Une machine de Boltzmann pour PBIL

Arnaud Berny
ECTIA
1 rue de la Noë
BP 92119
44321 Nantes cedex 03
email : `arnaud.berny@free.fr`

Résumé

Nous proposons d'adapter une machine de Boltzmann à PBIL (population-based incremental learning). Nous remplaçons le modèle statistique de PBIL, qui fait l'hypothèse de l'indépendance statistique des variables du problème d'optimisation, par celui d'une machine de Boltzmann qui est un réseau de neurones totalement interconnectés. Nous dérivons une règle spécifique de mise à jour des poids de la machine de Boltzmann à partir du logarithme de l'espérance de la fonction à maximiser. Cette règle fait intervenir des calculs d'espérance relativement à la distribution propre à la machine de Boltzmann, mais également à sa version sélectionnée, dans l'esprit des algorithmes génétiques. Les nouvelles populations sont échantillonnées selon la distribution de la machine de Boltzmann par échantillonnage de Gibbs. L'algorithme proposé, BM-PBIL, alterne échantillonnage de Gibbs, sélection et mise à jour des poids. Nous évaluons BM-PBIL à l'aide de fonctions-tests appartenant à différentes classes, nous le comparons à PBIL et nous identifions certaines classes pour lesquelles il est supérieur à PBIL, en particulier les fonctions à saut et les fonctions quadratiques.

1 Introduction

Population-based incremental learning (PBIL) [1] est un algorithme adaptatif pour l'optimisation de fonctions numériques définies sur les vecteurs binaires de dimension donnée. On parle d'optimisation boîte noire dans la mesure où la fonction à optimiser n'est connue qu'au travers de son évaluation en des points de l'espace de recherche.

PBIL maintient un vecteur de probabilités de même dimension que celle des vecteurs binaires et dont les composantes représentent les probabilités d'obtenir zéro ou un. PBIL fait l'hypothèse de l'indépendance statistique des composantes. À chaque itération, PBIL échantillonne une population de vecteurs binaires à l'aide du vecteur de probabilités. Il en sélectionne les deux meilleurs, relativement à la fonction à maximiser ou fonction d'adaptation, et en calcule la moyenne. Enfin, il déplace le vecteur de probabilités en direction de la moyenne à l'aide d'une équation linéaire.

En dépit de ses limitations apparentes, PBIL a obtenu des résultats compétitifs sur différents problèmes, en particulier relativement aux algorithmes génétiques. Il semble cependant prometteur d'appliquer des modèles statistiques plus complexes que les simples probabilités produits tout en conservant l'idée d'un algorithme adaptatif. De tels études ont déjà été initiées.

Dans la communauté connexionniste, Williams et Peng [7] ont proposé d'utiliser un réseau de type feedforward de neurones stochastiques. D'après leurs conclusions, un algorithme avec des neurones stochastiques indépendants obtient de meilleurs résultats. Nous avançons l'hypothèse que l'architecture de type feedforward est à l'origine de cet échec. C'est ce qui a motivé notre intérêt dans les machines de Boltzmann qui sont des réseaux de neurones totalement interconnectés.

Dans la communauté évolutionniste, Zhang et Shin [10] ont proposé d'utiliser une machine de Helmholtz qui est un réseau de neurones à deux couches et bidirectionnel. Une objection à cette approche est que la dynamique d'une machine de Helmholtz ne dérive pas d'un critère explicite.

Dans ce travail, nous proposons d'associer un système dynamique à gradient maximisant un critère statistique explicite au modèle statistique d'une machine de Boltzmann. Le calcul du gradient fait naturellement apparaître l'opérateur de sélection, et la notion de corrélation entre les variables du problème découle de la structure de la machine de Boltzmann.

Le papier est organisé comme suit. Le § 2 présente rapidement le principe de l'optimisation statistique. Le § 3 introduit les modèles statistiques de PBIL et de la machine de Boltzmann. Le § 4 présente la dérivation de la règle de mise à jour du modèle de la machine de Boltzmann appliquée à l'optimisation. Le § 5 présente l'algorithme résultant, BM-PBIL, obtenu par approximation stochastique du système dynamique à gradient. Le § 6 présente les résultats obtenus avec PBIL et BM-PBIL sur un ensemble de fonctions-tests, déterministes ou aléatoires.

