Monte Carlo Decoding of LDPC Codes

Radford M. Neal

Dept. of Statistics and Dept. of Computer Science
University of Toronto

http://www.cs.utoronto.ca/~radford/
radford@stat.utoronto.ca

May 2001

Decoding as a Sampling Problem

We can view decoding as the problem of sampling from the distribution for codewords given received data.

A codeword, $\mathbf{x}$, is a vector of N bits.

Codewords satisfy $\mathbf{Hx} = \mathbf{0}$ (modulo 2), where $\mathbf{H}$ is the $M \times N$ parity check matrix.

The received data, $\mathbf{r}$, has probability $P(\mathbf{r}\,|\,\mathbf{x})$, determined by the channel characteristics.

Assuming codewords are equally likely *a priori*, the distribution for codewords given data is

$$P(\mathbf{x}\,|\,\mathbf{r}) \;\propto\; I\{\mathbf{Hx} = \mathbf{0}\}\; P(\mathbf{r}\,|\,\mathbf{x})$$

Sampling from this distribution for $\mathbf{x}$ would tell us which codewords are likely.

Usually, we expect/hope that this distribution is highly concentrated on a single value for $\mathbf{x}$.

Markov Chain Sampling

A difficult distribution, $\pi(\mathbf{x})$, can sometimes be sampled using an ergodic Markov chain that has π as its equilibrium distribution.

We define transition probabilities $T(\mathbf{x}, \mathbf{x}')$ for the chain to move to $\mathbf{x}'$ if it is currently at $\mathbf{x}$.

These transitions must leave π invariant:

$$\pi(\mathbf{x}') \;=\; \sum_{\mathbf{x}} \pi(\mathbf{x})\, T(\mathbf{x}, \mathbf{x}'), \quad \text{for all } \mathbf{x}'$$

We can construct T by applying in sequence several transitions, $T_1, \ldots, T_k$, that all leave π invariant.

We also need for T to produce an ergodic chain, which cannot be trapped in a subset of the state space.

For the method to be practical, the chain must converge to π reasonably quickly.

Gibbs Sampling / Heatbath Method

If the state is a vector, $\mathbf{x} = [x_1, \ldots, x_N]$, we can consider using the Gibbs sampling (aka, heatbath) method.

Let $T = T_1 \ldots T_N$, where transition T_i updates only x_i, leaving other components unchanged.

T_i randomly picks a new value for x_i from its conditional distribution (under π) given the current values of all other components. This is easily seen to leave π invariant.

If all the conditional distributions give non-zero probability to all possible values for x_i, the method will be ergodic — we could move from any $\mathbf{x}$ to any $\mathbf{x}'$ in one scan.

For the decoding problem, Gibbs sampling alone won't work – typically *all* the conditional distributions concentrate on a single value. Changing any single bit causes a parity error.

Auxiliary Variable Methods

Another approach is to introduce auxiliary variables, as in the Swendsen-Wang algorithm for the Ising system (generalized by Edwards and Sokal, 1988).

For a memoryless channel (eg, BSC or AWGN channel), we can write the distribution as a product of factors as follows:

$$P(\mathbf{x} \,|\, \mathbf{r}) \;\propto\; I\{(\mathbf{H}\mathbf{x} = 0\} \;\prod_{j=1}^{N} P(r_j \,|\, x_j)$$

We now introduce a real-valued vector $\mathbf{u}$ and define a joint distribution for $(\mathbf{x}, \mathbf{u})$ that produces the above distribution for $\mathbf{x}$:

$$P(\mathbf{x}, \mathbf{u} \,|\, \mathbf{r}) \;\propto\; I\{(\mathbf{H}\mathbf{x} = 0\} \;\prod_{j=1}^{N} I\{0 < u_j < P(r_j \,|\, x_j)\}$$

We now construct a Markov chain sampler that alternately samples for $\mathbf{u}$ given the current $\mathbf{x}$ and for $\mathbf{x}$ given the current $\mathbf{u}$. Both distributions will be uniform over some set.

Implementing Subcode Sampling

I call this auxiliary variable scheme "subcode sampling", because of how it is implemented.

