Projet MasterMind

1 Introduction

Ce document est mon rapport sur le projet consistant à étudier le problème de *MasterMind*. J'ai choisi d'étudier la version de *MasterMind* avec n **cases et** n **couleurs**, pour $n \geq 3$. Par la suite, la notation n désignera **toujours** le nombre de cases (et donc aussi le nombre de couleurs).

1.1 Algorithmes étudiés

Dans le cadre du projet, j'ai étudié les algorithmes suivants :

- *Randomized Local Search* (RLS)
- *Evolutionary Algorithms* (EA) avec ses différentes variantes :
  - $(\mu + \lambda)$-EA avec un taux de mutation p
  - (μ, λ)-EA avec un taux de mutation p
- *Genetic Algorithms* (GA) avec les deux variantes décrites dans le diaporama de la séance numéro 3 du cours MPRI 2.24.2.

Il ne s'agit que d'algorithmes élitistes. Etant donné que le problème à maximiser (*MasterMind*) ne possède pas de maximum local, il est possible de penser que des algorithmes non élitistes ne peuvent que moins performant relativement à leurs voisins élitistes. C'est pour cette raison que j'ai fait le choix de ne pas les étudier dans mon projet.

1.2 Problématique choisie

Dans le cadre de ce projet, j'ai fait le choix d'étudier les trois algorithmes précédents. Plus spécifiquement, j'ai étudié l'impact des paramètres sur les performances des algorithmes.

Les paramètres des différents algorithmes sont :

- RLS : Aucun paramètre
- EA : μ la taille de la population parente, λ la taille de la population fille, p le taux de mutation, et le choix entre « + » et « , ».
- GA :μ la taille de la population parente, λ la taille de la population fille, p le taux de mutation, c le taux de croisement, et le type de croisement.

1.3 Implémentation

J'ai implémenté les algorithmes et les expériences menées en C++. J'ai fait le choix **de privilégier la structure du code**, plutôt que l'optimisation de la performance. Plus précisément, j'ai implémenté les algorithmes étudiées **de manière totalement indépendante du problème étudié** (le *MasterMind*). *MasterMind* n'est, dans mon code, qu'un

paramètre des algorithmes. Je n'ai pas écrit ces algorithmes spécifiquement pour le problème étudié. Bien que j'ai fait attention d'optimiser les performances du code, j'en ai un peu perdu à cause de ce choix.

2 Etude du taux de mutation

Commençons par étudier le taux de mutation du *evolutionary algorithm*. Il a été annoncé dans le cours que le choix de $\frac{1}{n}$ comme taux de mutation était le plus commun. Vérifions cela.

2.1 Formalisation du problème

Notons T_{p_i} la variable aléatoire désignant le nombre de requêtes effectués par l'algorithme étudié sur le problème de *MasterMind* avec p_i comme taux de mutation. Notons μ_{p_i} l'espérance T_{p_i}.

On cherche le paramètre qui minimise le nombre de requêtes en moyenne, on cherche donc le paramètre p_i qui minimise μ_i, *ie* $p^* := \text{argmin}_{p_i} \mu_{p_i}$.

2.2 Expérience

Pour trouver ce paramètre parmi un ensemble E fini de paramètres, considérons cette expérience :

- Fixer un entier m et un petit réel $0 < \alpha < 1$.
- Pour chaque paramètre $p \in E$:
  - résoudre m instances de *MasterMind* avec l'algorithme étudiée en prenant p comme paramètre. On obtient $T_p^{(1)}$, $T_p^{(2)}$, ..., $T_p^{(m)}$ les nombres de requêtes des m expériences.
  - Calculer la moyenne empirique $\bar{T}_p := \frac{1}{m} \sum_{i=1}^{m} T_p^{(i)}$ et l'estimateur sans biais de la variance $S^2 := \frac{1}{n-1} \sum_{i=1}^{m} (T_p^{(i)} - \bar{T}_p)^2$.
  - Calculer $\alpha_p := \bar{T}_p - \frac{S}{\sqrt{m}} t_{\alpha/2}^{m-1}$ et $\beta_p := \bar{T}_p - \frac{S}{\sqrt{m}} t_{\alpha/2}^{m-1}$ où t_γ^k est le quantile d'ordre $1-\gamma$ de la loi de Student à k degrés de liberté.
- Calculer le seuil $\lambda := \min_p \beta_p$.
- Calculer $p_{\min} := \min\{p \in E : \alpha_p \leq \lambda\}$ et $p_{\max} := \max\{p \in E : \alpha_p \leq \lambda\}$
- Retourner l'intervalle $[p_{\min}, p_{\max}]$.

