

Data Fusion Algorithms for Collaborative Robotic Exploration

Jeremy Thorpe and Robert McEliece.

Abstract.

In this paper we will study the problem of efficient data fusion in an ad hoc network of mobile sensors (“robots”) using belief propagation on a graphical model similar to those used in turbo-style decoding. We also devise a new metric for evaluating the performance of general inference algorithms, including BP, which return “soft” estimates.

1. Introduction. The Robot Inference Problem.

In the future, NASA spacecraft sent to extraterrestrial planets to collect scientific data may deploy a large number of small, inexpensive robots. Each of the robots may be equipped with a number of sensors with which to measure its local microenvironment. Individually, one robot’s measurements would convey little information about the global situation. Collectively, however, the robots could overcome this by communicating with their neighbors and fusing their data. The robots should thus be able to collectively infer a great deal about the global environment. The trick is to do it with as little communication and/or computation as possible.

In this paper, we will begin to explore the problem of designing efficient data fusion strategies in an ad hoc network of robotic sensors, by first defining a simplified model for the problem (Section 2), and then applying the celebrated belief propagation algorithm [11] to obtain accurate estimates of the *a posteriori* probabilities about the environment (Section 3). We will measure the accuracy of these estimates using a new criterion which we derive axiomatically in the appendices.

2. A Simplified Model for the Robot Inference Problem.

In our model, the unknown environment is characterized by a random binary n -vector $\mathbf{X} = (X_1, \dots, X_n)$, with the component X_i 's chosen independently, and $\Pr\{X_i = +1\} = \Pr\{X_i = -1\} = 1/2$. The sensor network consists of m robots, and the j th robot's ability to sense the environment is characterized by an n -vector $\mathbf{g}_j = (g_{j,1}, \dots, g_{j,n}) \in \{0, +1, -1\}^n$. We assume each robot has exactly s nonzero components, which means it can sense exactly s components of \mathbf{X} . (A zero component of \mathbf{g}_j means that the robot cannot sense the corresponding component of \mathbf{X} .) The measurement y_j of the environment \mathbf{x} by a robot \mathbf{g}_j is the dot product $y_j = \mathbf{x} \cdot \mathbf{g}_j$. Thus the aggregate evidence about a particular environment \mathbf{x} provided by the robot sensor network is the vector $\mathbf{y} = (y_1, \dots, y_m) = G\mathbf{x}$, where G is the $m \times n$ matrix whose (i, j) th entry is $g_{j,i}$.

For future reference, we introduce the following notation. If \mathbf{y} is an n -vector, and j is an integer in the range $j = 1, \dots, m$, \mathbf{y}^j denotes the components of \mathbf{y} corresponding to the nonzero components of \mathbf{g}_j . Thus for example if \mathbf{x} represents the environment, \mathbf{x}^j represents the part of the environment sensed by the j th robot. In this notation, \mathbf{g}_j^j denotes the vector \mathbf{g}_j with its zero components deleted. As a result, we have $y_j = \mathbf{x}^j \cdot \mathbf{g}_j^j$.

For example, if $n = 6$, $m = 4$, and $s = 3$, a possible choice for G is displayed below.

$$G = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{pmatrix} -1 & +1 & +1 & & & \\ & +1 & & -1 & -1 & \\ & -1 & & -1 & +1 & \\ +1 & & -1 & +1 & & \end{pmatrix} \end{matrix}.$$

Figure 1 gives a graphical representation of this robot-environment scenario. Note that we have the following “local environments”:

$$(1) \quad \begin{aligned} \mathbf{x}^1 &= (x_1, x_2, x_3) \\ \mathbf{x}^2 &= (x_3, x_5, x_6) \\ \mathbf{x}^3 &= (x_2, x_5, x_6) \\ \mathbf{x}^4 &= (x_1, x_3, x_4). \end{aligned}$$

Also, we have

$$(2) \quad \begin{aligned} \mathbf{g}_1^1 &= (-1, +1, +1) \\ \mathbf{g}_2^2 &= (+1, -1, -1) \\ \mathbf{g}_3^3 &= (-1, -1, +1) \\ \mathbf{g}_4^4 &= (+1, -1, +1). \end{aligned}$$