Sampling for $\mathbf{u}$ given the current $\mathbf{x}$ is easy. Just independently pick each u_j uniformly from the interval $(0, P(r_j \,|\, x_j))$.

To sample for $\mathbf{x}$ given $\mathbf{u}$, we first note that bit x_j will be free to take on any value if

$$u_j < \min\Big[P(r_j \,|\, x_j = 0),\, P(r_j \,|\, x_j = 1)\Big]$$

Note that x_j will always be free if its previous value was the one with lower likelihood given r_j, and will otherwise be free with probability equal to the inverse likelihood ratio.

Once we know which bits are free, we sample uniformly from the set of all $\mathbf{x}$ that match the bits that aren't free and that satisfy the parity checks. This is done with matrix operations analogous to those used for encoding.

The Need for Annealing

As noted, Gibbs sampling for the decoding problem is not ergodic.

Subcode sampling is ergodic, since there is a positive probability that all bits will be free at once, allowing any codeword to be chosen.

But in practice, subcode sampling converges very slowly for interesting codes, since the number of free bits tends to be less than the number of parity checks, usually producing a degenerate subcode with only one codeword.

One way to try to get around this problem is via annealing — we approach the desired distribution in small steps, starting with a distribution that's easier to sample from.

Annealing the Likelihood

We can make subcode sampling easier by weakening the influence of the likelihood. This increases the number of free bits.

We vary a likelihood annealing parameter, λ, between 0 and 1 according to some schedule (ending at $\lambda = 1$). The distribution of $\mathbf{x}$ for a given λ is

$$P^{\lambda}(\mathbf{x}) \;\propto\; I\{\mathbf{H}\mathbf{x} = \mathbf{0}\} \;\prod_{j=1}^{N} L_j^{\lambda}(x_j)$$

Two possibilities for the functions L_j^{λ} are a simple power modification:

$$L_j^{\lambda}(x) \;=\; \Big[P(r_j \,|\, x_j = x)\Big]^{\lambda}$$

and a modification derived by inserting a "virtual BSC" with error probability $(1-\lambda)/2$ before the actual channel:

$$\begin{aligned} L_j^{\lambda}(x) \;&=\; \Big(1 - \tfrac{1-\lambda}{2}\Big)\, P(r_j \,|\, x_j = x) \\ &\quad +\; \tfrac{1-\lambda}{2}\, P(r_j \,|\, x_j = 1-x) \end{aligned}$$

Annealing Parity Check Constraints

Annealing the likelihood doesn't help Gibbs sampling, and doesn't help subcode sampling enough. We need to anneal the parity check constraints too.

We vary a parity check annealing parameter, ρ, between 0 and 1, along with λ, according to some joint schedule. The distribution of $\mathbf{x}$ for given λ and ρ is

$$P^{\lambda,\rho}(\mathbf{x}) \;\propto\; \prod_{i=1}^{M} R^{\rho}([\mathbf{H}\mathbf{x}]_i) \;\; \prod_{j=1}^{N} L_j^{\lambda}(x_j)$$

where the function R^{ρ} is defined as

$$R^{\rho}(y) \;=\; (1-y)\,\rho \;+\; (1-\rho)/2$$

Subcode sampling now uses an additional vector of auxiliary variables, $\mathbf{v}$, for the R^{ρ} factors. When sampling for $\mathbf{x}$ given $\mathbf{u}$ and $\mathbf{v}$, only the subset of parity checks for which $v_i > R^{\rho}(1)$ are active, allowing more movement.

Parallel Tempering

Annealing doesn't produce the exact answer, even asymptotically.

The parallel tempering method of Geyer (1991) fixes this, by sampling from the joint distribution for independent values of $\mathbf{x}$ at all values of the annealing parameters.

The Markov chain updating this joint state combines two sorts of updates:

1. Separate Gibbs sampling or subcode sampling updates of $\mathbf{x}$ for each value of the annealing parameters.
2. Proposals to swap the $\mathbf{x}$ vectors for nearby values of the annealing parameters, which we accept or reject in "Metropolis" fashion.

We can obtain a "phase diagram" of the state at all values of the annealing parameters.
Both 1D and 2D tempering are possible.