Explications :

Pour chaque paramètre $p \in E$, d'après la **loi de Student** (généralisation du théorème central limite, pour quand la variance théorique est inconnue), on a

$$\mathbb{P}[\alpha_p \leq \mu_p \leq \beta_p] = 1 - \alpha$$

Rappel : on recherche $p^* = \text{argmin}_{p \in E} \mu_p$.

Notons $\hat{p}$ le paramètre p tel que $\beta_{\hat{p}} = \lambda$, et calculons $\mathbb{P}[p^* \in [p_{\min}, p_{\max}]]$.

$$\begin{aligned}
\mathbb{P}[p^* \notin [p_{\min}, p_{\max}]] &\leq \mathbb{P}[\alpha_{p^*} > \lambda] \\
&= \mathbb{P}[\alpha_{p^*} > \lambda, \mu_{p^*} \leq \lambda] + \mathbb{P}[\alpha_{p^*} > \lambda, \mu_{p^*} > \lambda] \\
&\leq \mathbb{P}[\alpha_{p^*} > \mu_{p^*}] + \mathbb{P}[\mu_{p^*} > \lambda] \\
&\leq \mathbb{P}[\alpha_{p^*} > \mu_{p^*}] + \mathbb{P}[\mu_{\hat{p}} > \lambda], \text{ car } \forall p \in E, \mu_{p^*} \leq \mu_p \\
&= \mathbb{P}[\alpha_{p^*} > \mu_{p^*}] + \mathbb{P}[\mu_{\hat{p}} > \beta_{\hat{p}}] \\
&= \frac{\alpha}{2} + \frac{\alpha}{2} = \alpha
\end{aligned}$$

donc

$$\mathbb{P}[p^* \in [p_{\min}, p_{\max}]] \geq 1 - \alpha$$

L'expérience construite permet de déterminer un intervalle de confiance à 95% pour le paramètre optimal.

2.3 Application

J'ai appliqué l'expérience sur mon implémentation de $(1+1)$-EA avec :

- n variant de 3 à 50,
- $(1 - \alpha) = 95\%$

Les figures 1 et 2 montrent le résultat de l'expérience. Avec la figure 2, il semble assez clair que la valeur optimale du taux de mutation est proportionnelle à $1/n$. Sachant que la courbe de $1/n$ est dans l'intervalle de confiance à 95%, on peut considérer que cette valeur est validée.

2.4 Étude du taux de croisement

A présent, passons aux algorithmes génétiques. Ces algorithmes dépendent de différents paramètres : taille de la population parent, taille de la population enfant, taux de mutation (*mutation rate*), taux de croisement (*crossover rate*)...

J'ai déjà étudié le taux de mutation. Passons au taux de croisement. L'idée est de trouver le taux de croisement qui minimise avec le nombre de requête, pour une taille de problème fixé (pour n fixé) et pour une taille de populations donnée. Par souci de simplification, je vais considérer que la taille des deux populations est égale ($\mu = \lambda$).

Notons $c(n, \lambda)$ le taux de croisement qui minimise, en moyenne, le nombre de requêtes. On souhaite connaître la forme de $c(n, \lambda)$.