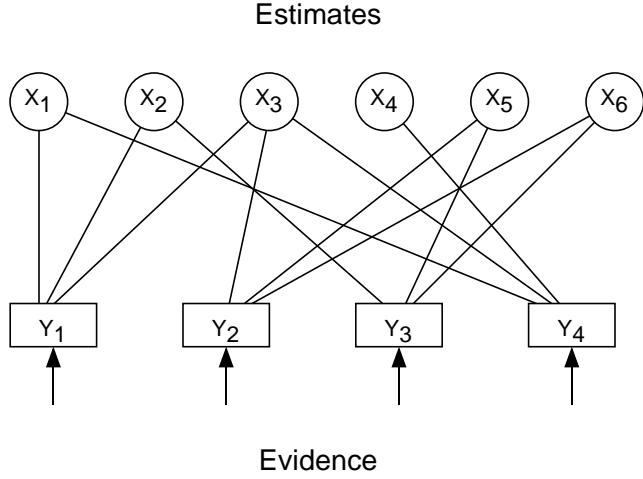


Figure 1. A robotic inference network. The topology of the network reflects the matrix G . (There is an edge from Y_j to X_i if $g_{j,i} \neq 0$.)

The basic problem is to “infer” the environment $\mathbf{X} = (X_1, \dots, X_n)$ using the evidence provided by the sensor outputs Y_1, \dots, Y_m . In other words, for each index $i \in \{1, \dots, n\}$, we wish to produce an estimate of the conditional probabilities $\Pr\{X_i = +1|\mathbf{Y}\}$ and $\Pr\{X_i = -1|\mathbf{Y}\}$, where $\mathbf{Y} = (Y_1, \dots, Y_m)$. In principle, the exact values of these *a posteriori* probabilities can be obtained by Bayes’ rule:¹

$$\Pr\{X_i = \epsilon|\mathbf{Y} = \mathbf{y}\} = \frac{\Pr\{\mathbf{Y} = \mathbf{y}|X_i = \epsilon\} \Pr\{X_i = \epsilon\}}{\Pr\{\mathbf{Y} = \mathbf{y}\}} \propto \Pr\{\mathbf{Y} = \mathbf{y}|X_i = \epsilon\}.$$

Unfortunately, the direct computation of the likelihoods $\Pr\{\mathbf{Y} = \mathbf{y}|X_i = \epsilon\}$ (for $\epsilon = \pm 1$) requires testing each of the 2^n possible \mathbf{X} ’s for compatibility with \mathbf{Y} :

$$\begin{aligned} \Pr\{\mathbf{Y} = \mathbf{y}|X_i = \epsilon\} &= \frac{\sum_{\mathbf{x}:x_i=\epsilon} \Pr\{\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}\} \Pr\{\mathbf{X} = \mathbf{x}\}}{\Pr\{X_i = \epsilon\}} \\ &\propto \sum_{\mathbf{x}:x_i=\epsilon} \Pr\{\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}\} \\ &\propto \sum_{\mathbf{x}:x_i=\epsilon} \delta(\mathbf{y} - G\mathbf{x}), \quad (3) \end{aligned}$$

where δ is the Kronecker delta function. If n is large, this calculation will be prohibitively complex, and alternative methods must be adopted. We discuss one such method, *loopy belief propagation*, in the next section.

¹ In the following calculations, we use the “is proportional to” symbol (\propto) in which the implied proportionality constants are chosen so that the quantities with $\epsilon = +1$ and $\epsilon = -1$ sum to one.

3. Loopy Belief Propagation.

Belief Propagation is a general algorithm which solves, exactly in some cases and approximately in others, a certain class of probabilistic inference problems. Roughly speaking, BP is applicable whenever the inference problem can be represented by a graphical model of a particular kind. The term “belief propagation” was coined by Pearl [9, 11], using a Bayesian network as the underlying graphical model; alternative formulations of equivalent algorithms are “probability propagation” on junction trees or factor graphs [12, 8, 7], the “sum-product algorithm” on normal graphs [4], and the “generalized distributive law” [1, 2] on junction trees or graphs. In the following discussion, we will use the GDL/junction graph formulation.

The GDL is an iterative algorithm which works by making progressively improving estimates of the *a posteriori* probabilities (or beliefs) about a set of hidden variables, by passing messages on a junction graph. If the junction graph is cycle-free, the GDL solves the inference problem exactly [9,1]. However, if cycles are present, then the GDL’s solution is only approximate. This is sometimes called *loopy BP* [2,5,13].

A belief propagation decoder for the robotic inference problem described in Section 2 is shown in Figures 1 and 2. The graph in Figure 1 represents a Bayesian network for the $n + m$ random variables X_1, \dots, X_n and Y_1, \dots, Y_m . This network is in the form of a bipartite graph, in which the hidden variables to be inferred are on the top, and the evidence nodes are on the bottom.