Results of a Full 2D Tempering Run

The following pictures show variation in some quantities as a function of the two annealing parameters, when decoding a single block.

Code: 2000 bits, 1000 checks, 3 checks/bit.

Channel: AWGN, $\sigma = 0.8$, $E_b/N_0 = 1.94$db.

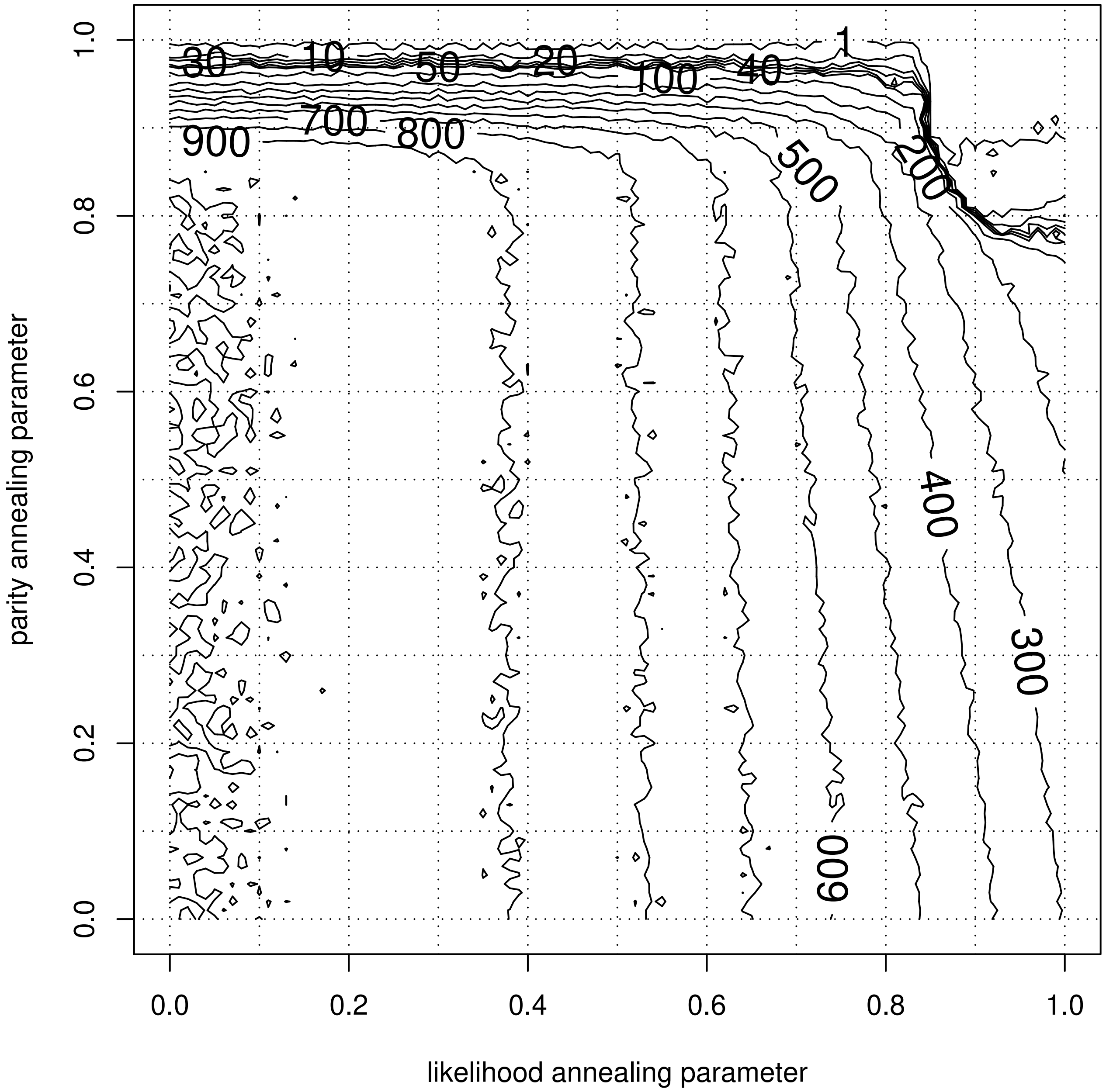
Tempering: Parity annealing and likelihood annealing (virtual BSC), all combinations of 101 levels of each, spaced from 0.00 to 1.00.