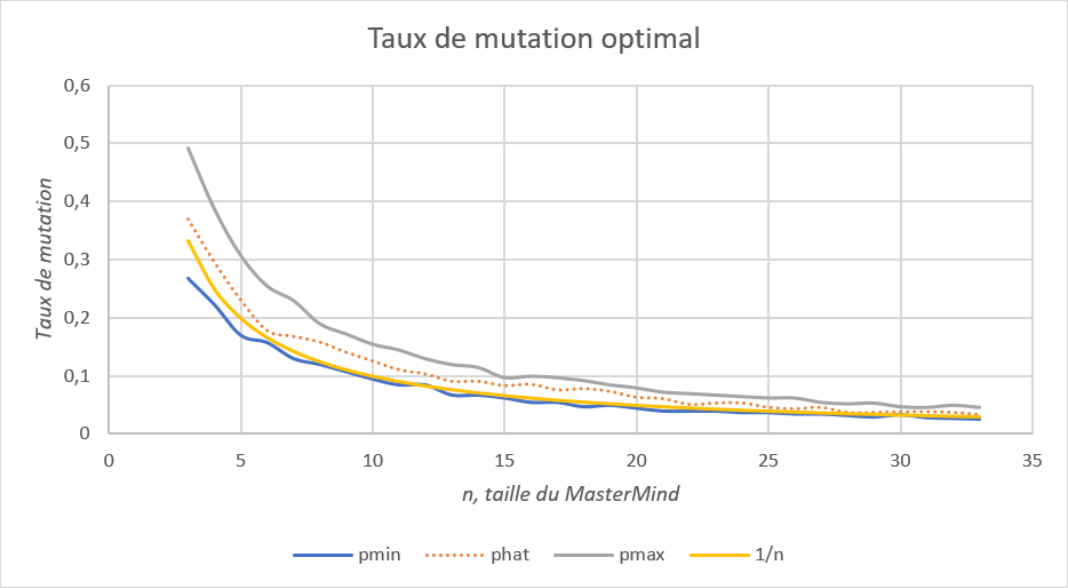

FIGURE 1 – Taux de mutation optimal

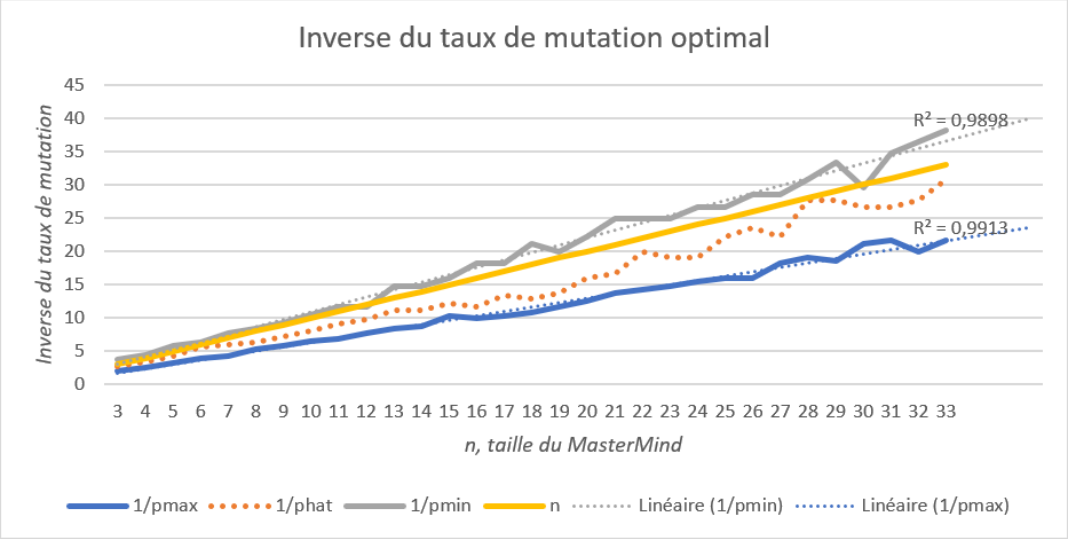

FIGURE 2 – Inverse taux de mutation optimal

2.4.1 Protocole expérimentale

Pour répondre à la problématique, on va appliquer l'expérience « Optimisation de paramètre » pour des n et des λ différents.

A partir de ces données produites, on va déterminer la forme de $\lambda \mapsto c(n, \lambda)$ pour n fixé. Puis, on va regarder l'impact de n sur la fonction précédente pour extrapoler $(n, \lambda) \mapsto c(n, \lambda)$.

