Statistical Machine Learning and Combinatorial Optimization

A. Berny

Abstract

In this work we apply statistical learning methods in the context of combinatorial optimization, which is understood as finding a binary string minimizing a given cost function. We first consider probability densities over binary strings and we define two different statistical criteria. Then we recast the initial problem as the problem of finding a density minimizing one of the two criteria. We restrict ourselves to densities described by a small number of parameters and solve the new problem by means of gradient techniques. This results in stochastic algorithms which iteratively update density parameters. We apply these algorithms to two families of densities, the Bernoulli model and the Gaussian model. The algorithms have been implemented and some experiments are reported.

1 Introduction

In this work, we apply statistical learning methods in the context of combinatorial optimization, which is understood as finding a binary string minimizing a given cost function. We transform the initial search problem over binary strings into a search problem over binary strings probability distributions. The solutions of the latter are mixtures of Dirac distributions charging optimal solutions of the former. However, the search for a probability distribution rather than for a string do not reduce the complexity. In order to make this model tractable, we decide to restrict ourselves to distributions completely determined by a small – compared to the size of the binary strings space – number of parameters. In this paper we consider two families of probability distributions, the Bernoulli model and the Gaussian model.

The search criterion in the space of binary strings is the cost function of the problem itself. We define two additional criteria for the search over distributions. The first one is simply the expectation of the cost function relatively to a distribution. The second one is the Kullback-Leibler divergence between a distribution and the Gibbs distribution at a given temperature. Finally we replace the search for a minimum over the set of binary strings, which is discrete and finite, by the search for a minimum of one of the two criteria, which are differentiable functions relatively to distribution parameters.

In order to minimize one of the two criteria, we use gradient dynamical systems acting on distribution parameters. There are some well-known limitations to this approach, namely the fact that it is subject to local minima. But we do not simulate this dynamical system. We rather turn to a stochastic approximation of it, enabling the system to escape local minima to some extent. We set the initial conditions of the algorithm such that, at first, it behaves like a random search. While relaxing, it is biased towards binary strings with lower cost. We are aware of the fact that this approach is likely to be defeated by deceptive functions (such as those exhibiting Hamming-isolated optimal solutions). It is intended to demonstrate the link between statistical learning procedures and combinatorial optimization.

The paper is organized as follows. Section 2 is a derivation of the first algorithm from the principles of statistical physics, the Gibbs distribution and the Kullback-Leibler divergence. In section 3 we study the second algorithm which relies on the concept of reinforcement learning. In section 4 we consider the choice of Bernoulli random variables and in section 5 the choice of the Gaussian law to take into account second-order statistics. Section 6 is a summary of the main equations, including dynamics and distributions. Finally, in sections 7 and 8, we apply the above ideas to two combinatorial optimization problems, the four peaks problem and the iterated prisoner's dilemma problem, and report some experiments and results.

2 Free Energy Minimization

This section first introduces some concepts drawn from statistical physics, in particular the Gibbs distribution which models a system in thermodynamical equilibrium at a given temperature. Such a statistical framework has been introduced into many fields. One can cite combinatorial optimization [9], machine learning [14, 10] or image processing [6]. However, relaxation methods (without memory) have prevailed in the field of combinatorial optimization. In the remainder of the paper we focus on the application of learning or adaptive algorithms to combinatorial optimization. We use the Gibbs distribution as a reference distribution and try to approximate or learn it with a target distribution. This is motivated by the fact that the Gibbs distribution charges low cost states at low temperatures. The comparison between both distributions relies on the Kullback-Leibler divergence which gives a statistical criterion to be minimized. The process of minimizing the criterion is then achieved by means of a gradient dynamical system which is finally approximated by a stochastic discrete time dynamical system.

2.1 Statistical framework

We first give some mathematical notations. Let X be a finite set of n binary variables X_i taking their values in the hypercube $\mathcal{S} = \{0,1\}^n$. Let x_i be the value of variable X_i. The problem consists in minimizing a function $H : \mathcal{S} \to \mathbb{R}$

called the cost function of the problem.

In order to plunge an optimization problem into a statistical framework we use the Gibbs distribution [6]. The Gibbs distribution maps each value of the cost function onto a probability defined by

$$p_T^*(x) = \frac{1}{Z} \times \exp\left(-\frac{H(x)}{T}\right),$$

where $T > 0$ is the analogue of a temperature and $Z = \sum_{y \in \mathcal{S}} \exp(-H(y)/T)$ is a function which only depends on T. Z is sometimes called the *partition function*. The Gibbs distribution p_T^* has the fundamental property that when $T \to 0$ it converges to a limit distribution p^* which uniformly charges optimal solutions. Put another way, if $U = \{x \in \mathcal{S} / \forall y \in \mathcal{S}, H(x) \leq H(y)\}$ is the set of all optimal solutions, then

$$p^*(x) = \begin{cases} 1/|U| & \text{if } x \in U, \\ 0 & \text{otherwise,} \end{cases}$$

where $|U|$ is the cardinal of U (see [6]).

2.2 Optimization as learning

The theoretical distribution p_T^* is an implicit information source. The Metropolis algorithm allows to sample from p_T^* by building a Markov chain whose stationary distribution is precisely p_T^* (see [8]). We adopt a different point of view in this work. It is clear from the statistical framework (see page 2) that if we approximate or learn the theoretical distribution p_T^* with a target distribution p then low cost states will be more likely than high cost states. In that sense optimization can be seen as a learning task.