Operations: For each level combination, swaps with three earlier levels were considered, followed by 100 Gibbs sampling updates, and 10 subcode sampling updates.

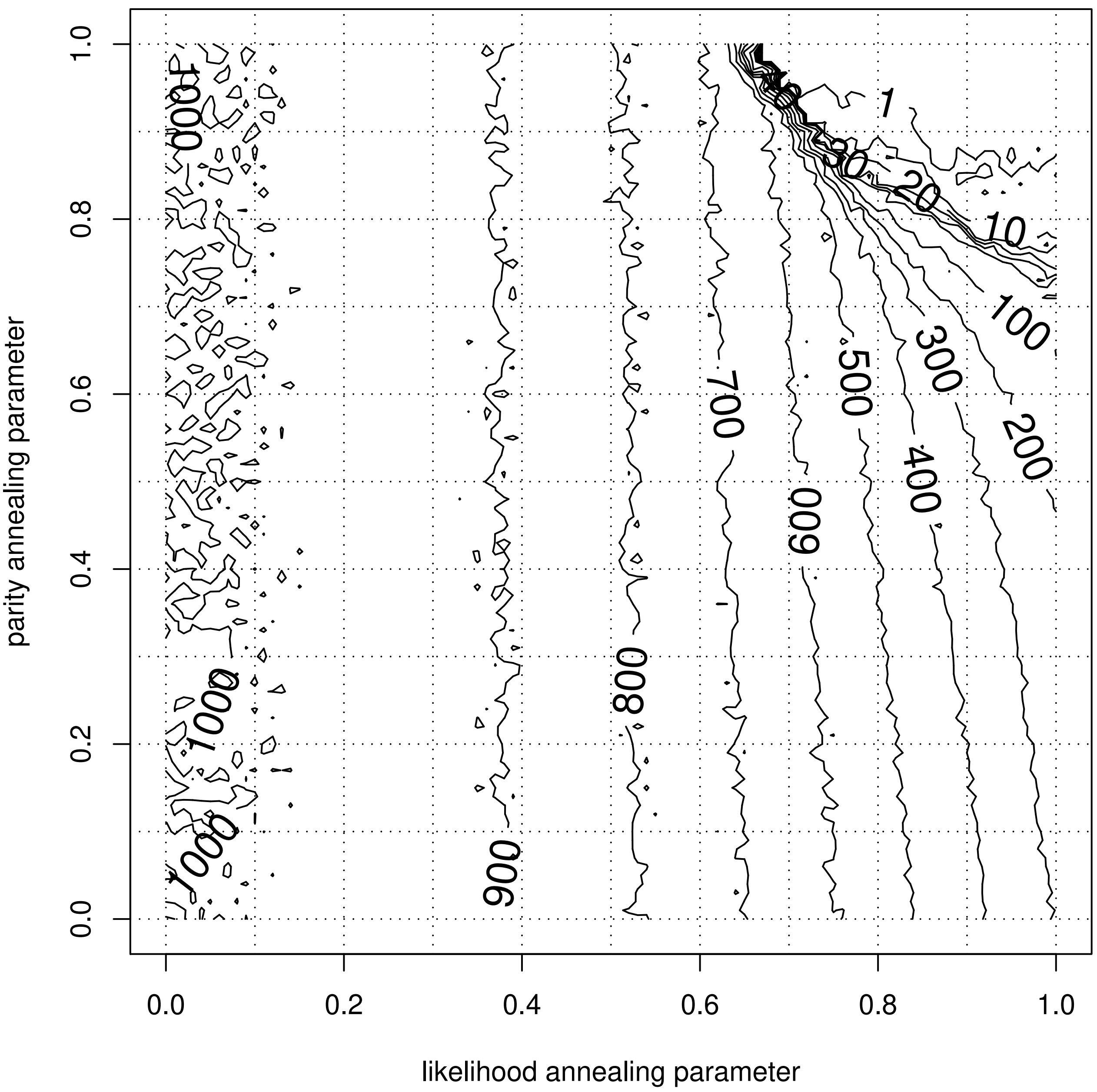
Run length: The initial annealing run was followed by 10 tempering scans. The values shown are averages over the last 7 scans.

