**I Principe de la méthode**

La méthode de cross-entropy (CE) est adaptée non seulement pour estimer les probabilités d'événements rares, mais également pour résoudre des problèmes complexes d’optimisation combinatoire difficile.

La méthode CE présente une méthode simple, efficace et générale pour résoudre une grande variété de problèmes d’estimation et d’optimisation. La méthode CE est un outil précieux pour la simulation de Monte-Carlo en particulier lorsque de très faibles probabilités doivent être estimées avec précision (ce qu'on appelle simulation d'événements rares).

Dans le domaine de la simulation d'événements rares, la méthode CE est utilisée conjointement avec un « importance sampling » (IS), une technique bien connue de réduction de la variance.

En pratique, les paramètres de référence optimaux à utiliser en IS sont généralement très difficiles à obtenir et nécessitent de faire appel à des programmes de minimisation de la variance (VM). L'avantage du CE est de fournir une procédure adaptative simple et rapide pour estimer les paramètres de référence optimaux dans l’IS. De plus, la méthode CE s’accompagne de « bonnes » propriétés de convergence asymptotique.

De nombreuses optimisations combinatoires (COP) peuvent être formulés comme des problèmes d’optimisation concernant un graphique pondéré. En fonction du problème particulier, l'ASP introduit

Il convient de souligner que la méthode CE peut être appliquée à la fois COP déterministes et stochastiques. Dans ce dernier, la fonction objective elle-même est aléatoire ou doit être estimée par simulation.

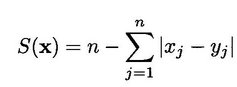
Récemment, il a été constaté que la méthode CE a un lien étroit avec les domaines du calcul neuronal (neural computation) et de l'apprentissage par renforcement (reinforcement learning). En effet, l'algorithme CE peut être vu comme un algorithme d’apprentissage impliquant les deux phases itératives suivantes :

1. Génération d’un échantillon de données aléatoires (trajectoires, vecteurs, etc.) selon un mécanisme spécifié.

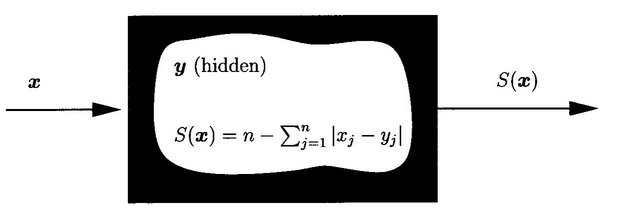
2. Mise à jour des paramètres du mécanisme aléatoire (généralement des paramètres de la fonction de densité) en fonction des données, afin de produire un «meilleur» échantillon lors de la prochaine itération.

**II Exemple d'optimisation combinatoire**

Considérons un vecteur binaire y = (y\_1, …, y\_n). Supposons que nous ne savons pas quelles composantes de y valent 0 et lesquelles valent 1 mais que, par contre, nous disposons d’un « oracle » qui pour chaque vecteur d'entrée binaire x = (x\_i,..., X\_n) renvoie le nombre de correspondances entre les éléments de x et y (Score)



L’algorithme permet alors de retrouver la valeur du vecteur inconnu y en maximisant la fonction de score.

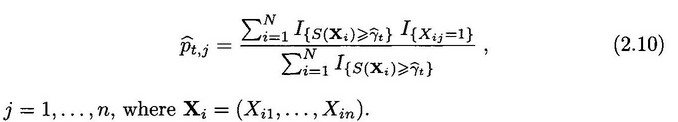


Une manière naïve consiste à générer N vecteurs binaires X = (X\_i,..., X\_n) tels que Xi,. . . , Xn sont des variables aléatoires de Bernoulli indépendantes X ~ Ber (p), où p = (pi,..., Pn)

Algorithme

Initialisation : Choisir p\_0\_hat = (p\_0\_1\_hat, …, p\_0\_n\_hat). Fixer t = 1

1. Générer un échantillon X\_1,. . . , X\_N de N vecteurs de Bernoulli avec probabilité de succès p\_t-1\_hat.
2. Calculer les S(X\_i) et ordonnez les du plus petit au plus grand, S(1)… S(N)
3. Calculer 
4. Calculer p\_t\_hat = (p\_t\_1\_hat, …, p\_t\_n\_hat) grâce à la formule



1. Si le critère d'arrêt est satisfait, arrêtez-vous; sinon incrémentez t et repartez de l’étape 1.  
   Remarque : Un critère d'arrêt possible est de s'arrêter lorsque gamma\_t\_hat ne change pas à l’itération suivante. Un autre critère d'arrêt possible est de s’arrêter lorsque le vecteur p\_t\_hat a convergé vers un vecteur binaire.