2.4.2 Expérience

J'ai fait réaliser tourner l'expérience « Optimisation de paramètre » sur $GA(\lambda+\lambda)$, avec n fixé, pour déterminer un intervalle de confiance à 95% du taux de croisement optimal :

- n allant de 5 à 19 (inclu)
- λ allant de 5 à 100
- précision de taux de croisement optimale : 0.01

Le temps de calcul de l'expérience est de 48h pour un coeur de $2,5$ GHz. Cela a produit $15 \times 95 = 1425$ intervalles de confiance. Les données produites sont dans le fichier *crossover-rate.csv*.

2.4.3 Taux de croisement pour n fixé

J'ai testé deux modèles de fonctions :

- $c_n(\lambda) = 1 - \frac{\alpha_n}{\lambda}$
- $c_n(\lambda) = 1 - \exp^{-\alpha_n \lambda + \beta_n}$

J'ai donc tracé les courbe pour n fixé (figure 3 pour $n = 10$), et j'ai testé les deux modèles (figure 4 pour $n = 10$). Le modèle de la fonction inverse est peu probable, mais par contre, le modèle de l'exponentiel semble pertinent !

2.4.4 Taux de croisement pour n variant

En utilisant des régressions linéaires, j'en déduis les coefficients α_n et β_n de $c_n(\lambda) = 1 - e^{-\alpha_n \lambda + \beta_n}$, pour n variant de 5 à 19, pour les deux extrémités des intervalles de confiance. Pour avoir des courbes propres, on retire les données avec un λ et n faible. En effet, comme on peut voir sur la figure 5 ($\lambda > 60$), la borne supérieure est très proche de 1 et comme l'expérience d'optimisation de paramètre dépend d'un ensemble E fini, on tombe sur un problème d'imprécision.

D'après la figure 5, on a que

$$\ln(\alpha_n) \approx -0,1433n - 2,0602$$

Donc,

$$\alpha_n \approx e^{-0,1433n-2,0602}$$

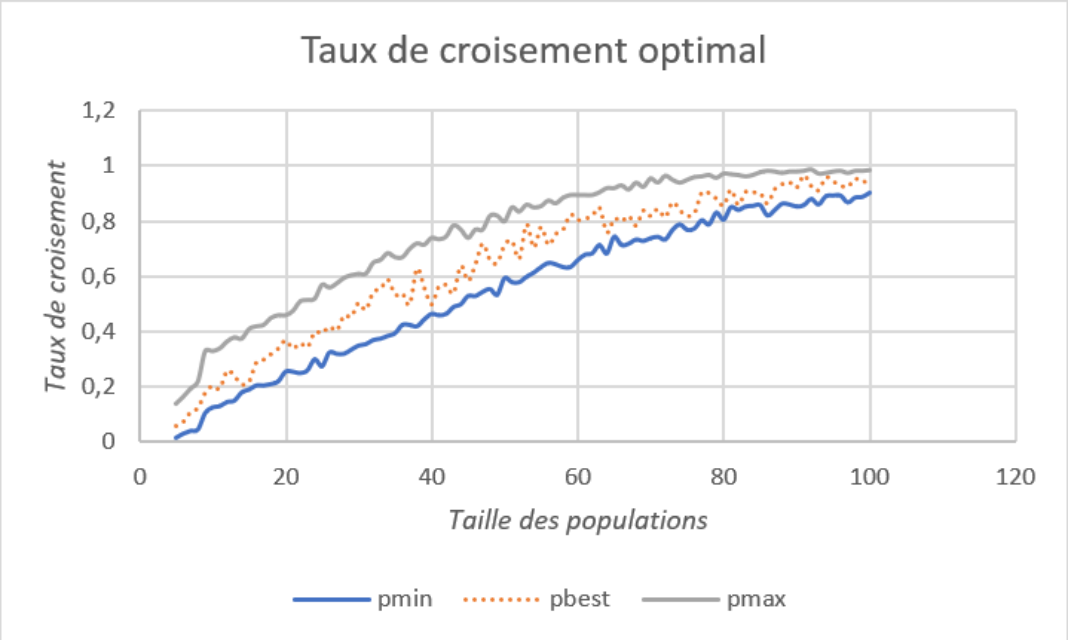

FIGURE 3 – Taux de croisement optimal ($n = 10$)