2 Optimisation statistique

Soit $f : E \to \mathbf{R}$ une fonction à maximiser, ou fonction d'adaptation, définie sur l'ensemble $E = \{0,1\}^n$ des vecteurs binaires de dimension n. L'optimisation statistique consiste à remplacer la maximisation de f sur E par celle d'un critère statistique défini sur l'ensemble des probabilités sur E. L'un de ces critères est l'espérance statistique de f qui est définie, pour toute probabilité μ sur E, par

$$J(\mu) = \mathbf{E}_\mu(f) = \sum_{x \in E} f(x)\mu(x)\,.$$

Nous utilisons dans la suite de ce papier les deux critères logarithmiques $\log J(\mu) = \log \mathbf{E}_\mu(f)$ et $\log \mathbf{E}_\mu(e^{\beta f})$, $\beta > 0$, de façon à faire apparaître les opérateurs de sélection proportionnelle et de Boltzmann (ne pas confondre avec la machine de Boltzmann) respectivement.

Une telle transformation du problème initial ne réduit pas la complexité de la recherche. Nous devons nous limiter à une certaine famille de probabilités, les probabilités produits dans le cas de PBIL et les machines de Boltzmann dans le cas de BM-PBIL.

3 Modèles statistiques

3.1 Probabilités produits

Les probabilités produits sont caractérisées par l'indépendance des composantes de leurs vecteurs aléatoires. On parle de modèle linéaire dans la mesure où n paramètres réels déterminent une telle probabilité. C'est le modèle utilisé par PBIL.

Une probabilité produit μ sur E est définie par

$$\mu(x) = \prod_{i=1}^{n} \phi(x_i, v_i)\,,$$

où $x = (x_i)$ est un élément de E, v_i représente la probabilité marginale d'obtenir $x_i = 1$, ce qui implique $0 \leq v_i \leq 1$, et ϕ est définie par

$$\phi(x_i, v_i) = \begin{cases} v_i & \text{if } x_i = 1, \\ 1 - v_i & \text{if } x_i = 0. \end{cases}$$

L'échantillonnage de μ se réduit à n tirages à pile ou face indépendants et biaisés. Le vecteur $v = (v_i)$ est appelé vecteur de probabilités.

3.2 Machines de Boltzmann

Pour enrichir le modèle linéaire, nous proposons d'utiliser une machine de Boltzmann qui est un réseau de neurones stochastiques totalement interconnectés dont le modèle statistique sous-jacent est une mesure de Gibbs avec une fonction d'énergie quadratique.

Soit Q une matrice symétrique $n \times n$ et m un vecteur de $\mathbf{R}^n$. Nous définissons l'opposé de la fonction d'énergie d'une machine de Boltzmann par

$$h(x) = \langle x - m \mid Q \mid x - m \rangle\,,$$

pour tout $x \in E$, où $\langle x|$ représente le vecteur transposé de x que nous écrivons $|x\rangle$ dans ce contexte.

La mesure de Gibbs associée à h est définie par

$$\pi(x) = e^{h(x)} / Z\,,$$

pour tout $x \in E$, où Z est la fonction de partition

$$Z = \sum_{x \in E} e^{h(x)}\,,$$

qui assure la normalisation des probabilités.

4 Système à gradient

Pour maximiser le critère J, nous avons recours à un système dynamique à gradient qui agit sur les paramètres de la machine de Boltzmann et qui prend la forme d'une équation différentielle

$$\mathrm{d}\,\theta / \mathrm{d}\,t = \nabla_\theta J\,, \tag{1}$$

où $\nabla_\theta J$ est le gradient de J par rapport au paramètre $\theta = m$ ou Q. Dans le cas de m, le gradient de J est le vecteur des dérivées partielles $(\partial J/\partial m_i)$. Dans le cas de Q, le gradient de J est la matrice des dérivées partielles $(\partial J/\partial q_{ij})$.