**III Problème d’Optimisation combinatoire (COP) via entropie croisée**

Dans ce chapitre, nous montrons comment la méthode CE peut être facilement transformée en un algorithme randomisé efficace et polyvalent pour résoudre les problèmes d’optimisation des problèmes d’optimisation combinatoire.

L'idée est de partitionnée une région en sous-régions plus petites jusqu'à ce que certaines des sous-régions ne contiennent qu'un seul point. La méthode se déplace ensuite d'une région à l'autre sur la base d'informations obtenues par échantillonnage aléatoire.

Les algorithmes des fourmis sont basés sur le comportement des colonies de fourmis. Il est connu que les colonies de fourmis sont capables de résoudre les problèmes les plus courts dans leur environnement naturel. En s'appuyant sur un mécanisme biologique assez simple: en marchant,les fourmis déposent au sol une substance chimique appelée phéromone. Les fourmis ont tendance à suivre ces sentiers de phéromones. Dans un délai fixe, plus court les chemins entre le nid et la nourriture peuvent être parcourus plus souvent que les chemins plus longs, et donc ils obtiennent une plus grande quantité de phéromone, ce qui, à son tour, tente un un plus grand nombre de fourmis pour les choisir et ainsi les renforcer à nouveau.

L'idée principale derrière l'utilisation de CE pour les COP est d'abord de s'associer à chaque COP un problème d'estimation des événements rares. L’idée est de construire une séquence aléatoire de solutions

**4.2 L'algorithme CE principal pour l'optimisation**

Supposons que nous souhaitons maximiser une fonction de performance S(x) sur tous les états x dans un ensemble X.



**Remarque :** Le maximiseur de S dans la proposition 4.2 peut être assurée si on impose un ordre dans l’ensemble fini X (par exemple l’ordre lexicographique).

Nous introduisons de l’aléatoire dans notre problème déterministe en définissant une famille de densités  sur l'ensemble X. Le problème d’estimation est le suivant (l’ASP : associated stochastic problem) :



Ici, X est un vecteur aléatoire de densité (par exemple, X pourrait être un Bernoulli aléatoire) et gamma est un paramètre (connu ou inconnu). Pour résoudre (4.3), on peut soit estimer l en fixant gamma, soit l’inverse. L’événement  est rare, c’est pourquoi l’estimation peut s’avérer complexe.

L’idée de l’algorithme est de définir une suite de couples  qui converge rapidement vers 

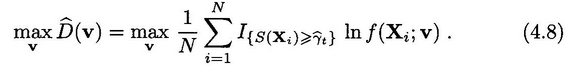
Choix des paramètres : Choisir un rho petit (10^-2). Choisir alpha le paramètre de smoothing (entre 0,7 et 1, prendre 0.7). Choisir d (d=5). Fixer N = C\*n\*m, où C constante (C = 5 par exemple).

Initialisation : Choisir v\_0\_hat = (p\_0\_1\_hat, …, p\_0\_n\_hat). Fixer t = 1

1. Générez un échantillon X\_1, ..., X\_N à partir de la densité  et calculer le (1 - rho) -quantile de gamma\_t\_hat : 

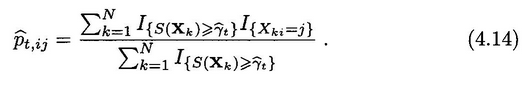
**Remarque :** En général, il existe de nombreuses façons de randomiser la fonction objectif S(x) et donc de générer des échantillons à partir de X, tout en estimant la probabilité d'événements rares l(gamma) de l'ASP.