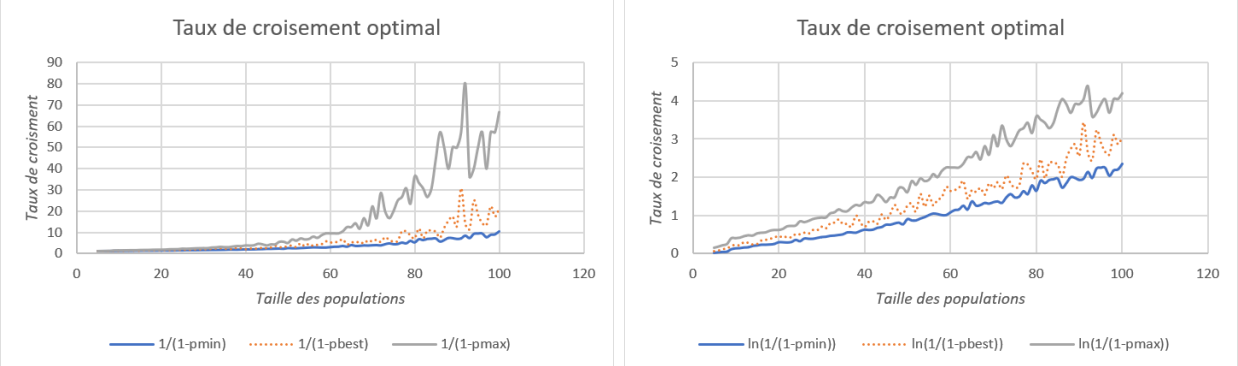

FIGURE 4 – Modélisation du taux de croisement optimal ($n = 10$)

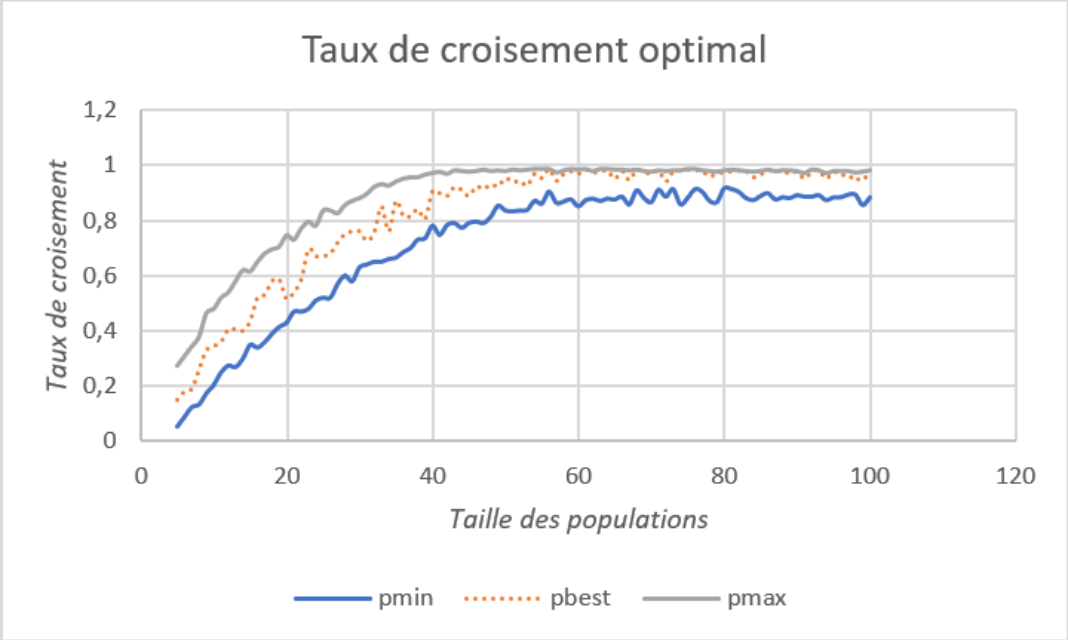

FIGURE 5 – Taux de croisement optimal ($n = 6$)