L'observation essentielle est que le gradient du critère peut s'écrire sous la forme d'une espérance, ce qui suggère un algorithme stochastique. Plus précisément,

$$\begin{aligned}\nabla J &= \sum_{x\in E} f(x) \times \nabla \pi(x)\\ &= \sum_{x\in E} f(x) \times \nabla \log \pi(x) \times \pi(x)\\ &= \mathbf{E}_\pi(f \times \nabla \log \pi)\,.\end{aligned}$$

Nous avons

$$\log \pi(x) = h(x) - \log Z\,.$$

Le gradient du logarithme de la fonction de partition s'écrit

$$\begin{aligned}\nabla \log Z &= \nabla Z \big/ Z\\ &= \sum_{x\in E} \nabla h(x) \times e^{h(x)} \big/ Z\\ &= \mathbf{E}_\pi(\nabla h)\,.\end{aligned}$$

Donc

$$\nabla \log \pi(x) = \nabla h - \mathbf{E}_\pi(\nabla h)\,,$$

et le gradient du critère s'écrit

$$\nabla J = \mathbf{E}_\pi(f \times \nabla h) - \mathbf{E}_\pi(\nabla h) \times \mathbf{E}_\pi(f)\,.$$

Nous nous intéressons au gradient du *logarithme* du critère J de façon à rendre explicite l'opérateur de sélection proportionnelle des algorithmes génétiques :

$$\begin{aligned}\nabla \log J &= \mathbf{E}_\pi\left(\nabla h \times \frac{f}{\mathbf{E}_\pi(f)}\right) - \mathbf{E}_\pi(\nabla h)\\ &= \mathbf{E}_\sigma(\nabla h) - \mathbf{E}_\pi(\nabla h)\,,\end{aligned}$$

où σ est la probabilité qui résulte de la sélection de π :

$$\pi(x) \xrightarrow{\text{selection}} \sigma(x) = \frac{f(x) \times \pi(x)}{\mathbf{E}_\pi(f)}$$

En utilisant le même raisonnement, le gradient du critère $\log \mathbf{E}_\pi(e^{\beta f})$ s'obtient en remplaçant la sélection proportionnelle par la sélection de Boltzmann :

$$\pi(x) \xrightarrow{\text{selection}} \sigma(x) = \frac{e^{\beta f(x)} \times \pi(x)}{\mathbf{E}_\pi(e^{\beta f})}$$

Tout ce qui précède est indépendant de l'expression de la fonction h. Pour préciser le gradient du logarithme de J, il reste à calculer le gradient de h par rapport aux paramètres m et Q :

$$\begin{aligned} \nabla_m h(x) &= 2Q(m-x) \\ \nabla_Q h(x) &= |x-m\rangle\langle x-m|\,, \end{aligned}$$

l'élément général d'une matrice symétrique $|x\rangle\langle x|$ étant égal au produit $x_i x_j$. Le gradient du logarithme de J par rapport à chacun des paramètres s'écrit finalement

$$\nabla_m J = 2Q\big(\mathbf{E}_\pi(x) - \mathbf{E}_\sigma(x)\big) \tag{2}$$

$$\nabla_Q J = \mathbf{E}_\sigma\big(|x-m\rangle\langle x-m|\big) - \mathbf{E}_\pi\big(|x-m\rangle\langle x-m|\big)\,. \tag{3}$$

L'équation (2), en particulier la différence entre l'espérance des vecteurs selon la probabilité de recherche π et l'espérance des vecteurs selon sa version sélectionnée σ, rappelle l'algorithme PBIL, la principale différence se situant dans la prise en compte de la matrice Q. L'équation (3) fait apparaître la différence de deux pseudo-matrices de corrélation, relativement à σ et π.

Les conditions initiales du système (1) sont

$$\begin{aligned} m(0) &= (1/2, \ldots, 1/2) \\ Q(0) &= 0\,. \end{aligned}$$

5 Algorithmes stochastiques

5.1 Échantillonnage de Gibbs

Les gradients que nous avons calculés nécessitent le calcul d'espérances relativement à la probabilité de recherche, qui est la mesure de Gibbs, ou à sa version sélectionnée. Ces espérances sont approchées par des moyennes sur des populations finies de vecteurs échantillonnés selon la mesure de Gibbs ou sa version sélectionnée. L'échantillonnage de Gibbs est une technique classique pour générer des vecteurs aléatoires dont la loi approche la mesure de Gibbs.