1. Utilisez le même échantillon X\_1, ..., X\_N et résoudre le programme stochastique suivant en appelant la solution v\_t\_tilde



**Remarque**: Le principal avantage de la méthode CE par rapport à la minimisation de la variance est que dans la plupart des problèmes d'optimisation combinatoire, la mise à jour de V\_t peut souvent être fait analytiquement, c'est-à-dire qu'il n'y a pas besoin d'optimisation numérique.

**Remarque :** Dans le cas de mastermind, la solution analytique du programme 4.8 pour le paramètre pij est donné par



1. Appliquer l’équation suivante pour lisser le vecteur v\_t\_tilde  
   

**Remarque :** la raison de lisser v\_t\_tilde est double :   
(a) pour lisser les valeurs de v\_t\_hat  
(b) pour réduire la probabilité qu'une composante v\_t\_i\_hat soit nulle ou égale à un dès les premières itérations. Ceci est particulièrement important dans le cas de matrice de probabilités. Cela impliquerait que l’algorithme puisse converger vers les mauvaises solutions. Ainsi on aura toujours v\_t\_i\_hat > 0

1. Si pour certains t> d (d = 5 par exemple) l’équation suivante est vérifiée, arrêtez l’algorithme. Sinon reprenez à l’étape 1.



**IV Application au mastermind**

Dans le cas du jeu du mastermind, nous appliquons la méthode CE dans un cadre de « machine learning ». En plus de sa simplicité, l'avantage d'utiliser le CE en machine learning est qu'il ne nécessite pas d'estimation directe de gradients, comme le font de nombreux autres algorithmes. De plus, en tant que procédure d'optimisation globale, la méthode CE est assez robuste en ce qui concerne les initialisations et les erreurs d'échantillonnage.

L’objectif du mastermind est de deviner un « code » de billes (n billes) colorées (m couleurs numérotées de 1 à m), à la suite de plusieurs propositions. La solution cachée y et les propositions x sont donc représentées par des vecteurs ligne de longueur n avec des nombres entre 1 et m. Soit X l’ensemble des solutions possibles. Le nombre total de possibilités est m^n.

Après chaque nouvelle proposition, des informations sur le code sont fournies sous la forme de billes noires (billes exactement au bon endroit) et blanches (bille dans la solution, mais dans la mauvaise position)

Sur ***X***, on définit une fonction de performance S qui retourne pour chaque supposition x un « score » calculé comme suit :

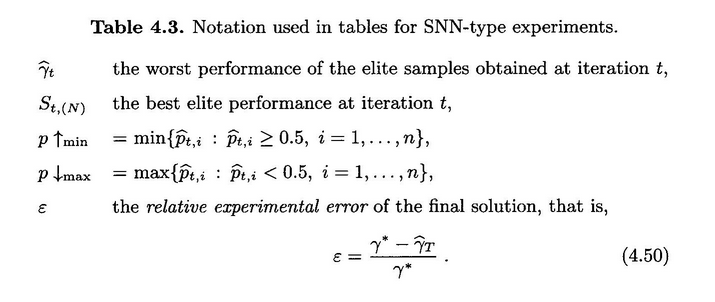


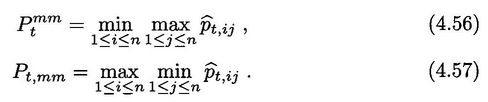
Avec NBlack et NWhite respectivement le nombre de billes noires et blanches obtenues après avoir fait une proposition. Ici, le choix de donné un poids de 2 aux billes noires et d’un aux billes blanches est arbitraire et aurait pu être autrement.

Exemple : Pour n=5 et m=7, Soit y = (4, 2,4, 7, 3) la solution cachée et x= (4, 3,4, 2,5) une proposition. On obtient en réponse une billes blanches (2e et 4e bille) et deux billes noires (1e et 3e billes). Par conséquent, le score pour cette supposition est de 6.