The GDL algorithm works by passing messages back and forth between the X_i nodes and the Y_j nodes. Each of these messages is a function $f(\epsilon)$, for $\epsilon \in \{0, 1\}$, such that $0 \leq f(\epsilon) \leq 1$. The message from X_i to Y_j is denoted by $m_{i,j}(\epsilon)$, and can be interpreted as an estimate of the *probability* $\Pr\{X_i = \epsilon | Y_j = y_j\}$, for $\epsilon = \pm 1$. The message from Y_j to X_i is denoted by $\mu_{j,i}(\epsilon)$, and can be interpreted as an estimate of the *likelihood* $\Pr\{Y_j = y_j | X_i = \epsilon\}$, for $\epsilon = \pm 1$.

Initially, the $\mu_{j,i}(\epsilon)$ ’s are set to 1. Recursively, the m -messages are calculated by the rule

$$(4) \quad m_{i,j}(\epsilon) \propto \prod_{k \neq j} \mu_{k,i}(\epsilon),$$

where the constant of proportionality is chosen so that $m_{i,j}(0) + m_{i,j}(1) = 1$. The μ -messages are updated by the rule

$$(5) \quad \mu_{j,i}(\epsilon) = \sum_{\mathbf{x}^j: x_i^j = \epsilon} \alpha_j(\mathbf{x}^j) \prod_{k \neq i} m_{k,j}(x_k).$$

In (5), the function $\alpha_j(\mathbf{x}^j)$ is (in the parlance of [1]) the *local kernel* at Y_j , and is defined as follows:

$$\alpha_j(\mathbf{x}^j) = \begin{cases} 1 & \text{if } \mathbf{x}^j \cdot \mathbf{g}_j^j = y_j \\ 0 & \text{if } \mathbf{x}^j \cdot \mathbf{g}_j^j \neq y_j. \end{cases}$$

(For the notation \mathbf{g}_j^j , see (1) and (2).) In words, $\alpha_j(\mathbf{x}^j)$ tests the s -vector \mathbf{x}^j for compatibility with the observed measurement y_j .

An iteration of the algorithm consists of a complete updating of all messages in both directions. After any iteration, we can choose (or not) to compute *beliefs* in the components of \mathbf{x} :

$$b_i(\epsilon) = \alpha \prod_{j=1}^m \mu_{j,i}(\epsilon).$$

These beliefs are the output of the algorithm.

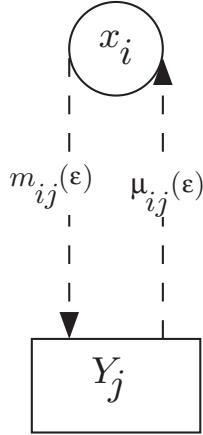


Figure 2. The message-passing scheme of the GDL
 (Cf. Figure 1). The $m_{i,j}(\epsilon)$'s are approximations
 to $\Pr\{X_i = \epsilon | Y_j = y_j\}$, and the $\mu_{j,i}(\epsilon)$'s
 are approximations to $\Pr\{Y_j = y_j | X_i = \epsilon\}$.

4. Experimental Results.

We applied the loopy BP algorithm described in Section 3 to to robot inference problem described in Section 2, using the graphical model shown in Figure 1. The results are shown in Figures 3 and 4. The parameters are $s = 10$ non-zero elements per robot, with $n = 20$ hidden variables in Figure 3 and $n = 50$ in Figure 4.²

In Figures 3 and 4 the independent variable (horizontal axis) is m , the number of robots, and the dependent variable (vertical axis) is the “average uncertainty,” measured in bits, per component of \mathbf{X} . By this we mean the following. For each run of our simulation, we generated an n -bit “environment” (x_1, \dots, x_n) drawn randomly from the set of 2^n possibilities. Then we selected the entries of the $m \times n$ matrix G randomly, subject to the constraint that each row have exactly s nonzero entries, and calculated the corresponding evidence vector $\mathbf{y} = G\mathbf{x}$. Then we ran the loopy BP inference algorithm, which returned a set of beliefs $(b_i(+1), b_i(-1))$ for $i = 1, \dots, n$, where $b_i(\epsilon)$ denotes the algorithm’s belief in the event $X_i = \epsilon$. We then defined the “uncertainty” of the algorithm for that particular run as the quantity

$$\Delta = \frac{1}{n} \sum_{i=1}^n -\log_2 b_i(x_i),$$

(An axiomatic justification of this measure is given in the Appendices.) In Figures 3 and 4, the vertical axis represents the average value of Δ , where the averaging was done over the set of runs.

