))  # calculer l'écart-type du temps d'exécution
    ecart = sum(M)/n
    return(round(listecomp,2),round(listecomp/len(E),4),ecart,maxi)
```