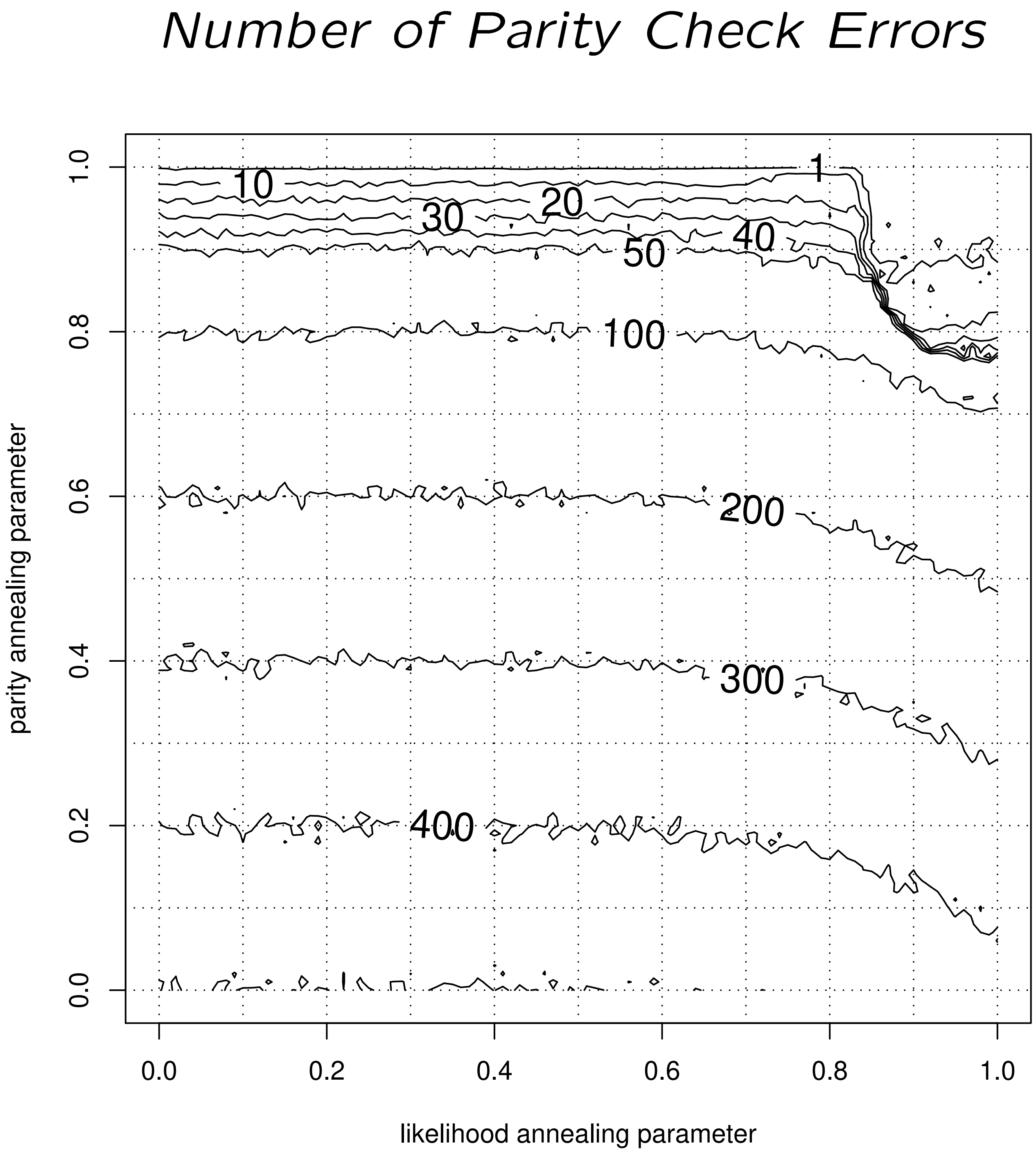
Result: Correct decoding found. Other aspects seem correct, but it's hard to be sure.