We need some sort of distance between the Gibbs distribution and the target distribution to evaluate the quality of approximation. For this purpose, we use the Kullback-Leibler (KL) divergence [10] which is well suited to exponential functions. The KL divergence between p and p_T^* is defined by

$$\mathcal{D}(p, p_T^*) = -\sum_{x \in \mathcal{S}} p(x) \times \log \frac{p_T^*(x)}{p(x)}.$$

It has, in particular, the following properties:

1. $\mathcal{D}(p, p_T^*) \geq 0$;
2. $\mathcal{D}(p, p_T^*) = 0$ if and only if $p = p_T^*$.

Replacing p_T^* by its expression, we find

$$\mathcal{D}(p, p_T^*) = \sum_{x \in \mathcal{S}} p(x) \times \left(\log Z + \frac{H(x)}{T} + \log p(x)\right).$$

Since Z is a function which only depends on the temperature T and $\sum_{x\in\mathcal{S}} p(x) = 1$, we simplify the expression:

$$\begin{aligned}\mathcal{D}(p, p_T^*) &= \log Z + \frac{1}{T}\left(\sum_{x\in\mathcal{S}} p(x)H(x) + T \times \sum_{x\in\mathcal{S}} p(x)\log p(x)\right)\\ &= \log Z + \frac{1}{T}(\mathbf{E} - T\mathbf{S})\end{aligned}$$

where we have introduced some thermodynamical quantities:

- $\mathbf{E} = \sum_{x\in\mathcal{S}} p(x)H(x)$ is the *internal energy* of the system;
- $\mathbf{S} = -\sum_{x\in\mathcal{S}} p(x)\log p(x)$ is its *entropy*;

The quantity $\mathbf{F} = \mathbf{E} - T\mathbf{S}$ is called the *free energy* of the system. When operating at constant temperature, minimizing the KL divergence, i.e., the learning task, reduces to minimizing the free energy of the system.

2.3 Dynamics

Recall that we want to minimize the free energy $\mathbf{F}$ of the system. The corresponding search space is the set of probability distributions over $\mathcal{S}$. However, $2^n - 1$ positive numbers are needed to describe a probability distribution. So we restrict the search to probability distributions described by a reduced number ν of parameters $\theta = (\theta_1, \ldots, \theta_\nu)^t$, where $\cdot^t$ denotes transpose. We write $p(\cdot, \theta)$ such a distribution. We also require $p(x, \cdot)$ to be differentiable with respect to its parameters for each $x \in \mathcal{S}$ so that the free energy $\mathbf{F}$ is itself differentiable, although the initial space $\mathcal{S}$ is discrete. The problem may now be restated as finding a vector $\theta^* \in \mathbb{R}^\nu$ such that $\mathbf{F}(\theta^*) = \min_{\mathbb{R}^\nu} \mathbf{F}$. In order to yield a solution, we apply gradient techniques.

We start with a dynamical system defined by the ordinary differential equation

$$\frac{d\theta}{dt} + \frac{\partial \mathbf{F}}{\partial \theta} = 0,$$

where $\partial\mathbf{F}/\partial\theta$ is the gradient vector $(\partial\mathbf{F}/\partial\theta_1, \ldots, \partial\mathbf{F}/\partial\theta_n)^t$. We exchange the sum over the state space and the gradient:

$$\frac{d\theta}{dt} + \sum_{x\in\mathcal{S}}\left(H(x)\frac{\partial p}{\partial\theta}(x) + T \times \frac{\partial(p\log p)}{\partial\theta}(x)\right) = 0$$

We finally obtain:

$$\frac{d\theta}{dt} + \sum_{x\in\mathcal{S}}\left(H(x) + T(1 + \log p(x))\right)\frac{\partial p}{\partial\theta}(x) = 0$$

We write $\partial p/\partial\theta = p \times \partial \log p/\partial\theta$ and we apply the stochastic approximation technique [5], which means that we replace the ordinary differential equation by a stochastic discrete time dynamical system: we just leave the weighted sum over the state space and only retain the current sample. An intermediary solution is to compute a partial sum before updating parameters. The basic update rule is then

$$\Delta\theta = -\alpha\big(H + T(1 + \log p)\big)\frac{\partial \log p}{\partial \theta}, \tag{1}$$

where $\alpha > 0$ is a small constant called the learning rate. This update rule is similar to the one derived by Sabes and Jordan in the field of unsupervised associative learning [14].

2.4 Algorithm

Assuming we know how to randomly generate solutions in $\mathcal{S}$ following a given probability distribution $p(\cdot, \theta)$, we are now ready to write the algorithm:

step 1 initialize θ;

step 2 generate $x \in \mathcal{S}$ from $p(\cdot, \theta)$;

step 3 evaluate $H(x)$;

step 4 update θ with Eq. 1;

step 5 go to step 2.

The initial parameter θ_0 should be chosen such that $p(\cdot, \theta_0)$ is uniform over $\mathcal{S}$ in order to ensure sufficient exploration before convergence. The algorithm runs until a global optimum is reached, a time bound is encountered or the distribution is concentrated enough so that one considers that the system has converged. Both the temperature T and the learning rate α are chosen empirically.

3 Reinforcement Learning

Reinforcement learning is the second framework for mixing learning and optimization that we study. It also leads to a dynamical system and an update rule similar to the one derived in the previous section. Basic reinforcement learning involves an agent faced with a learning task and an environment. Learning is unsupervised in the sense that the agent is not taught with examples but rather makes trials evaluated by the environment which sends back to the agent a reinforcement signal. The agent modifies its behavior in order to maximize or minimize the reinforcement signal which is a scalar information. In the context of combinatorial optimization, the agent learns to produce low cost binary strings and the reinforcement signal is simply the cost function. We first review some related work on reinforcement learning and then derive an update rule from its principles. Finally we give the corresponding algorithm.

3.1 Related work

Williams has reviewed many reinforcement learning schemes and has abstracted them into a generic class called REINFORCE [15]. He has focused his work on tasks which involve learning with associative networks. We can also cite the work of Barto, Sutton and Anderson in the context of learning control [3].