Au contraire de l'apprentissage supervisé dans les machines de Boltzmann, nous ne pouvons pas nous exonérer de l'échantillonnage de Gibbs (par exemple en utilisant une technique de champ moyen) qui permet de produire des points de l'espace de recherche, donc de l'information concernant la fonction d'adaptation au travers de leur évaluation.

Nous rappelons brièvement le principe de l'échantillonnage de Gibbs. À chaque itération, une composante i de l'état courant x est choisie de manière uniforme et sa probabilité *conditionnelle* $p = \pi(x_i = 1 \mid x_j, j \neq i)$ est calculée. La valeur suivante de x_i est alors tirée à pile ou face avec une probabilité p, et une autre composante est considérée.

D'après l'expression de la mesure de Gibbs, nous avons $p = 1/1 + e^{-\Delta}$, où $\Delta = h(x : x_i = 1) - h(x : x_i = 0)$, c.-à-d. la variation de h quand la valeur de la composante i passe de 0 à 1, les autres composantes restant inchangées. D'après l'expression de h, nous obtenons

$$\Delta = (1 - 2m_i)q_{ii} + 2\sum_{j\neq i}(x_j - m_j)q_{ij}\,,$$

qui est calculable en temps linéaire en n. On peut voir l'échantillonnage de Gibbs dans une machine de Boltzmann comme un opérateur de mutation dirigée.

La chaîne de Markov associée à l'échantillonnage de Gibbs possède une unique probabilité invariante qui est la mesure de Gibbs. La longueur de la chaîne conditionne la qualité de l'approximation de la mesure de Gibbs. Deux échantillons consécutifs (leur distance de Hamming est au plus égale à 1) sont très fortement corrélés et le second n'apporte que peu d'information sur la fonction d'adaptation. Nous proposons donc de simuler la chaîne de Markov pendant T itérations entre avant de prélever un nouvel individu. La distance de Hamming entre deux individus consécutifs est donc bornée par T (et n). La complexité de l'échantillonnage d'une population de N individus est proportionnelle à nTN. L'état initial x_0 est choisi uniformément et la chaîne de Markov n'est pas réinitialisée après l'échantillonnage d'une population.

5.2 Règle de mise à jour pour BM-PBIL

En réunissant l'échantillonnage de Gibbs, l'expression des gradients et l'opérateur de sélection, nous obtenons l'algorithme BM-PBIL (pour Boltzmann Machine PBIL) :

1. échantillonner une population (x^i), $1 \leq i \leq N$, de vecteur binaires en utilisant l'échantillonnage de Gibbs ;
2. lui appliquer un opérateur de sélection pour produire une nouvelle population (y^i), $1 \leq i \leq P$. Dans le cas de la sélection de Bolzmann, la probabilité pour que l'individu x^i soit sélectionné est égale à
$$e^{\beta f(x^i)} \Big/ \sum_{j=1}^{N} e^{\beta f(x^j)} ,$$
où $\beta > 0$ contrôle la pression de sélection ;
3. calculer les moyennes :
$$m_\pi = 1/N \sum_{i=1}^{N} x^i \quad m_\sigma = 1/P \sum_{i=1}^{P} y^i$$
$$Q_\pi = 1/N \sum_{i=1}^{N} |x^i - m\rangle\langle x^i - m|$$
$$Q_\sigma = 1/P \sum_{i=1}^{P} |y^i - m\rangle\langle y^i - m| \, ;$$
4. mettre à jour les paramètres avec
$$\begin{aligned} m_{t+1} &= m_t + \alpha \times Q_t(m_\pi - m_\sigma) \\ Q_{t+1} &= Q_t + \alpha \times (Q_\sigma - Q_\pi) \, , \end{aligned}$$
où $\alpha > 0$ est le gain d'apprentissage. La complexité de la mise à jour est proportionnelle à n^2.