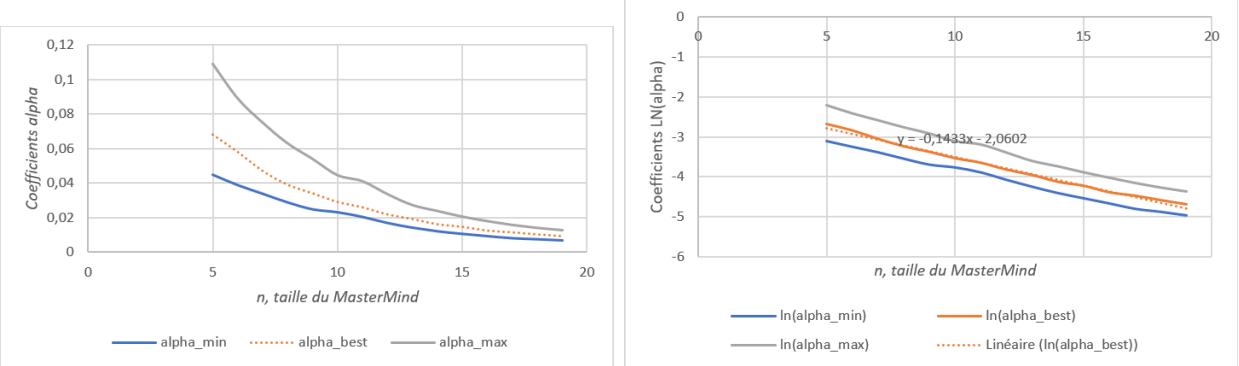

FIGURE 6 – Détermination du coefficient α_n

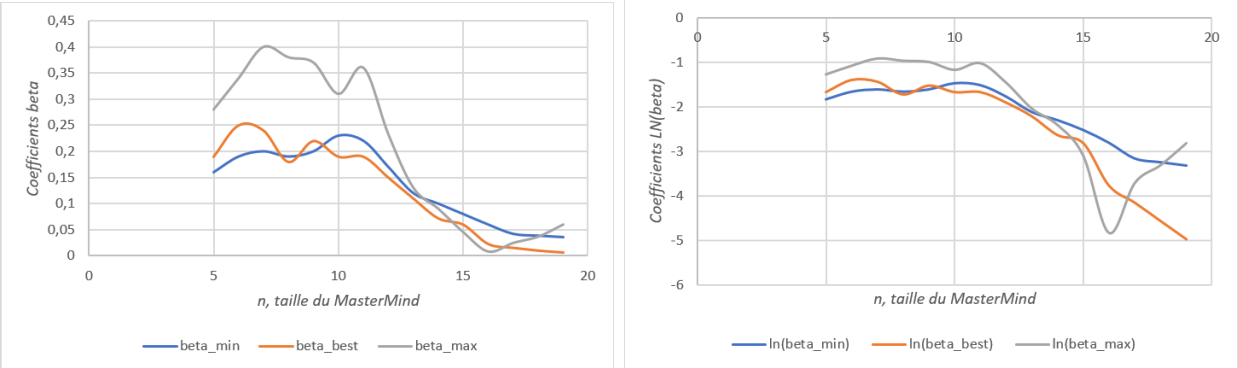

FIGURE 7 – Détermination du coefficient β_n

Pour β_n, la courbe est un peu plus erratique (figure 6). Mais si on ignore les points pour n petit, on peut imaginer un semblant de comportement exponentielle, comme pour α_n. Sous cette hypothèse, avec une régression linéaire, on trouve

$$\beta_n \approx e^{-0,43n-3,4}$$

In fine, on trouve

$$c(n,\lambda) = 1 - \exp(-e^{-0,1433n-2,0602}\lambda + e^{-0,43n-3,4})$$

Testons le modèle. Sur les données de l'expérience, est-ce que la modélisation de $c(n,\lambda)$ est situe dans les intervalles de confiances ? Réponse :

- Pour $93,3\%$ des données, le modèle se situe dans les intervalles de confiance.
- Si l'on retire les donnés où il y a le problème d'imprécision, on tombe sur $98,5\%$.