Let us recall the REINFORCE algorithm:

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})\frac{\partial \log g_i}{\partial w_{ij}},$$

where r is the reinforcement signal to be maximized in this context, w_{ij} is the weight of the connection from input j to unit i, g_i is the probability of the output of unit i given the input and the weights to this unit. b_{ij} is called the baseline: it should be independent of the output of unit i. α_{ij} is the learning rate: it should depend at most on the weights w_{ij} and time. The partial derivative of the logarithm of g_i is called the eligibility of weight w_{ij}. We have already seen such a log-derivative in Eq. 1 while minimizing the free energy of the system. Williams has proved that REINFORCE algorithms are in fact gradient-following algorithms.

3.2 Reinforcement learning for combinatorial optimization

In Williams' model, there are three sources of randomness:

1. the environment's choice of input to the network;
2. the network's choice of output;
3. the environment's choice of reinforcement value.

In the context of combinatorial optimization, we only keep the second source of randomness: there is no input and the reinforcement signal (the cost function) is determinist. We will only consider stationary reinforcement signals.

We propose to modify the REINFORCE algorithm to fit our task:

$$\Delta\theta_i = -\alpha\,(H(x) - b_i)\,\frac{\partial \log p}{\partial \theta_i}, \tag{2}$$

where α is the learning rate, b_i is the baseline and $\partial \log p/\partial\theta_i$ is the eligibility. Formally, this is still a REINFORCE algorithm but we directly prove that it is gradient-following:

$$\begin{aligned}
E_X[\Delta\theta_i] &= -\,E_X\left[\alpha\,(H(x) - b_i)\,\frac{1}{p}\frac{\partial p}{\partial \theta_i}\right] \\
&= -\sum_{x\in\mathcal{S}} \alpha\,(H(x) - b_i)\,\frac{\partial p}{\partial \theta_i} \\
&= -\,\alpha \sum_{x\in\mathcal{S}} H(x)\frac{\partial p}{\partial \theta_i} + \alpha b_i \sum_{x\in\mathcal{S}} \frac{\partial p}{\partial \theta_i},
\end{aligned}$$

where we have used the fact that $\partial \log p/\partial\theta_i = 1/p \times \partial p/\partial\theta_i$ and that both α and b_i are independent of X. The second term of the sum vanishes since $\sum_{x\in\mathcal{S}} p(x) = 1$ and $\sum_{x\in\mathcal{S}} \partial p/\partial\theta_i = 0$. We exchange the sum and the derivation:

$$\begin{aligned} E_X[\Delta\theta_i] &= -\,\alpha\frac{\partial}{\partial\theta_i}\sum_{x\in\mathcal{S}} H(x)p(x) \\ &= -\,\alpha\frac{\partial}{\partial\theta_i}E_X[H(x)], \end{aligned}$$

which completes the proof. If we let $\mathbf{E} = E_X[H(x)]$, then Eq. 2 is a stochastic approximation of the dynamical system

$$\frac{d\theta}{dt} + \frac{\partial\mathbf{E}}{\partial\theta} = 0.$$

Moreover $\mathbf{E}$ is the internal energy of the system as defined on page 4 in an analogy with statistical physics. Consequently, reinforcement learning is equivalent to minimizing the internal energy of the system (not its free energy).

Although the baseline b_i is not necessary to the proof, it greatly improves performance. For a discussion on this issue see [3, 15]. We have studied two different possibilities for baseline which both relies on first order linear estimation of some expectations.

The first strategy for baseline is the expectation $E_X[H(x)]$ itself. The idea behind it is that if a randomly generated solution x happens to have a cost $H(x)$ lower than the average cost $E_X[H(x)]$, then the automata should be reinforced to make this solution x more likely. The average cost $E_X[H(x)]$ is estimated with the linear recursive equation

$$b(t+1) = \gamma b(t) + (1-\gamma)H(x(t)), \tag{3}$$

where $0 < \gamma < 1$. The baseline is independent of the component of the vector θ. This is the common approach and we will refer to it as the "expectation strategy".

The second strategy for baseline stems from the choice of minimizing the variance of $\Delta\theta_i$ which is written $V_X[\Delta\theta_i]$. We have $V_X[\Delta\theta_i] = E_X[\Delta\theta_i^2] - E_X^2[\Delta\theta_i]$ and the second term is independent of b_i so minimizing the variance reduces to minimizing the first term, $J_i = E_X\left[e_i^2(H(x)-b_i)^2\right]$, where $e_i = \partial \log p/\partial\theta_i$ is the eligibility. J_i is a polynomial of degree 2 in b_i and its minimum is reached at

$$b_i = \frac{E_X\left[e_i^2 H(x)\right]}{E_X\left[e_i^2\right]}.$$

Both the numerator n and the denominator d are estimated with linear recursive equations:

$$\begin{aligned} n(t+1) &= \gamma n(t) + (1-\gamma)e_i^2(t)H(x(t)) \\ d(t+1) &= \gamma d(t) + (1-\gamma)e_i^2(t) \end{aligned} \tag{4}$$

In this strategy each component has a different baseline. We will refer to it as the "minimum variance strategy".

3.3 Algorithm

The following algorithm is a variant of the one described on page 5:

step 1 initialize θ and b (expectation) or n_i and d_i for each component (minimum variance);

step 2 generate $x \in \mathcal{S}$ from $p(\cdot, \theta)$;

step 3 evaluate $H(x)$;

step 4 update θ with Eq. 2;

step 5 update b with Eq. 3 (expectation) or n_i and d_i for each component with Eq. 4 (minimum variance);

step 6 go to step 2.