In Figure 3, we see that it takes 20 sensors five iterations to reduce the average uncertainty to approximately 0.1 bit for each of 20 bits in \mathbf{X} , compared to about half that number for an exhaustive exact solution. In Figure 4 we see that it takes about 34 sensors five iterations to reduce the uncertainty of 50 components to the same value. (In this case, an exhaustive exact solution is not feasible.)³ Note that the ratio m/n for an uncertainty of 0.1 bit per environment bit is $20/20 = 1.00$ in Figure 3 and $34/50 = 0.68$ in Figure 4, so that an economy of scale is evident. We suspect that with $s = 10$, as n increases, the ratio of robots to environment components will decrease to a value of about 0.34, based on the following information-theoretic argument.

² It is important to understand that despite the “message-passing” nature of this algorithm, it is nevertheless *centralized*, i.e. the observations Y_1, \dots, Y_m must first be transmitted to a central location, e.g. a base station, before the data fusion begins. A study of *decentralized* BP remains an interesting challenge.

³ If the number of iterations is increased, simulations show that the performance is not improved in either case.

Each Y_j is has a binomial density function, i.e.,

$$\Pr\{Y = s - 2k\} = \frac{1}{2^s} \binom{s}{k} = b(s, k),$$

for $k = 0, 1, \dots, s$, so that the entropy of each Y_j is

$$h_B(s) = - \sum_{k=0}^s b(s, k) \log_2 b(s, k).$$

Thus the minimum conceivable number of s -component sensors needed to determine $\mathbf{X} = (X_1, \dots, X_n)$ exactly is $H(\mathbf{X})/h_B(s) = n/h_B(s)$, i.e., for $m < n/h_B(s)$, exact inference is impossible. For $s = 10$ we have $h_B(10) = 2.70643$, and so in Figures 3 and 4 we have placed an “ \times ” on the horizontal axis at the point $m = 20/2.70643 = 7.39$ and $m = 50/2.70643 = 18.47$, respectively, to represent this entropy bound.

Finally, it is worth noting that with loopy BP, the number of messages which must be passed at each iteration is $2ms$, which is twice the number of edges in the graph of Figure 1. Thus if I denotes the number of iterations, the total number of messages passed (a reasonable measure of complexity) is $2msI$. On the other hand, an exhaustive brute-force calculation of the exact a posteriori probabilities (see eq. (3)) requires the calculation of $G\mathbf{x}$ for all 2^n values of \mathbf{x} , which requires about $2^n ms$ arithmetic operations. In summary:

$$\text{Complexity of Direct Solution} \approx 2^n ms$$

$$\text{Complexity of Loopy BP} \approx 2Ims.$$

Thus provided the number of iterations is kept small, loopy BP is much less complex than a brute-force approach.