Number of Bits Altered by the 100 Gibbs Sampling Updates



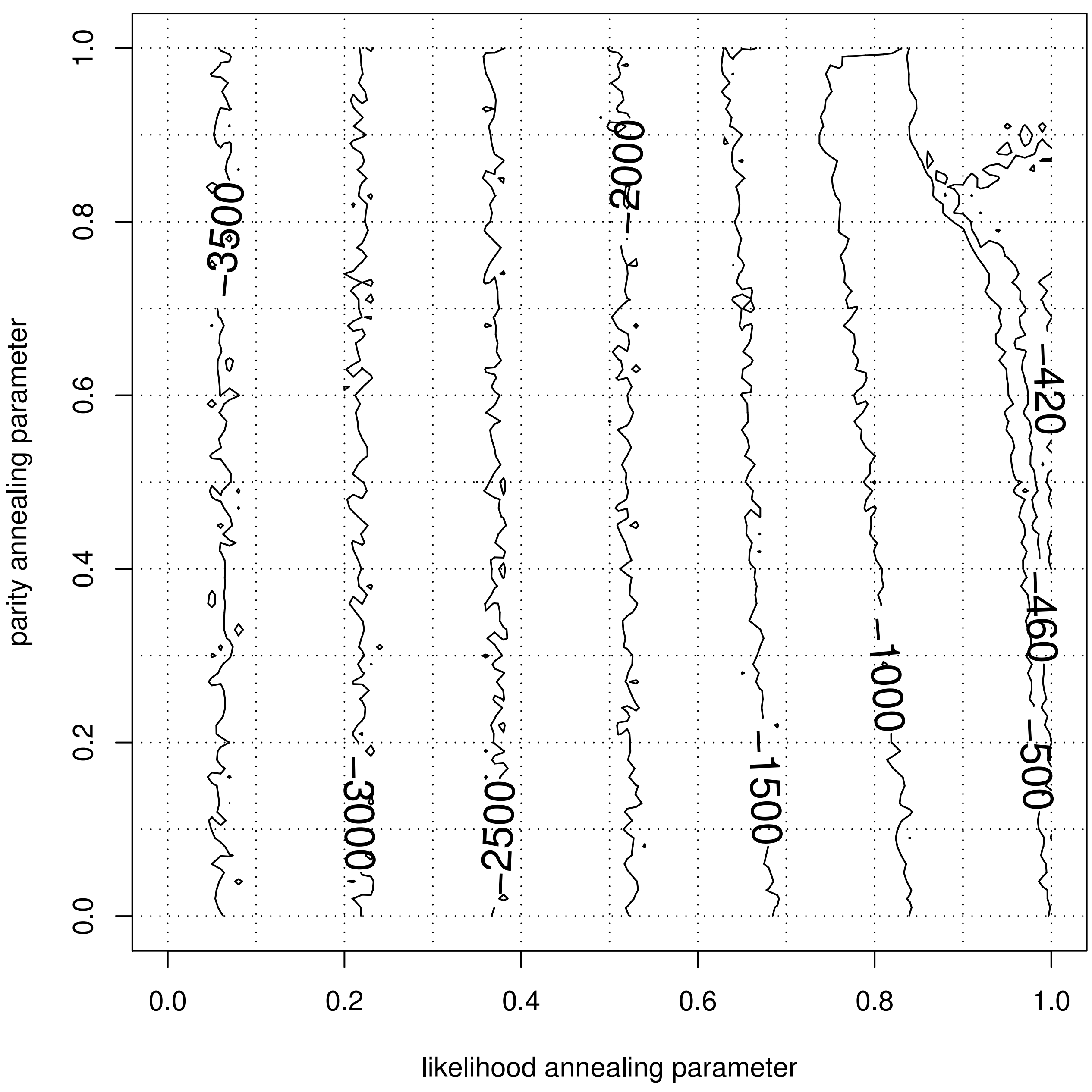
Number of Bits Altered by the 10 Subcode Sampling Updates