Avec la fonction de performance ci-dessus (le score), le problème peut être formulé comme un problème d'optimisation de la forme (4.2) que nous pouvons résoudre avec une méthode de CE. Pour ce faire, on génère un vecteur X appartenant à ***X*** de réponses aléatoires. Nous pouvons le faire grâce à la création d’une matrice P de taille n x m. Chaque élément pij de P décrit la probabilité que de choisir la j-ème couleur pour la i-ème bille. Puisqu'une seule couleur peut être assignée à une bille, la somme des lignes de la matrice est toujours égale à 1. A chaque itération t de l’algorithme, nous échantillon indépendamment pour chaque ligne (c’est-à-dire chaque bille) une couleur utilisant la matrice de probabilité P\_t\_hat, puis calculons le score selon la formule (8.1). La mise à jour des éléments P\_t\_hat\_ij de la matrice P\_t\_hat est effectuée selon la formule (4.14)[[1]](#footnote-1).

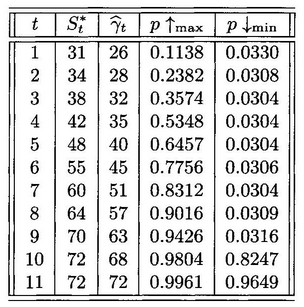
**V « Evaluation » du modèle**





fdiff is the proportion, out of N, of differentvalues for the performance function.

pour mastemermind : 34 secondes pour n = 36, m = 33, N=5\*m\*n, rho = 0.01, alpha = 0.7



Algorithm 4.1 (Generation of permutations (tours) in the TSP)

1. De\_ne P(1) = P and X1 = 1. Let k = 1.

2. Obtain P(k+1) from P(k) by \_rst setting the Xk-th column of P(k) to 0

and then normalizing the rows to sum up to 1. Generate Xk+1 from the

distribution formed by the Xk-th row of P(k).

3. If k = n􀀀1 then stop; otherwise set k = k +1 and reiterate from step 2.

It is important to realize that the updating formula remains the same; by

using Algorithm 4.1 we are merely speeding-up our naive way of generating the

tours. Moreover, since we now only generate tours, the updated value for pij

can be estimated as

bpij =

XN

k=1

IfS(􀀀 k)\_g If􀀀 k2Xijg

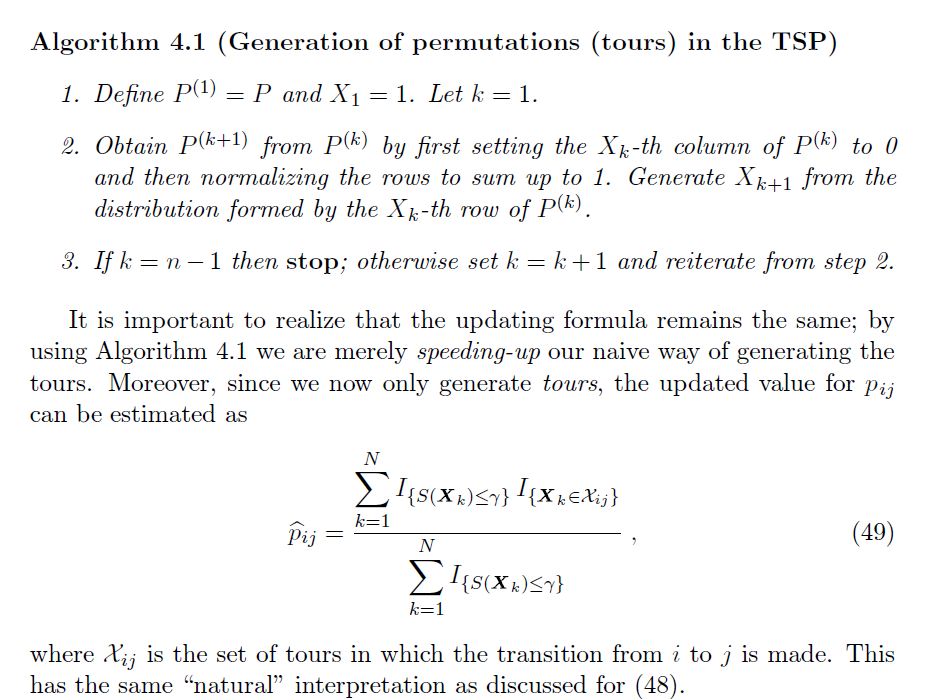
XN

k=1

IfS(􀀀 k)\_g

; (49)

where Xij is the set of tours in which the transition from i to



Question 2 quelques premières réflexions…  
  
- x : une solution possible c’est-à-dire un ensemble de n billes ordonnées choisies parmi m couleurs **(tirage sans remise)**  
- X : ensemble des solutions possibles (le nombre total de possibilités est **n parmi m**).   
- S : idem  
- v : idem   
- f : la famille de lois sur l’ensemble X :