4 Bernoulli Random Variables

At this point we have seen two dynamics or update rules which operate on the parameter vector of the target or approximate distribution: free energy minimization for Eq. 1 and reinforcement learning for Eq. 2.

We now have to choose a family of distributions from which we can sample efficiently. In doing so, we may restrict the class of problems that are solvable with our approach but this is the only way to derive practical algorithms. In this section, we choose the random variables X_i as being independent. $p(x)$ then writes as the product

$$\prod_i \big(x_i p_i + (1 - x_i)(1 - p_i)\big), \tag{5}$$

where p_i is the probability that x_i equals 1. We emphasize the fact that, in the context of population genetics, the asymptotic distribution of an infinite population of binary strings evolved through crossover only is precisely of the form of Eq. 5 and that convergence to equilibrium is fast (see [13]).

The n numbers p_i cannot be the parameters of the distribution on which Eq. 1 or Eq. 2 operate since they are bounded over $[0, 1]$ and nothing in these equations ensures that this constraint is enforced. We thus insert a nonlinear function which bounds the p_i. We choose for example a sigmoid such as

$$p_i = g(\theta_i) = \frac{1}{2}\big(1 + \tanh(\beta\theta_i)\big).$$

Hopfield has used this technique in his fully connected neural networks [7]; it is also used in multilayer perceptrons trained with backpropagation.

From the expression of $p(x)$, we compute the gradient of its logarithm:

$$\begin{aligned}\frac{\partial \log p}{\partial \theta_i} &= \frac{\partial \log p}{\partial p_i} \times \frac{\partial p_i}{\partial \theta_i} \\ &= \frac{2x_i - 1}{x_i p_i + (1 - x_i)(1 - p_i)} g'(\theta_i)\end{aligned}$$

We take advantage of the fact that $(\tanh)' = 1 - \tanh^2$ and express $g'(\theta_i)$ as a function of p_i:

$$g'(\theta_i) = 2\beta p_i(1 - p_i)$$

We substitute $g'(\theta_i)$ in the expression of the derivative:

$$\frac{\partial \log p}{\partial \theta_i} = 2\beta(2x_i - 1)\frac{p_i(1 - p_i)}{x_i p_i + (1 - x_i)(1 - p_i)}$$

We finally factorize both cases $x_i = 0/1$ into the following expression:

$$\frac{\partial \log p}{\partial \theta_i} = 2\beta(x_i - p_i)$$

The mean of the random variable X_i is p_i. We then write:

$$\frac{\partial \log p}{\partial \theta} = 2\beta(x - \mu) \tag{6}$$

where $\mu = (p_1, \ldots, p_n)^t$ is the mean of X.

Narendra and Thathachar [12] have previously studied similar stochastic learning automata based upon Bernoulli random variables, for which the environment is itself stochastic rather than determinist. Baluja and Caruana [2] have proposed an algorithm which combines a population technique with selection and an update rule similar to that proposed by Narendra and Thathachar. Let us describe their algorithm.

step 1 initialize μ with $p_i = 1/2$, $i = 1, \ldots, n$;

step 2 generate a population of N individuals sampled from the Bernoulli distribution p with parameter μ;

step 3 evaluate each individual according to H;

step 4 select the fittest x among the population;

step 5 update μ using the fittest x with the rule

$$\mu(t + 1) = \mu(t) + \alpha(x - \mu(t)); \tag{7}$$

step 6 go to step 2.

Provided that $0 < \alpha < 1$ and $\mu(t)$ lies inside the hypercube $\mathcal{S}$ and since $x \in \mathcal{S}$ the resulting μ will lie inside this same hypercube. There is no need for a bounding function in this approach. The right part of Eq. 7 is similar to the expression of the log-derivative in Eq. 6. The main difference with our work is that the cost function H is made implicit in Eq. 7 via selection, in contrast to both Eq. 1 and 2.

5 Gaussian Probability Distribution

The choice of independent Bernoulli random variables is the simplest one and may not fit complex problems. One direction to improve algorithm performance may be to take into account second-order statistics. In the context of combinatorial optimization, it means being able to generate binary random variables following second-order statistics specifications, not estimating those statistics. To this regard, the Gaussian probability distribution is of interest and can be considered as a quadratical approximation of optimal distributions such as the Gibbs distribution p_T^* defined on page 3. In this section we will mainly compute the log-derivative of the Gaussian probability density p.

For any real-valued n-dimensional vector x, we consider the Gaussian density

$$p(x) = \frac{1}{(2\pi)^{n/2}} (\det C)^{1/2} \exp\left(-\frac{1}{2}(x-\mu)^t C (x-\mu) \right),$$

where C is the inverse of the covariance matrix, μ is the mean and $\det C$ is the determinant of C. Since C is symmetric positive definite, it can be factorized into its positive Choleski's decomposition $C = LL^t$, where L is a lower triangular matrix with $l_{ii} > 0$. We substitute L in $p(x)$:

$$p(x) = \frac{1}{(2\pi)^{n/2}} \times \det L \times \exp\left(-\frac{1}{2}(x-\mu)^t LL^t (x-\mu) \right)$$

Its logarithm writes:

$$\log p(x) = \text{constant} + \log\det L - \frac{1}{2}(x-\mu)^t LL^t (x-\mu)$$

We want to compute

- $\partial \log p(x)/\partial L$ which is the matrix of general element $\partial \log p(x)/\partial l_{ij}$ and
- $\partial \log p(x)/\partial \mu$ which is the vector of general element $\partial \log p(x)/\partial \mu_i$.