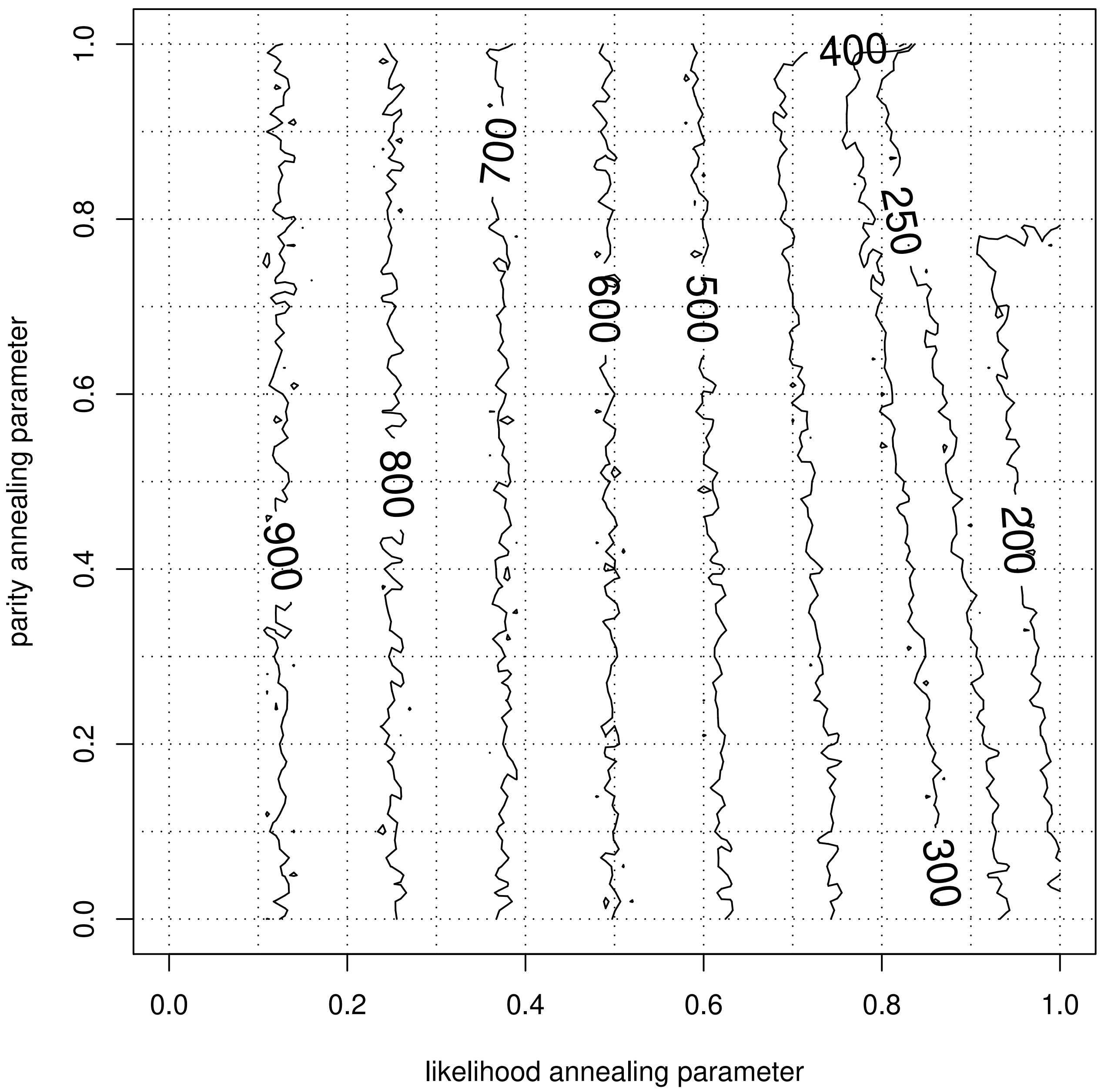
Number of Parity Check Errors
parity annealing parameter
likelihood annealing parameter
1
10
20
30
40
50
100
200
300
400
0.0
0.2
0.4
0.6
0.8
1.0