* Dans la question 2), du projet, chaque X\_i appartenant à X est généré à chaque étape t en échantillonnant ~~indépendamment pour chaque ligne i (c’est-à-dire chaque bille)~~ **pour la première bille 1 une couleur avec la probabilité P\_t\_hat\_[1,j]. Soit j0 l’indice de la couleur tirée : pour la bille 2, on tire une couleur avec la probabilité P\_t\_hat\_[2,j]/somme sur k différent de j0 (P\_t\_hat\_[2,k]) si j ≠ j0 et 0 si j = j0**

**Rapport**

La méthode de cross-entropy (CE) est ici utilisée pour résoudre un problème complexe d’optimisation combinatoire : le jeu du mastermind.

1. **Optimisation par Cross-Entropy**
2. **Principe général**

L’objectif est de de trouver le maximum d’une fonction de performance S(x) sur tous les états x dans un ensemble X.



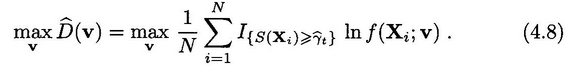
Le « problème stochastique associé » (associated stochastic problem, ASP) à notre problème, déterministe, est le suivant :   
  
pour un certain u appartenant à V, avec  une famille de densités sur l'ensemble X paramétrisées par un vecteur de réels appelé v.

L’idée de l’algorithme est de définir une suite de couples  qui converge rapidement vers 

1. **Algorithme**
2. [Initialisation] Choisir v\_0\_hat. Fixer t = 1
3. Générez un échantillon X\_1, ..., X\_N à partir de la densité  et calculer

avec 1-rho l’ordre du quantile d’ordre à choisir (de l’ordre de 0,01)

1. Utilisez le même échantillon X\_1, ..., X\_N et résoudre le programme stochastique :

V\_tilde = 

1. [Smoothing : facultatif] Appliquer l’équation suivante pour lisser le vecteur v\_t\_tilde  
   avec alpha appartient à [0,1]

**Remarque :** la raison de lisser v\_t\_tilde est double :   
(a) pour lisser les valeurs de v\_t\_hat  
(b) pour réduire la probabilité qu'une composante v\_t\_i\_hat soit nulle ou égale à un dès les premières itérations. Ceci est particulièrement important dans le cas de matrice de probabilités qui pourraient amener l’algorithme à converger vers de mauvaises solutions.

1. Si pour certains t> d (d = 5 par exemple) l’équation suivante est vérifiée, arrêtez l’algorithme. Sinon reprenez à l’étape 1.



1. **Application au mastermind**
2. **« Règles » du jeu**

L’objectif du mastermind est de deviner un « code » de billes (n billes) colorées (m couleurs numérotées de 1 à m), à la suite de plusieurs propositions. La solution cachée y et les propositions x sont donc représentées par des vecteurs ligne de longueur n avec des nombres entre 1 et m.

Après chaque nouvelle proposition, des informations sur le code sont fournies sous la forme de billes noires (billes exactement au bon endroit) et blanches (bille dans la solution, mais dans la mauvaise position)

1. **Mise en place de l’algorithme précédent**

- x : une solution possible c’est-à-dire un ensemble de n billes ordonnées choisies parmi m couleurs  
- X : ensemble des solutions possibles (le nombre total de possibilités est m^n).   
- S : On choisit la fonction de score suivante (qui aurait pu être choisie autrement) qui à une solution possible associe 2 fois le nombre de billes bien placées + 1 fois le nombre de billes mal classées

**Exemple**: Pour n=5 et m=7, Soit y = (4, 2,4, 7, 3) la solution cachée et x= (4, 3,4, 2,5) une proposition. On obtient en réponse une billes blanches (2e et 4e bille) et deux billes noires (1e et 3e billes). Par conséquent, le score pour cette supposition est de 6.