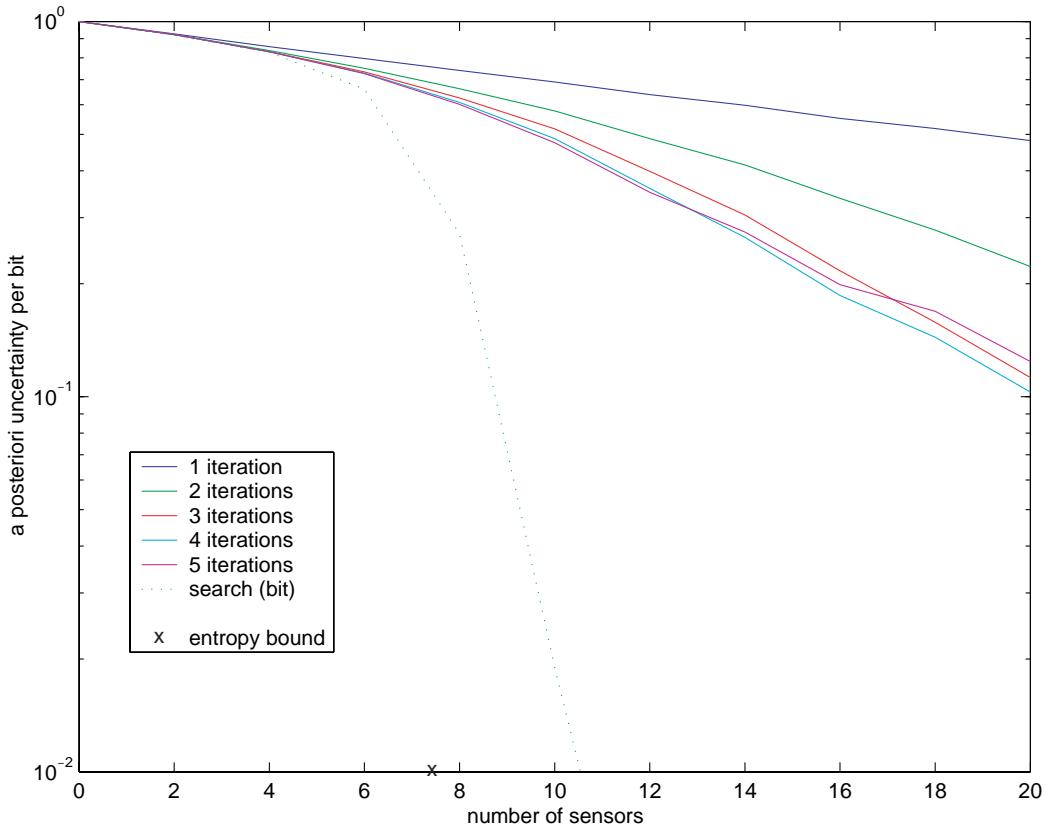


Figure 3. Loopy BP inference performance for an unknown environment \mathbf{X} with $n = 20$ components, where each robot senses $s = 10$ components of \mathbf{X} .

5. Conclusions and Suggestions for Further Work.

In this paper, we have built on expertise gained in solving the probabilistic inference problem in the context of decoding algorithms [6,10] in particular the approximate and exact solutions obtained using belief propagation on networks with and without cycles, to the problem of designing *intelligent communication networks of collaborative robots*. The underlying idea, of course, is that the problem of interpreting scientific data gathered at separated locations is an instance of the probabilistic inference problem. Our results are quite promising, and suggest that loopy belief propagation may represent a low-complexity, high accuracy, approach.

As for further work in this subject, we might suggest the following topics.

- As we noted in Section 4, despite the message-passing nature of loopy BP, it is nevertheless a centralized algorithm. It would be interesting to investigate BP-type algorithms in

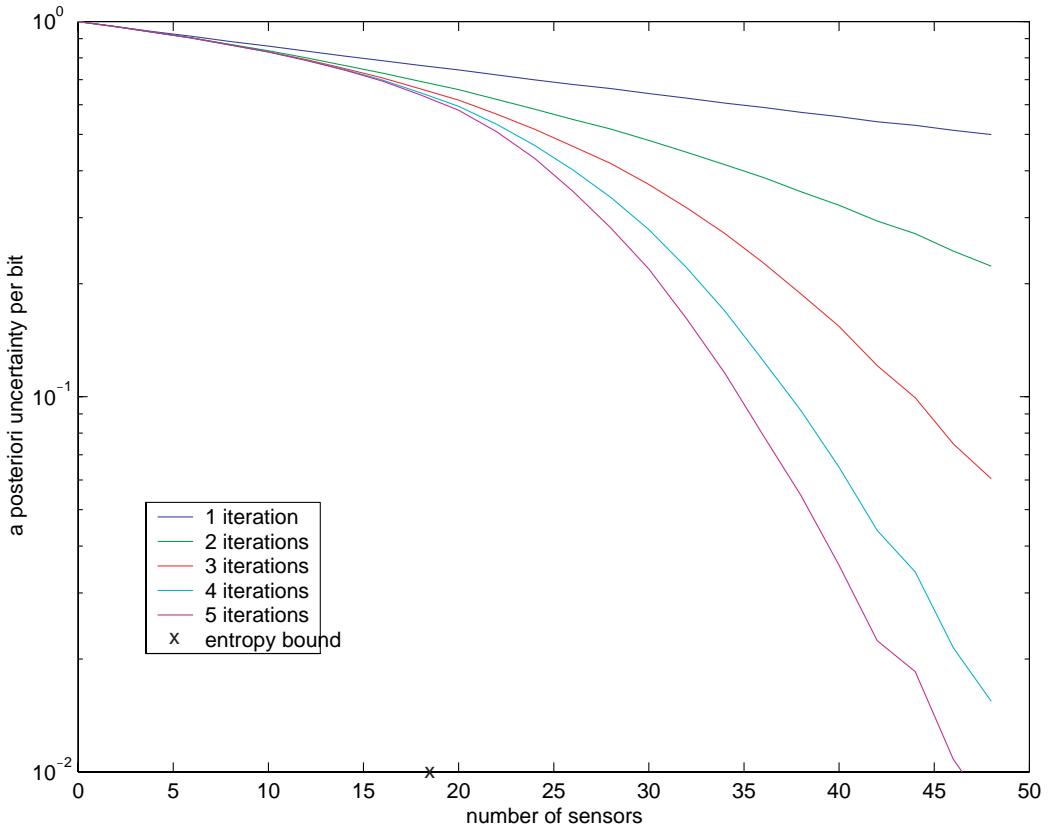


Figure 4. Loopy BP inference performance for an unknown environment \mathbf{X} with $n = 50$ components, where each robot senses $s = 10$ components of \mathbf{X} .

which the messages were actually passed among the sensors, without the need for a central base station.