We first compute the matrix $\partial \log p(x)/\partial L$. Since L is lower triangular its determinant is equal to $\prod_i l_{ii}$. Consequently we have:

$$\frac{\partial}{\partial L}(\log\det L) = \operatorname{diag}\left(\frac{1}{l_{ii}}\right)$$

We now use the fact that $\partial(x^t LL^t x)/\partial L = 2xx^t L$ to derive the last term of $\log p(x)$:

$$\frac{\partial}{\partial L}\left((x-\mu)^t LL^t (x-\mu) \right) = 2(x-\mu)(x-\mu)^t L$$

We finally have:

$$\frac{\partial \log p(x)}{\partial L} = \operatorname{diag}\left(\frac{1}{l_{ii}}\right) - (x-\mu)(x-\mu)^t L$$

We now compute the log-derivative with respect to the mean μ. We simply have:

$$\frac{\partial \log p(x)}{\partial \mu} = LL^t(x-\mu)$$

An important issue is the complexity of the Gaussian approach. The complexity of update rules is proportional to n^3 (matrix product). Another time-consuming operation is the random generation of samples. In fact we use the affine transformation $x = (L^t)^{-1}y + \mu$ to produce a sample x from $p(x)$, y being a normal Gaussian random variable. The complexity lies in the inversion of L which is proportional to n^3. This complexity is independent of that of the cost function evaluation.

There are several constraints on this model. First the matrix L has to stay lower triangular. We simply do not update elements l_{ij} for $j > i$.

Second, the matrix L has to stay invertible or, equivalently, its diagonal elements l_{ii} have to stay positive. This constraint is not taken into account in any of the update rules (see equations 1 and 2). One way to ensure that diagonal elements of L stay positive is to introduce n new parameters λ_i such that $l_{ii} = e^{\lambda_i}$ and make the dynamical system evolve λ_i instead of l_{ii}. This technique is close to that of the sigmoid function described on page 8. It has been implemented but has not improved convergence on the four peaks problem.

Finally, we deal with binary values not real values. Consequently, any outcome x of the random generator has to be mapped onto a binary value x' such that $x'_i = 1$ if $x_i > 1/2$ and $x'_i = 0$ otherwise. This mapping may introduce some distortion between second-order statistics of both sets of random variables.

Sabes and Jordan have used Gaussian units in the context of associative learning by probability matching [14]. They have chosen a linear combination of Gaussian kernels with covariance matrices equal to $\sigma^2 I$. The Gaussian law is also used to model complex mutation schemes in evolutionary strategies [1], especially in case of self-adaptation.

6 Summary

We recall the two dynamics we have encountered:

Free Energy Minimization

$$\Delta\theta = -\alpha\big(H + T(1 + \log p)\big)\frac{\partial \log p}{\partial \theta}$$

Reinforcement Learning

$$\Delta\theta = -\alpha(H - b)\frac{\partial \log p}{\partial \theta}$$

Both update rules are similar and the expression $-T(1 + \log p)$ taken from the free energy minimization update rule can be identified with the baseline of the reinforcement learning update rule.

We also recall the two log-derivatives each corresponding to a different way to parameterize the target distribution p:

Bernoulli Model

$$\frac{\partial \log p}{\partial \theta} = 2\beta(x - \mu)$$

Gaussian Model

$$\frac{\partial \log p(x)}{\partial L} = \operatorname{diag}\left(\frac{1}{l_{ii}}\right) - (x - \mu)(x - \mu)^t L$$

$$\frac{\partial \log p(x)}{\partial \mu} = LL^t(x - \mu)$$

7 Four Peaks Problem

7.1 Problem definition

This problem is taken from [2]. The cost function to be maximized is

$$H(x) = \max\big(z(x), o(x)\big) + R(x),$$

where $z(x)$ is the number of trailing "0" in the string x and $o(x)$ is the number of leading "1". $R(x)$ is a conditional reward defined by:

$$R(x) = \begin{cases} n & \text{if } z(x) > s \text{ and } o(x) > s, \\ 0 & \text{otherwise,} \end{cases}$$

where s is a threshold equal to $n/10$. Let us give some examples in the case $n = 6$ and $s = 1$:

- for $x = 111111$, $R(x) = 0$ and $H(x) = 6$ (local maximum);
- for $x = 111110$, $R(x) = 0$ and $H(x) = 5$;
- for $x = 111100$, $R(x) = 6$ and $H(x) = 10$ (global maximum).

We want to maximize H, which is equivalent to minimizing $-H$. H has four maxima, among which two are global maxima and two are local maxima. Local search methods quickly become trapped into attraction basins of local maxima. Quoting [2], as s increases,

> "the basins of attraction surrounding the inferior local optima increase in size exponentially while the basins around the global optima decrease at the same rate".

Moreover H has large plateaus which bring no information to local search methods.

7.2 Evaluation with Bernoulli random variables

De Bonet, Isbell and Viola propose an optimization algorithm called MIMIC and compare it with three other algorithms [4]:

- Population Based Incremental Learning as described on page 8,
- Random Hill Climbing and
- Standard Genetic Algorithm.

We first briefly describe MIMIC and then compare our algorithm with those reviewed in [4].

The main idea in MIMIC is to estimate the probability distribution of solutions whose cost is below a given threshold. The threshold is then lowered and new samples are generated from the estimated distribution. The process is iterated until a good solution is found. However, 2^n numbers are needed to completely describe the probability distribution of n binary random variables. De Bonet, Isbell and Viola reduce the complexity with distributions which can be expressed as

$$p(x_{\sigma(1)}|x_{\sigma(2)})p(x_{\sigma(2)}|x_{\sigma(3)})\ldots p(x_{\sigma(n-1)}|x_{\sigma(n)})p(x_{\sigma(n)}),$$

where σ is a permutation of $1, 2, \ldots, n$. This implies a search for σ which is performed with a greedy algorithm minimizing a KL divergence.