- v : ici nommé P qui correspond à une matrice de taille n\*m dont chaque élément pij (ligne i et colonne j) décrit la probabilité que de choisir la j-ème couleur pour la i-ème bille[[2]](#footnote-2).

- f : la famille de lois sur l’ensemble X :

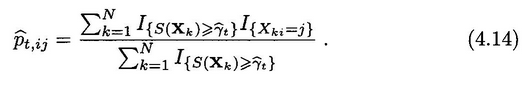
* Dans la question 1) du projet, chaque X\_i appartenant à X est généré à chaque étape t en échantillonnant indépendamment pour chaque ligne i (c’est-à-dire chaque bille) une couleur. La probabilité de tirer chaque couleur étant déterminée par P\_t\_hat\_[i,j]
* Dans la question 2),

Reprenons les différentes étapes de l’algorithme précédent.

1. [Initialisation] Choisir P\_0\_hat. Fixer t = 1
2. Générez un échantillon X\_1, ..., X\_N à partir de la loi « f » exprimée ci-dessus et calculer

 Classiquement on fixe N = C\*n\*m, où C constante (C = 5 par exemple).

1. Utilisez le même échantillon X\_1, ..., X\_N et résoudre le programme stochastique qui, dans le cas du mastermind revient à calculer la matrice P\_hat\_tilde définie ci-dessus et actualisée de la manière suivante[[3]](#footnote-3) :

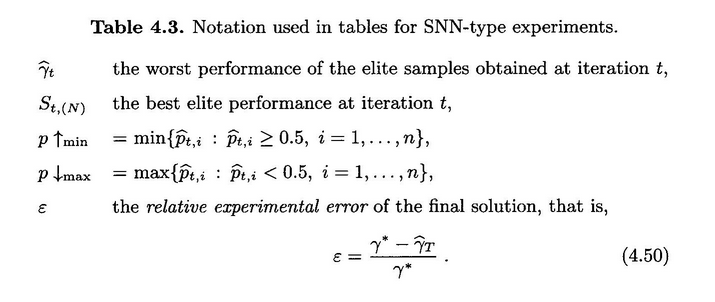
(appeler ptij tilde)

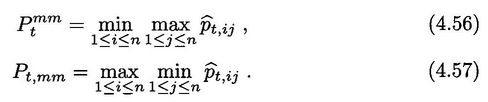
1. [Smoothing : facultatif] Appliquer l’équation suivante pour lisser la matrice P\_hat\_tilde

P\_hat <- alpha \* P\_hat\_tilde + (1-alpha)\* P\_hat\_liste[[iter-1]]

1. Critère d’arrêt (cf. partie précédente)
2. **Evaluation**

Afin d’évaluer la cconverge de l’algorithme, nous pouvons tabuler pour chaque itération t les quantités suivantes :





**Indices**

Minimiser sur x\* puis sur lambda. => Pas minimum global mais peut-être local.

X\* qui minimise : partition la plus fréquente dans les meilleures permutations. Sachant cela on peut minimiser lambda en remarquant que c’est le paramètre exponentiel.

Loi de x sachant la distance de hamming : loi de X = exponentielle de paramètre lambda. => MLE avec le lambda.

Prends plus d’itération que si on avait la bonne réponse.

C’est intéressant car plus malin que question 1 mais voir que ça prend pas plus de temps que la méthode

1. Le numérateur en (4.14) compte simplement le nombre de fois l'indice de couleur j a été attribué à la bille 1 pour les échantillons. [↑](#footnote-ref-1)
2. Puisqu'une seule couleur peut être assignée à une bille, la somme des lignes de la matrice est toujours égale à 1. [↑](#footnote-ref-2)
3. Le numérateur en (4.14) compte simplement le nombre de fois l'indice de couleur j a été attribué à la bille 1 pour les échantillons. [↑](#footnote-ref-3)