- In a dynamic data-gathering situation, the robot observations will be updated periodically. In such a case, it would be important to know how rapidly loopy BP could react to these changes.
- In a distributed version of loopy BP, the transmitted messages may be subject to errors. How sensitive is loopy BP to occasional message errors?
- Finally, in Appendix 1, we remarked that Axiom 1 was reasonable if there is no notion of “closeness” of an incorrect prediction. It would be interesting to study penalty functions when there *is* an a priori notion of closeness.

Appendix A. Penalty Functions for Approximate Inference Algorithms.

In this appendix, we discuss possible criteria with which to judge an oracle which gives estimated probability distributions. Giving three well motivated axioms, we arrive at a single suitable measure, viz. the logarithm of the probability estimate of the event that actually occurs.

There are many possible applications for this theory besides the robot inference problem discussed above. In principle, it can be used to evaluate the accuracy of any source of probabilistic information. For example, we can compare the weathermen of different TV stations to see who is the most accurate. Or we could rank game-playing computer algorithms based on the accuracy of their predictions of the moves made in a large number of professional games.

Consider a discrete random variable X with (hidden) density function $p(x)$, for $x \in A = \{a_1, \dots, a_n\}$. Let $b(x)$ be an estimate of $p(x)$, called a “belief” about X , computed by an inference algorithm using some preliminary or indirect observations of X . We would like to measure the goodness of b as predictor of p . To this end, we assume we have a function $\Delta(b, x)$, which is interpreted as the penalty assessed the belief vector b if an experimental outcome is $X = x$. The penalty cannot depend on $p(x)$, because $p(x)$ may be unknown, or ill-defined. It may be unknown because it cannot be efficiently calculated (e.g., what is the probability that a number selected randomly from 1 to 10^{20} will have fewer than 6 prime factors?); it may be ill-defined because the experiment is in principle not repeatable (e.g., what will the weather be on December 18, 2002?).

We make the following three axiomatic assumptions about the penalty function Δ .

Axiom 1.

$$(6) \quad b_1(x) = b_2(x) \Rightarrow \Delta(b_1, x) = \Delta(b_2, x).$$

Axiom 1 says that if Algorithm 1 and Algorithm 2 both assign the same belief to the actual experimental outcome, then both algorithms are assessed the same penalty, regardless of the beliefs they may assign to the other possible outcomes. Thus

$$(7) \quad \Delta(b, x) = f(b(x))$$

for some function $f : [0, 1] \rightarrow R$. Axiom 1 seems reasonable if there is no notion of “closeness” of an incorrect prediction, in which case a prediction is either right or wrong.

Axiom 2.

For all possible $p(x)$ ’s and $b(x)$ ’s,

$$(8) \quad E_p(\Delta(p, X)) \leq E_p(\Delta(b, X)),$$

which because of (7) is

$$\sum_{x \in A} p(x)f(p(x)) \leq \sum_{x \in A} p(x)f(b(x)).$$

Axiom 2 requires that the belief about $p(x)$ with minimum average penalty must be $p(x)$ itself.

Axiom 3.

$$b(x) = 1 \Rightarrow \Delta(b, x) = 0.$$

Axiom 3 specifies that if the algorithm predicts the correct outcome with certainty, the penalty is 0.

It is easy to see that a penalty function defined by

$$\Delta(b, x) = \log \frac{1}{b(x)}$$

(for an arbitrary base of the logarithm) satisfies Axioms 1, 2, and 3. It is perhaps surprising that for $n \geq 3$, this is the only possibility:

Theorem. *If Axioms 1, 2, and 3 hold,⁴ and $n \geq 3$, then*

$$\Delta(b, x) = K \ln b(x)$$

for some constant $K \leq 0$.⁵

Proof: Let $A = \{a_1, \dots, a_n\}$, and let $p_i = p(a_i)$, $b_i = b(a_i)$, $i = 1, \dots, n$. By Axiom 1, $\Delta(b, x) = f(x)$, for some $f(x)$. Then Axiom 2 can be written as

$$\sum_{i=1}^n p_i f(p_i) \leq \sum_{i=1}^n p_i f(b_i),$$

⁴ In fact, our proof also requires that the function $f(x)$ in (7) have a continuous first derivative. We conjecture that continuity of $f(x)$ is sufficient for Theorem 1 to hold, however.