We follow the benchmark described in [4]. We average over 1000 independent trials the number of cost function evaluations before a global maximum is found. This is different from the experimental protocol followed in [2], which consists in allowing a fixed number of cost function evaluations and averaging the best evaluations over 25 runs.

We have evaluated the reinforcement algorithm of page 8 with Bernoulli random variables. We have carried out two series of experiments, with the expectation strategy and the minimum variance one. All results are given in Tab. 1 and Tab. 2. The values of simulation parameters are: learning rate $\alpha = 0.5$, baseline factor $\gamma = 0.9$, sigmoid gain $\beta = 1/n$, where n is the size of the problem.

The results with the minimum variance strategy are significantly worse than those with the expectation one. At least for this problem, the minimum variance strategy is not the optimal policy. It is not clear why the expectation one is a better choice.

MIMIC and PBIL have better results than GA and RHC. MIMIC finds an optimum in approximately one tenth the number of evaluations required by PBIL. Our algorithm performs better than PBIL but worse than MIMIC. MIMIC and PBIL show a complexity which is linear in the problem size on the contrary to our algorithm.

Problem Size	30	40	50	60	70	80
Min.	227	1584	4351	9934	17524	38759
Mean	8065	21169	38320	69110	109160	169260
Median	5455	15454	32125	58878	99122	154270
Max.	102860	313260	754300	990030	386350	793520
Std. Dev.	9002	26089	33493	55980	51577	76104
Avg. Dev.	5277	12436	17632	29634	38839	55227

Table 1: Average time before a global maximum is found for different sizes of problem (expectation strategy).

Problem Size	30	40	50	60	70	80
Min.	206	1463	3736	11102	19179	37191
Mean	9973	28238	52066	91049	152330	229530
Median	6951	19113	42292	82019	136300	208370
Max.	117230	1220500	979850	637520	1026500	1864500
Std. Dev.	10937	56052	49568	55267	84558	114760
Avg. Dev.	6751	18799	25920	37007	57709	78550

Table 2: Average time before a global maximum is found for different sizes of problem (minimum variance strategy).

7.3 Evaluation with the Gaussian probability distribution

We have evaluated a reinforcement algorithm with the Gaussian probability distribution and the expectation strategy. Due to the complexity of the approach (in n^3) and available computational power, we only consider the case $n = 30$. Figure 1 shows the evolution of the cost function value until a global maximum is found. The case $s = 3$ is rather easy since the algorithm quickly "finds" the reward and the remaining of the graph shows the effect of a stochastic gradient ascent. The results are given in the table below:

Min.	Mean	Median	Max.	Std. Dev.	Avg. Dev.
10149	15692	15388	21993	2751	2126

These numbers are calculated from the outcome of 100 instead of 1000 independent trials. The algorithm performs in average worse than Bernoulli random variables with expectation strategy. However worst case analysis, i.e. the maximum number of cost function evaluations before global optimum, shows that the Gaussian model is superior to the Bernoulli model. The Gaussian algorithm has also proved to be very sensitive to the value of the learning rate: higher learning rates lead to faster convergence if any but are also more likely to make the algorithm diverge. Consequently we have chosen a learning rate which ensures convergence and slows down the algorithm. The standard and average deviations of the number of cost function evaluations are significantly lower than with the Bernoulli model.

The complexity of the approach makes it rather impractical although it may use statistical dependencies between random variables better than the simpler scheme of Bernoulli random variables.

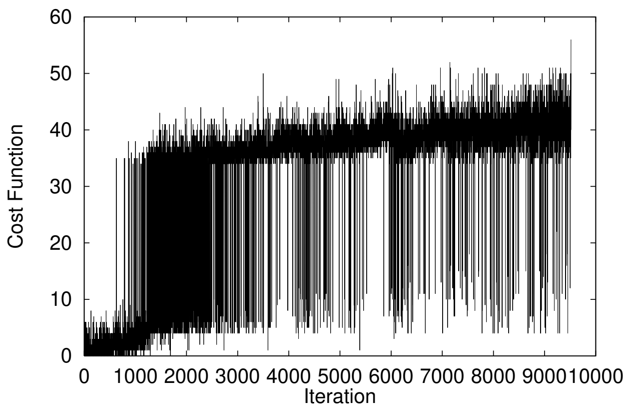

Figure 1: Evolution of the cost function value until a global maximum is reached.

8 Prisoner's Dilemma

8.1 Problem definition

The prisoner's dilemma is a game for two players where each player chooses an action among two possible actions in a synchronous and independent manner. The two actions are:

- Cooperate
- Defect

Depending on the outcome of the round, each player is credited with a payoff whose matrix is given in the following table:

	D	C
D	1	5
C	0	3

A game is a finite sequence of rounds and each player tries to maximize its accumulated score. This is known as the iterated prisoner's dilemma. The problem consists in designing a strategy in order to maximize a player's score.

We start with the encoding proposed in [11] for a genetic algorithm. Each action is coded with one bit (D/C). A strategy is defined as the way to play in the light of the 3 previous rounds (or 6 actions). A 64 bit string encodes such a strategy. For the first round, a player has to make an assumption for the 3 virtual previous rounds which can be coded with 6 more bits. Finally 70 bits are necessary to encode a full strategy.

8.2 Simulations

In the following simulations, two prisoners play against each other. They are implemented with Bernoulli random variables as seen on page 8.

The cost function to be maximized is defined as the score at the end of the game (finite sequence of rounds). The simulation algorithm is organized as a loop which decomposes as:

1. generate strategies for each player;
2. for 1000 rounds
    - generate first player's choice;
    - generate second player's choice;
    - credit each player with the corresponding payoff;
3. compute both average payoffs;
4. reinforce both automata.