5.3 Règle de mise à jour pour PBIL

À titre de comparaison, nous rappelons la règle de mise à jour de PBIL. Le modèle statistique a été décrit au § 3.1. PBIL maintient un vecteur de probabilités v qui est directement utilisé pour échantillonner une population de N individus à chaque itération. Le vecteur de probabilités est mis à jour avec l'équation

$$v_{t+1} = v_t + \alpha \times (m_\sigma - v_t)\,, \tag{4}$$

où m_σ est égal à la moyenne des individus sélectionnés. Rappelons que l'on peut également écrire $v = \mathbf{E}_\mu(x)$, où μ est la probabilité produit de vecteur v, ce qui rapproche les équations (2) et (4). En fait, PBIL peut également être vu comme l'approximation stochastique d'un système dynamique à gradient [3].

6 Simulations

D'après le théorème *No Free Lunch* [8], les performances moyennes de deux algorithmes de recherche quelconques sur l'ensemble de toutes les fonctions d'adaptation (à valeurs dans un ensemble fini) sont identiques. Cependant, la valeur pratique de ce théorème est contestée. En particulier, il a été démontré que le théorème NFL ne s'applique pas sur des ensembles de fonctions satisfaisants à certaines contraintes générales [4]. Dans ces conditions, il semble légitime de chercher à comparer ou améliorer les algorithmes. La non conservation des performances conduit également à la spécialisation des algorithmes.

6.1 Fonctions tests

Nous avons appliqué BM-PBIL et PBIL à six classes différentes de fonctions-tests, soit déterministes soit aléatoires :

1. *"Big Jump"* est la fonction "One Max" (comptage de uns) dont l'optimum a été isolé. Soit x un vecteur binaire de dimension n et m le nombre de uns qui le composent. La valeur en x de la fonction "Big Jump" est définie par

$$\begin{cases} n, & \text{si } m = n, \\ 0, & \text{si } n - t < m < n, \\ m, & \text{sinon}, \end{cases}$$

   où t contrôle l'isolement de l'optimum global. Nous avons considéré les cas $t = 0$ ("One Max"), 5 et 10.

2. Le problème des quatre extrema ou *Four Peaks* a été introduit par Baluja et Caruana [1]. Sa fonction d'adaptation possède quatre optima, dont deux locaux et deux globaux. Comme pour "Big Jump", un seuil contrôle l'isolement de l'optimum global. La valeur en x de la fonction d'adaptation est définie par

$$\max(u(x), z(x)) + r(x)\,,$$

   où $u(x)$ est égal au nombre de uns en tête de vecteur, $z(x)$ au nombre de zéros en queue de vecteur, et $r(x)$ est tel que

$$r(x) = \begin{cases} n, & \text{si } u(x) > t \text{ et } z(x) > t \\ 0, & \text{sinon.} \end{cases}$$

Nous avons considéré les cas $t = 10$, 15 et 20.

3. Le problème de l'annulation de sommes ou *Summation Cancellation* [2] consiste à annuler

$$\sum_{i=1}^{n/9} |b_i| ,$$

où $b_i = b_{i-1} + a_i$, $b_1 = a_1$, et chaque a_i est un nombre flottant représenté à l'aide de 9 bits compris entre -2.56 et 2.55. Chaque vecteur binaire de dimension n est tout d'abord converti en $n/9$ flottants a_i puis évalué. La fonction d'adaptation est définie comme l'opposé de la somme. Nous avons également considéré une variante de ce problème décrite par Schoenauer et Sebag [6] où $b_i = \sin b_{i-1} + a_i$.

4. Le problème des produits égaux ou *Equal Products* [2] consiste à trouver une partition, représentée par un vecteur binaire x, d'un ensemble de n réels (a_i) telle que

$$\prod_{i=1}^{n} a_i^{x_i} = \prod_{i=1}^{n} a_i^{1-x_i} .$$

Si aucune partition ne convient, le problème consiste à minimiser la distance entre les deux produits ou maximiser son opposé. Nous avons généré cinq instances de ce problème en choisissant indépendamment et uniformément sur l'intervalle $[0, 4]$ les (a_i).