⁵ The constant K , in effect, determines the base of the logarithm. We could write an axiom that enforces say $K = -1$, but we prefer to leave the base of the logarithm unspecified, which is in keeping with information theory tradition. Also note that for $K = 0$ the penalty function is identically zero, which satisfies the axioms but is obviously useless.

or equivalently

$$(9) \quad \sum_{i=1}^n p_i (f(p_i) - f(b_i)) \leq 0.$$

Now choose $(p_i)_{i=1}^n \in S_n^o$, the interior of S_n , the n -dimensional simplex:

$$\begin{aligned} S_n &= \{p : p_1 + \cdots + p_n = 1, p_i \geq 0\} \\ S_n^o &= \{p : p_1 + \cdots + p_n = 1, p_i > 0\}. \end{aligned}$$

and choose $(\epsilon_1, \dots, \epsilon_n)$ so that

$$(10) \quad \sum_{i=1}^n \epsilon_i = 0.$$

Then for $|\lambda|$ sufficiently small, $(b_i)_{i=1}^n = (p_i + \lambda\epsilon_i)_{i=1}^n \in S_n$. But by the Mean Value Theorem,

$$f(p_i + \lambda\epsilon_i) = f(p_i) + \lambda\epsilon_i f'(p_i + \alpha\lambda\epsilon_i),$$

for some $0 < \alpha < 1$, so that (9) becomes

$$(11) \quad \lambda \sum_{i=1}^n p_i f'(p_i + \alpha\lambda\epsilon_i) \epsilon_i \geq 0.$$

But since (11) holds for arbitrarily small values of λ , both positive and negative, and $p_i + \alpha\lambda\epsilon_i \rightarrow p_i$ as $\lambda \rightarrow 0$, it must be true that⁶

$$(12) \quad \sum_{i=1}^n p_i f'(p_i) \epsilon_i = 0.$$

for all $(\epsilon_1, \dots, \epsilon_n)$ satisfying (10). By taking $\epsilon_1 = +1$, $\epsilon_i = -1$, and $\epsilon_j = 0$ for all $j \neq 1, i$, we see that

$$(13) \quad p_1 f'(p_1) = \cdots = p_n f'(p_n),$$

for all $\mathbf{p} \in S_n^o$.

We pause for a refreshing Lemma.

⁶ This is the step that requires $f'(x)$ to be continuous.

Lemma. Suppose $n \geq 3$. If $g(x)$ is a real-valued defined on $(0, 1)$ with the property that whenever $\mathbf{p} = (p_1, \dots, p_n) \in S_n^o$,

$$(14) \quad g(p_1) = \dots = g(p_n),$$

then there is a constant K such that $g(x) = K$ for all $x \in (0, 1)$.

Proof: By taking $\mathbf{p} = (x, 1-x, 0, \dots, 0)$, in (14), we find that

$$(15) \quad g(x) = g(1-x), \text{ for all } 0 < x < 1.$$

On the other hand, by taking $\mathbf{p} = (1/2, x, 1/2-x, 0, \dots, 0)$, we find that

$$(16) \quad g(x) = g(1/2), \text{ for } 0 < x < 1/2.$$

Combining (15) and (16), we see that

$$g(x) = g(1/2) = K, \text{ for } 0 < x < 1. \quad \blacksquare$$

Returning to the proof of Theorem 1, by combining (13) with the Lemma, we see that there is a constant K such that $pf'(p) = K$ for $0 \leq p \leq 1$, so that

$$f(p) = K \ln p + K'.$$

for a constant K' . But by Axiom 3, $f(1) = 0$, which forces $K' = 0$, i.e., $f(p) = K \ln p$ for some value of K . If $K \leq 0$, then by Jensen's inequality, i.e.,

$$E_p(\Delta(p, X)) = K \sum_i p_i \ln p_i \leq K \sum_i p_i \ln b_i = E_p(\Delta(b, X)),$$

by Jensen's inequality, so that Axiom 2 is satisfied, whereas if $K > 0$, the inequality goes the wrong way. Thus $f(p) = K \ln p$ for some $K \leq 0$, as asserted. \blacksquare

The quantity $E_p \Delta(b, X)$ has a nice information-theoretic interpretation, as shown by the following theorem.