We put the stress on the fact that the environment of each player (its cost function H) is not stationary on the contrary to the analysis of previous sections: the opponent evolves itself and tries to maximize its own score.

We have first implemented the reinforcement learning algorithm of page 8. Figure 2 shows the evolution of a player's score. In this simulation, both players share the same parameters. The most common limit strategy is cooperation, each player being credited with 3 points in average. The algorithm also converges sometimes to suboptimal strategies, i.e., non-cooperative ones which can be interpreted as local optima.

We have also implemented the free energy minimization algorithm of page 5. The parameters of these simulations are: learning rate $\alpha = 1$ and sigmoid gain $\beta = 0.05$.

Figure 3 shows the evolution of a player's score for different values of temperature. From an analogy with statistical physics, at high temperature the system is purely random and at low temperature the system is close to low energy states (or ground states). This is what we observe in this simulation with the exception of the first graph: high temperatures introduce a bias in Eq. 1. At the beginning of the simulation, we have $p = 1/2^n$. Let us define the quantity

$$A = H + T(1 + \log p) < H_{\max} + T(1 - n \log 2).$$

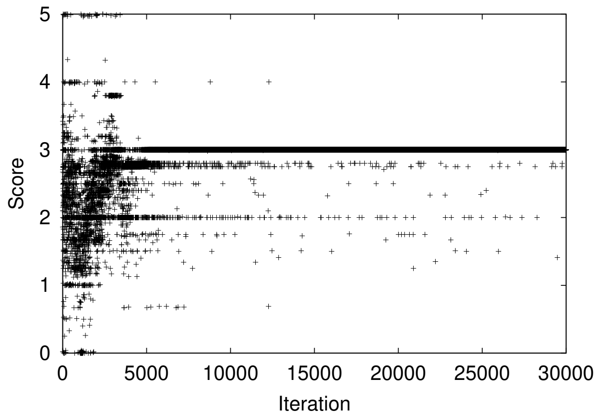

Figure 2: Prisoner's average score evolution for a reinforcement algorithm.

For $T > H_{\max}/(n \log 2 - 1)$ A is always negative hence the bias and the fast "saturation" of sigmoid functions. This results in "less random" than expected behaviors. Figure 4 shows the evolution of the parameter θ_i of a Bernoulli random variable X_i (which encodes the player's response to three particular previous rounds) and Fig. 5 shows the evolution of the corresponding probability $p_i = g(\theta_i)$.

9 Conclusion

We have shown how to apply statistical learning schemes to combinatorial optimization problems. The approach has been divided into two parts: the target or approximate probability distribution of the random variables which is parameterized by a real vector and the dynamics which operates on the parameters and makes the target distribution converge to a limit distribution charging optimal solutions. Two families of dynamics, free energy minimization algorithms and reinforcement learning algorithms, and two families of probability distributions, Bernoulli random variables and Gaussian random variables, have been studied. Several simulations have been carried out with both distributions and both dynamics. The algorithms have been applied to two optimization problems: the four peaks problem, for which our results have been compared to those yielded by other approaches, and the iterated prisoner's dilemma problem. In the former, the environment is stationary whereas it is not in the latter since both player strategies evolve at the same time. In the case of the free energy minimization, we have shown the influence of the temperature on the evolution of the average payoffs.

Both dynamics lead to global optima although only local optima may be found in the case of bigger or less regular problems. The free energy min-

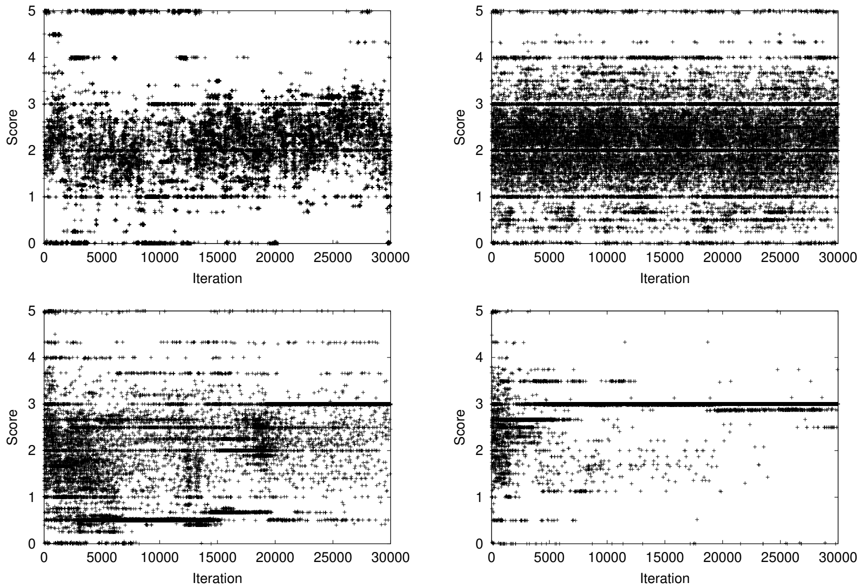

Figure 3: Prisoner's average score evolution for different values of temperature. From left to right and from top to bottom, we have $T = 1$, 0.1, 0.06 and 0.005.