5. Les modèles NK ou *NK landscapes* [5] décrivent les interactions complexes entre les gènes d'un individus et son environnement. Les deux paramètres structurels d'un modèle NK sont la dimension n des vecteurs binaires et l'ordre k des interactions. Chaque fonction d'adaptation est la somme de n fonctions partielles qui ne dépendent chacune que de $k+1$ composantes. L'optimisation d'un modèle NK est un problème NP-complet [9]. Pour construire des instances de modèles NK, nous avons échantillonné, pour chaque composante, un ensemble de k composantes voisines, et pour chacune de leurs 2^{k+1} combinaisons, nous avons échantillonné, selon la loi normale, la valeur de la fonction partielle correspondante. Nous avons généré cinq instances pour $k = 2$ et cinq autres pour $k = 4$.

6. Enfin, nous avons considéré le problème de la programmation quadratique binaire, qui est équivalent à la recherche d'un état d'énergie minimale d'un système de spins dans le contexte de la physique statistique (modèle de Sherrington-Kirkpatrick). Il s'agit également d'un problème NP-complet. Les fonctions d'adaptation pour ce problème sont équivalentes aux fonctions d'énergie des machines de Bolzmann et sont de la forme

$$\sum_{i<j} (2x_i - 1)(2x_j - 1)a_{ij} + \sum_i (2x_i - 1)b_i .$$

Nous avons généré cinq fonctions quadratiques en choisissant les coefficients a_{ij} et b_i indépendamment et uniformément suivant la loi normale.

6.2 Paramètres

La dimension des vecteurs binaires est $n = 80$, sauf pour les problèmes d'annulation de sommes pour lesquels la dimension est $n = 90$. Les deux algorithmes BM-PBIL et PBIL ont été simulés pendant 10000 itérations. La taille de la population échantillonnée selon la probabilité de recherche est $N = 50$. Le nombre d'évaluations de la fonction d'adaptation est donc égal à 500000. Pour donner un ordre de grandeur, la fraction visitée de l'espace de recherche est bornée par 10^{-18}. Dans le cas de BM-PBIL, la longueur de la chaîne de Markov entre deux individus consécutifs est $T = 20$. À chaque itération, seul le meilleur individu est sélectionné pour la mise à jour des paramètres, ce qui revient à utiliser la sélection de Bolzmann avec $\beta = \infty$. Le gain d'apprentissage contrôle le compromis entre qualité des solutions découvertes et vitesse de convergence. BM-PBIL et PBIL possèdent des dynamiques différentes et n'obtiennent pas les meilleurs résultats pour les mêmes valeurs de gain. Nous avons considéré les valeurs $\alpha = 0.001$ et $\alpha = 0.005$ pour chacun, mais $\alpha = 0.005$ est apparu trop grand pour BM-PBIL.

6.3 Résultats

Les résultats sont visibles sur les Tab. 1–3. Pour chaque algorithme, nous avons calculé la moyenne sur 30 exécutions de la meilleure adaptation découverte ainsi que l'itération moyenne à laquelle elle a été atteinte pour la première fois. Nous avons également calculé les écarts-types associés. Pour chaque fonction-test, nous avons représenté en gras la meilleure adaptation.

BM-PBIL domine nettement PBIL sur les fonctions à saut, “Big Jump” et “Four Peaks”, ainsi que sur les fonctions spin. Ce dernier résultat n'est pas une surprise dans la mesure où BM-PBIL est capable de prendre en compte certaines interactions binaires entre les variables du problème. BM-PBIL domine légèrement PBIL sur le problème des produits égaux.

BM-PBIL est par contre dominé sur les problèmes d'annulation de sommes ainsi que sur les modèles NK, d'ordre 2 ou 4. On peut observer que les modèles NK d'ordre 2 ne sont pas représentables par des fonctions quadratiques. Ni BM-PBIL, ni PBIL ne sont capables de capturer la structure des interactions dans ces modèles. L'amélioration du modèle statistique ne s'est pas traduite par des performances accrues. Selon nous, la raison de cet échec tient plus au caractère aléatoire des interactions d'un modèle NK qu'à leur structure. Pour des fonctions d'adaptation fortement aléatoires, PBIL semble plus adapté que BM-PBIL.

La Tab. 4 montre les temps de calcul[1] pour les deux algorithmes sur deux fonctions, `fp`, qui est calculable en temps linéaire en n, et `spin`, qui est calculable en temps quadratique en n. En dépit de sa complexité, BM-PBIL est intéressant dans le cas d'une fonction d'adaptation dont la complexité est quadratique. Il est néanmoins nécessaire de limiter la longueur totale TN de la chaîne de Markov.