Theorem 2. With $\Delta(b, x) = -\log b(x)$, we have

$$E_p(\Delta(b, x)) = H(p) + D(p \parallel b),$$

where $H(p)$ is the entropy of p ,

$$H(p) = - \sum_x p(x) \log p(x),$$

and $D(p||b)$ is the Kullback-Leibler distance (or relative entropy) between p and b , defined by [3, Section 2.3]

$$D(p \parallel b) = \sum_{x \in A} p(x) \log \frac{p(x)}{b(x)}.$$

Thus if $p(x)$ is the underlying density function, the minimum possible average penalty for any inference algorithm is $H(p)$, which is attained if the inference algorithm selects $b(x) = p(x)$ for all x .

Proof: We have

$$\begin{aligned} E_p(\Delta(b, x)) &= \sum_x p(x) \Delta(b, x) \\ &= - \sum_x p(x) \log b(x) \\ &= - \sum_x p(x) (\log p(x) + \log b(x) - \log p(x)) \\ &= - \sum_x p(x) \log p(x) - \sum_x p(x) (\log b(x) - \log p(x)) \\ &= H(p) + D(p \parallel b). \quad \blacksquare \end{aligned}$$

Appendix B. The Case $n = 2$.

To prove Theorem 1, we assumed that n , the number of possible outcomes, is ≥ 3 . This assumption is necessary, as the following Theorem (whose proof we omit) demonstrates.

Theorem 2. *Let $n = 2$. Then Axioms 1, 2, and 3 hold if and only if*

$$\Delta(b, x) = f(x),$$

where $f(x)$ is of the form

$$(17) \quad f(x) = \int_x^1 \frac{g(t)}{t} dt,$$

for some function $g(t)$ which satisfies

$$g(t) \geq 0 \text{ and } g(t) = g(1-t) \text{ for all } t \in (0, 1). \quad \blacksquare$$

Theorem 2 gives infinitely many essentially different penalty functions for $n = 2$. For example, if $g(t) = 1$ for all $t \in (0, 1)$, (17) gives

$$f(x) = -\ln x,$$

in accordance with Theorem 1. However, if $g(t) = 2t(1-t)$, then

$$f(x) = (1-x)^2,$$

which satisfies Axioms 1, 2, and 3 for $n = 2$ but violates Axiom 2 for $n \geq 3$. Despite this multitude of possible penalty functions for $n = 2$, because of Theorem 1, we feel it is unnatural to choose any function other than $f(x) = -\log(x)$, even if $n = 2$.

References.

1. S. M. Aji and R. J. McEliece, “The generalized distributive law,” *IEEE Trans. Inform. Theory*, vol. 46, no. 2 (March 2000), pp. 325–343.
2. S. M. Aji and R. J. McEliece, “The generalized distributive law and free energy minimization,” Proc. 2001 Allerton Conf.
3. T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: John Wiley and Sons, 1991.
4. G. D. Forney, Jr.. “The sum-product algorithm,”
5. B. J. Frey and D. J. C. MacKay, “A revolution: Belief propagation in graphs with cycles,” Proc. 1997 Neural Information Processing Systems Conference, in press.
6. R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, Mass.: MIT Press, 1963.
7. F. R. Kschischang and B. J. Frey, “Iterative decoding of compound codes by probability propagation in graphical models.” *IEEE J. Sel. Areas Comm.* vol. 16, no. 2 (Feb. 1998), pp. 219–230.
8. F. V. Jensen, *An Introduction to Bayesian Networks*. New York: Springer-Verlag, 1996.
9. J. H. Kim and J. Pearl, “A computational model for causal and diagnostic reasoning,” *Proc. 8th International Joint Conf. Artificial Intelligence* (1983), pp. 190–193.
10. R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, “Turbo decoding as an instance of Pearl’s ‘belief propagation’ algorithm.” *IEEE J. Sel. Areas Comm.* vol. 16, no. 2 (Feb. 1998), pp. 140–152.
11. J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmann, 1988.

12. G. R. Shafer and P. P. Shenoy, “Probability propagation,” *Ann. Math. Art. Intel.*, vol. 2 (1990), pp. 327–352.
13. Y. Weiss, “Correctness of local probability propagation in graphical models with loops,” submitted to *Neural Computation*, July 1998.
14. J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Generalized belief propagation,” pp. 689–695 in ”Advances in Neural Information Processing Systems 13,” eds. Todd K. Leen, Thomas G. Dietterich, and Volker Tresp.
15. J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Bethe free energy, Kikuchi approximations, and belief propagation algorithms,” available at www.merl.com/papers/TR2001-16/