2.4.5 Amélioration des performances de la modélisation

Le modèle de $c(n,\lambda)$ dépend de quatre valeurs. Actuellement, il y a $0,1433$, $2,0602$, $0,43$ et $3,4$. Si la pente $(0,1433)$ du coefficient α_n est clair. On peut remarquer que l'ordonnée à l'origine peut varier entre $1,55$ et $2,40$. De plus, le modèle exponentielle pour β_n n'est pas 100% précis, donc les valeurs optimales de la pente et de l'ordonnée à l'origine pourrait fluctuer.

Pour cette raison, on peut essayer de faire varier un peu les 3 dernières valeurs pour obtenir un meilleur pourcentage. Ainsi, je tombe sur le nouveau modèle

$$c(n,\lambda) = 1 - \exp(-e^{-0,1433n-2,15}\lambda + e^{-0,3n-0,22})$$

dont les performances sont :

- Pour $96,8\%$ des données, le modèle se situe dans les intervalles de confiance.
- Si l'on retire les donnés où il y a le problème d'imprécision, on tombe sur $99,9\%$.

2.5 Conclusion sur les valeurs de taux

Le taux de mutation optimal est inversement proportionnelle à la taille du problème. Il devient rapidement faible pour des valeurs élevées de n. Dans le reste du projet, j'utiliserai

$$p(n) = \frac{1}{n}$$

Le taux de croisement optimal est également décroissant avec la taille du problème. Par contre, c'est croissant avec la taille des populations. Pour la suite du projet, j'utiliserai

$$c(n, \lambda) = 1 - \exp(-e^{-0,1433n-2,15}\lambda + e^{-0,3n-0,22})$$

3 Comparaison entre algorithmes

3.1 Random Local Search (RLS)

3.1.1 Etude théorique

Quelle est la complexité sur MasterMind ? Utilisons la *fitness level method* pour en avoir une borne supérieur.

Notons p_i une borne inférieur que RLS passe d'un candidat avec i couleurs en commun avec la solution à un candidat strictement mieux.

$$\begin{aligned} p_i &= \mathbb{P}[\text{changer une case pas optimisée}].\mathbb{P}[\text{trouver la bonne couleur}] \\ &= \frac{n-i}{n}\frac{1}{n-1} \end{aligned}$$

Donc,

$$\mathbb{E}[T(RLS, MasterMind)] \leq \sum_{i=0}^{n-1} \frac{1}{p_i} = \sum_{i=0}^{n-1} \frac{n(n-1)}{n-i} = n(n-1)\sum_{i=1}^{n} \frac{1}{i}$$

On en déduit que

$$\mathbb{E}[T(RLS, MasterMind)] = \mathcal{O}(n^2 log(n))$$

3.1.2 Vérification expérimentale

A faire : Placer l'évolution de la performance en fonction de n ***pour RLS, ainsi que la linéarisation en*** $n^2log(n)$***.***

3.2 Evolutionary Algorithm (EA)

Reprenons les notations de *RLS*. Conformément à la première partie de ce rapport, on va prendre $\frac{1}{n}$ comme taux de mutation.

$$\begin{aligned} p_i &\geq \mathbb{P}[\text{changer qu'une seule case}] \times \mathbb{P}[\text{que celle-ci ne soit pas déjà optimisée}] \\ &\qquad \times \mathbb{P}[\text{trouver la bonne couleur}] \\ &= \binom{n}{1}\left(\frac{1}{n}\right)^1\left(1-\frac{1}{n}\right)^{n-1} \times \frac{n-i}{n} \times \frac{1}{n-1} = \frac{n-i}{n(n-1)}\left(1-\frac{1}{n}\right)^{n-1} \\ &\geq \frac{n-i}{en(n-1)} \end{aligned}$$

Donc,

$$\mathbb{E}[T((\mu+\lambda)EA, MasterMind)] \leq \sum_{i=0}^{n-1} \frac{1}{p_i} = en(n-1)\sum_{i=1}^{n} \frac{1}{i}$$

On en déduit que

$$\mathbb{E}[T((\mu+\lambda)EA, MasterMind)] = \mathcal{O}(n^2 log(n))$$

3.3 Vérification expérimentale

A faire : Placer l'évolution de la performance en fonction de n ***pour RLS, ainsi que la linéarisation en*** $n^2log(n)$***.***

3.4 Genetic Algorithm (GA)