7 Conclusion

Nous avons proposé l'extension de PBIL à l'aide de machines de Bolzmann. Nous avons dérivé, à partir d'un critère statistique logarithmique, des règles de mise à jour

[1] Les algorithmes ont été implémentés en C++ et exécutés sur un Athlon XP 1800.

TAB. 1 – Résultats pour BM-PBIL ($\alpha = 0.001$).

	BM-PBIL			
	Max		Iteration	
Function	Mean	Dev.	Mean	Dev.
`bj-10`	**80.00**	0	1215	139
`bj-15`	**68.00**	6.0	922	268
`bj-20`	**60.00**	0	636	150
`cancel`	-1.053e-01	5.7e-02	9319	521
`cancel-sin`	-7.668e-02	3.6e-02	7902	614
`ep-0`	**-6.17**	6.4	4601	3194
`ep-1`	**-434.35**	448.1	5656	2761
`ep-2`	-20.93	19.7	5747	3291
`ep-3`	-11.16	12.8	5889	2554
`ep-4`	**-5.557e-01**	4.7e-01	6497	2570
`fp-10`	**149.00**	0	5216	1044
`fp-15`	**144.00**	0	4269	421
`fp-20`	**137.97**	4.1	3905	322
`nk-2-0`	**64.89**	1.1e-01	2725	461
`nk-2-1`	62.67	5.0e-01	2578	283
`nk-2-2`	61.44	2.9e-01	2530	277
`nk-2-3`	66.17	5.0e-01	3551	1004
`nk-2-4`	68.09	2.4e-01	2924	355
`nk-4-0`	69.39	2.3	5054	943
`nk-4-1`	76.02	1.4	4707	792
`nk-4-2`	72.79	3.2	5071	1003
`nk-4-3`	70.79	1.8	4668	868
`nk-4-4`	72.89	2.7	5242	1424
`spin-0`	**489.65**	13.2	4513	376
`spin-1`	**496.29**	12.9	4892	369
`spin-2`	**527.58**	14.5	4459	428
`spin-3`	**497.72**	9.1	4529	439
`spin-4`	**512.36**	8.5	4574	499

TAB. 2 – Résultats pour PBIL ($\alpha = 0.001$).

	PBIL			
	Max		Iteration	
Function	Mean	Dev.	Mean	Dev.
`bj-10`	79.00	3.0	5580	1823
`bj-15`	65.00	0	1096	104
`bj-20`	**60.00**	0	460	125
`cancel`	-1.12	3.1e-01	9563	566
`cancel-sin`	-4.217e-01	1.7e-01	9787	195
`ep-0`	-7.35	7.8	4671	2756
`ep-1`	-534.67	542.1	5240	2895
`ep-2`	-22.49	22.1	4957	2783
`ep-3`	-18.47	17.3	4750	2922
`ep-4`	-7.790e-01	7.1e-01	4984	2642
`fp-10`	144.30	2.3	9595	423
`fp-15`	141.47	2.3	9721	249
`fp-20`	129.00	7.2	8784	1197
`nk-2-0`	64.86	6.6e-02	6770	356
`nk-2-1`	63.08	3.4e-01	6736	688
`nk-2-2`	61.55	1.8e-01	7879	942
`nk-2-3`	**66.21**	4.9e-01	8093	1171
`nk-2-4`	**68.33**	6.2e-02	7415	660
`nk-4-0`	**72.06**	1.2	9565	246
`nk-4-1`	76.28	9.3e-01	8980	706
`nk-4-2`	75.06	2.5	9422	552
`nk-4-3`	71.15	1.5	9531	301
`nk-4-4`	**76.41**	2.0	9576	357
`spin-0`	481.90	6.9	8964	671
`spin-1`	475.87	10.4	9449	507
`spin-2`	519.29	7.8	8933	725
`spin-3`	493.84	5.0	9134	799
`spin-4`	490.36	14.6	9351	606

TAB. 3 – Résultats pour PBIL ($\alpha = 0.005$).