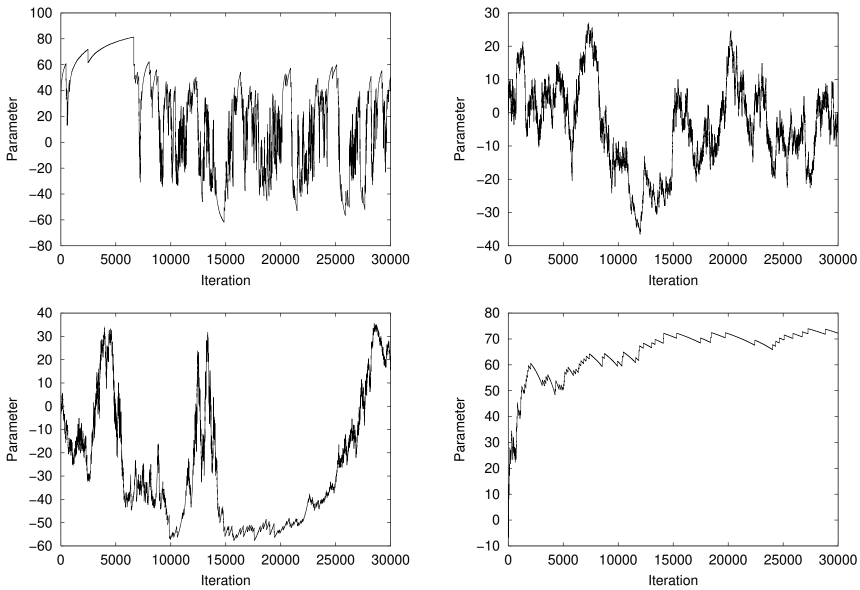

Figure 4: Evolution of the parameter θ_i of a Bernoulli random variable X_i for the same values of temperature as in Fig. 3.

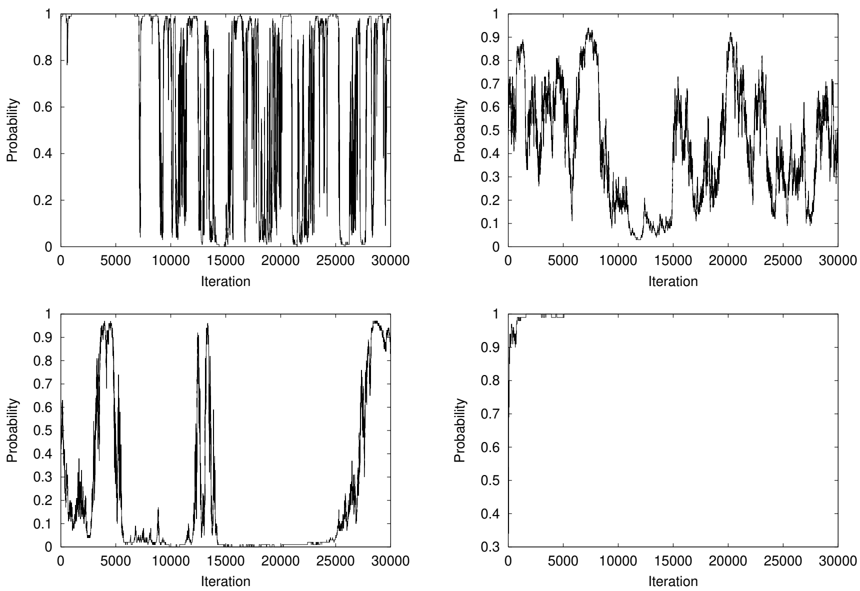

Figure 5: Evolution of the probability p_i of a Bernoulli random variable X_i for the same values of temperature as in Fig. 3.

imization dynamics automatically adjust its baseline on the contrary to the reinforcement learning one, which requires a comparison strategy. However, the former introduces a new parameter, the temperature, that one has to choose or control with a cooling scheme. Finally many thermodynamical quantities may be measured with the free energy minimization dynamics (such as the specific heat). The expectation strategy for reinforcement learning proved to be more efficient than the minimum variance one. We have no theoretical result about this fact but we believe that the latter makes the algorithm follow the trajectories of the determinist dynamical system more closely and therefore lowers its exploration ability.

Bernoulli random variables lead to algorithms which are faster (in terms of cost function evaluations) but less stable than their Gaussian counterparts. The Gaussian distribution seems less practical due to its time (matrix inversion) and space complexity (parameters) and requires finetuning of the learning rate to avoid divergence.

More experiments need to be carried out. Both dynamics should be compared directly and the overall approach should be applied to other problems. It should be instructive to compare our approach against well-known strategies for the prisoner's dilemma problem.

There are many open problems relative to combinatorial optimization by means of statistical learning. The first one is to determine the class of problems that are solvable with this approach. A strongly related issue is the regularity conditions which the cost function should fulfill.

References

[1] T. Bäck, U. Hammel, and H.-P. Schwefel. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):pp. 3–17, April 1997.

[2] 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.

[3] A. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 13(5):834–846, September 1983.

[4] J.S. De Bonet, C.L. Isbell, and P. Viola. MIMIC: finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems*, volume 9. MIT Press, Denver, 1996.

[5] M. Duflo. *Algorithmes stochastiques*. Springer-Verlag, 1996.

[6] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, 6(6):721–741, November 1984.

[7] J.J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci.*, 81:3088–3092, May 1984.

[8] M. Jerrum and A. Sinclair. *Approximation algorithms for NP-hard problems*, chapter The Markov chain Monte Carlo method: an approach to approximate counting and integration. PWS, 1996.

[9] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[10] E. Levin, N. Tishby, and S.A. Solla. A statistical approach to learning and generalization in layered neural networks. *Proceedings of the IEEE*, 78(10):1568–1574, October 1990.

[11] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, 3rd rev. and extended edition, 1996.

[12] K.S. Narendra and M.A.L. Thathachar. *Learning automata: an introduction*. Prentice Hall, NJ, 1989.

[13] Y. Rabani, Y. Rabinovitch, and A. Sinclair. A computational view of population genetics. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, 1995.

[14] P.N. Sabes and M.I. Jordan. Reinforcement learning by probability matching. In *Advances in Neural Information Systems*, volume 8, pages 1080–1086. MIT Press, Cambridge, MA, 1995.

[15] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.