	PBIL			
	Max		Iteration	
Function	Mean	Dev.	Mean	Dev.
`bj-10`	79.00	3.0	2033	1293
`bj-15`	65.00	0	259	26
`bj-20`	**60.00**	0	126	32
`cancel`	**-4.867e-02**	7.6e-03	3372	215
`cancel-sin`	**-4.833e-02**	7.8e-03	2929	181
`ep-0`	-7.31	8.6	5184	2874
`ep-1`	-539.61	465.9	5284	2994
`ep-2`	**-15.03**	14.1	5445	2516
`ep-3`	**-10.55**	11.5	5017	2556
`ep-4`	-7.432e-01	7.4e-01	4678	2921
`fp-10`	**149.00**	0	2194	92
`fp-15`	137.30	12.7	2020	226
`fp-20`	89.90	18.6	2256	417
`nk-2-0`	64.89	9.4e-02	1460	128
`nk-2-1`	**63.08**	3.6e-01	1495	251
`nk-2-2`	**61.59**	3.3e-01	1590	165
`nk-2-3`	66.10	5.1e-01	1718	230
`nk-2-4`	68.28	9.8e-02	1544	111
`nk-4-0`	71.85	1.8	2208	311
`nk-4-1`	**77.07**	1.4	1952	244
`nk-4-2`	**75.24**	3.2	2130	291
`nk-4-3`	**73.11**	2.1	2125	270
`nk-4-4`	76.39	1.9	2061	366
`spin-0`	476.29	16.2	2036	325
`spin-1`	479.95	14.2	2165	399
`spin-2`	519.61	10.6	1957	364
`spin-3`	484.23	13.6	2057	358
`spin-4`	491.93	21.3	2169	450

TAB. 4 – Comparaison des temps de calcul (en secondes).

Fonction	PBIL	BM-PBIL
`fp`	3.6	37.4
`spin`	17.6	51.7

spécifiques qui font appel à un opérateur de sélection. Leur complexité peut être maintenue à un niveau acceptable, c.-à-d. quadratique pour la dimension des vecteurs binaires.

Nous avons comparé l'algorithme résultant BM-PBIL à PBIL sur différentes classes de fonctions-tests. BM-PBIL domine PBIL sur les fonctions à saut, les fonctions quadratiques et le problème des produits égaux. Par contre, il est dominé sur les modèles NK et le problème d'annulation de sommes.

Ces résultats prometteurs devraient encourager le développement de nouveaux algorithmes adaptatifs d'optimisation de type boîte noire qui prennent en compte les interactions entre les variables d'un problème, en parallèle des méthodes d'estimation de densité par réseaux bayésiens.

Références

[1] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Proceedings of the 12th Annual Conference on Machine Learning*, pages 38–46, 1995.

[2] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization : learning the structure of the search space. Technical Report CMU-CS-97-107, Carnegie-Mellon University, January 1997.

[3] A. Berny. Selection and reinforcement learning for combinatorial optimization. In M. Schoenauer et al., editors, *Parallel Problem Solving from Nature VI*, Lecture Notes in Computer Science, pages 601–610, Paris, September 2000. Springer-Verlag.

[4] C. Igel and M. Toussaint. On classes of functions for which No Free Lunch results hold. arXiv : cs.NE/0108011, August 2001.

[5] S. A. Kauffman. *The origins of order : self-organisation and selection in evolution*. Oxford University Press, 1993.

[6] M. Sebag and M. Schoenauer. A society of hill-climbers. In *Proc. IEEE Int. Conf. on Evolutionary Computation*, pages 319–324, Indianapolis, April 1997.

[7] R. J. Williams and J. Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3) :241–268, 1991.

[8] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation*, 1(1) :67–82, April 1997.

[9] A. H Wright, R. K. Thompson, and J. Zhang. The computational complexity of N-K fitness functions. *IEEE Trans. on Evolutionary Computation*, 4(4) :373–379, November 2000.

[10] B.-T. Zhang and S.-Y. Shin. Bayesian evolutionary optimization using Helmholtz machines. In M. Schoenauer et al., editors, *Parallel Problem Solving from Nature VI*, Lecture Notes in Computer Science, pages 827–836, Paris, September 2000. Springer-Verlag.