Log Likelihood



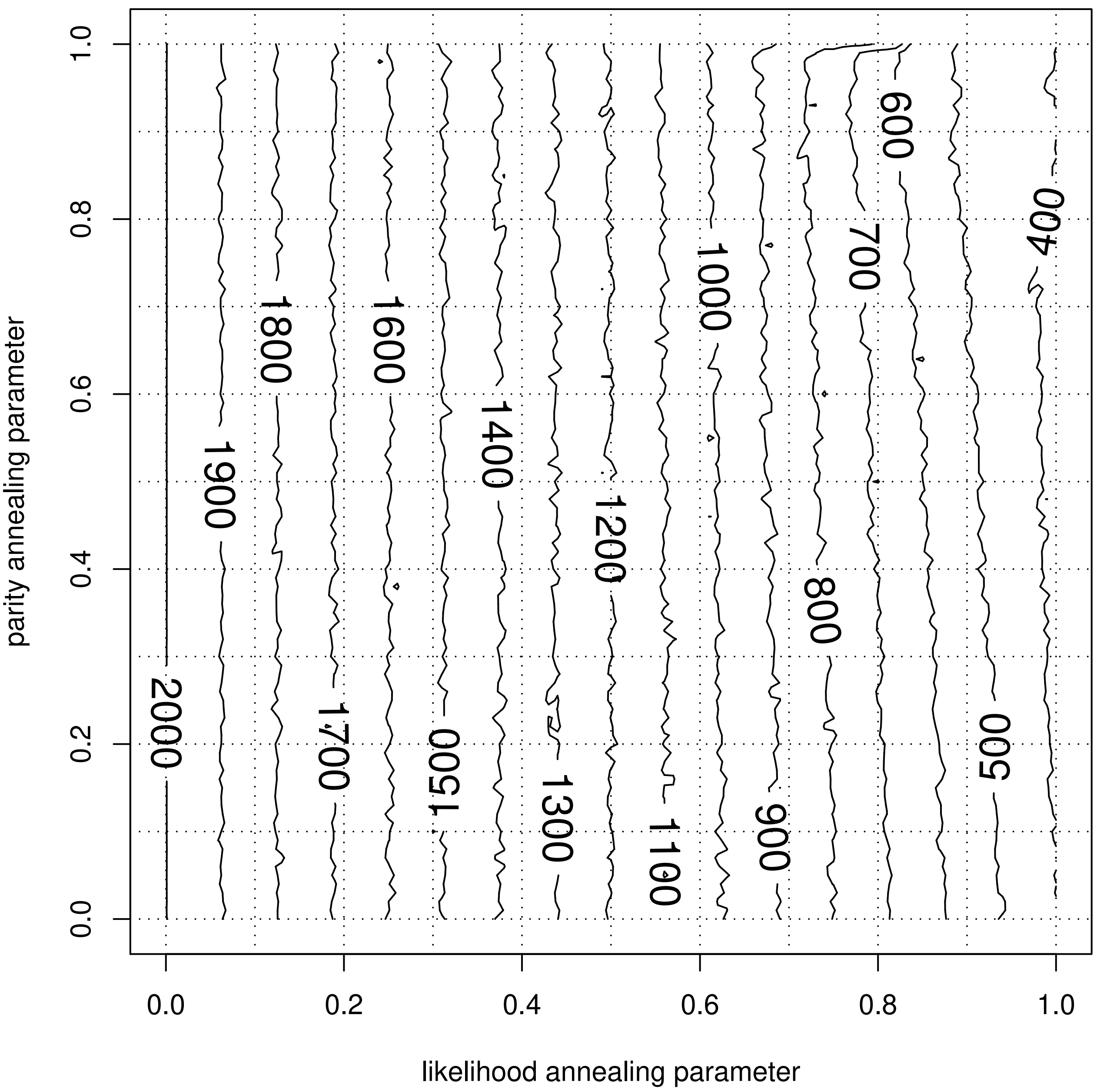
The log likelihood of the correct decoding is -461.

Number of Bits Differing from the Simple Bit-by-Bit Decoding

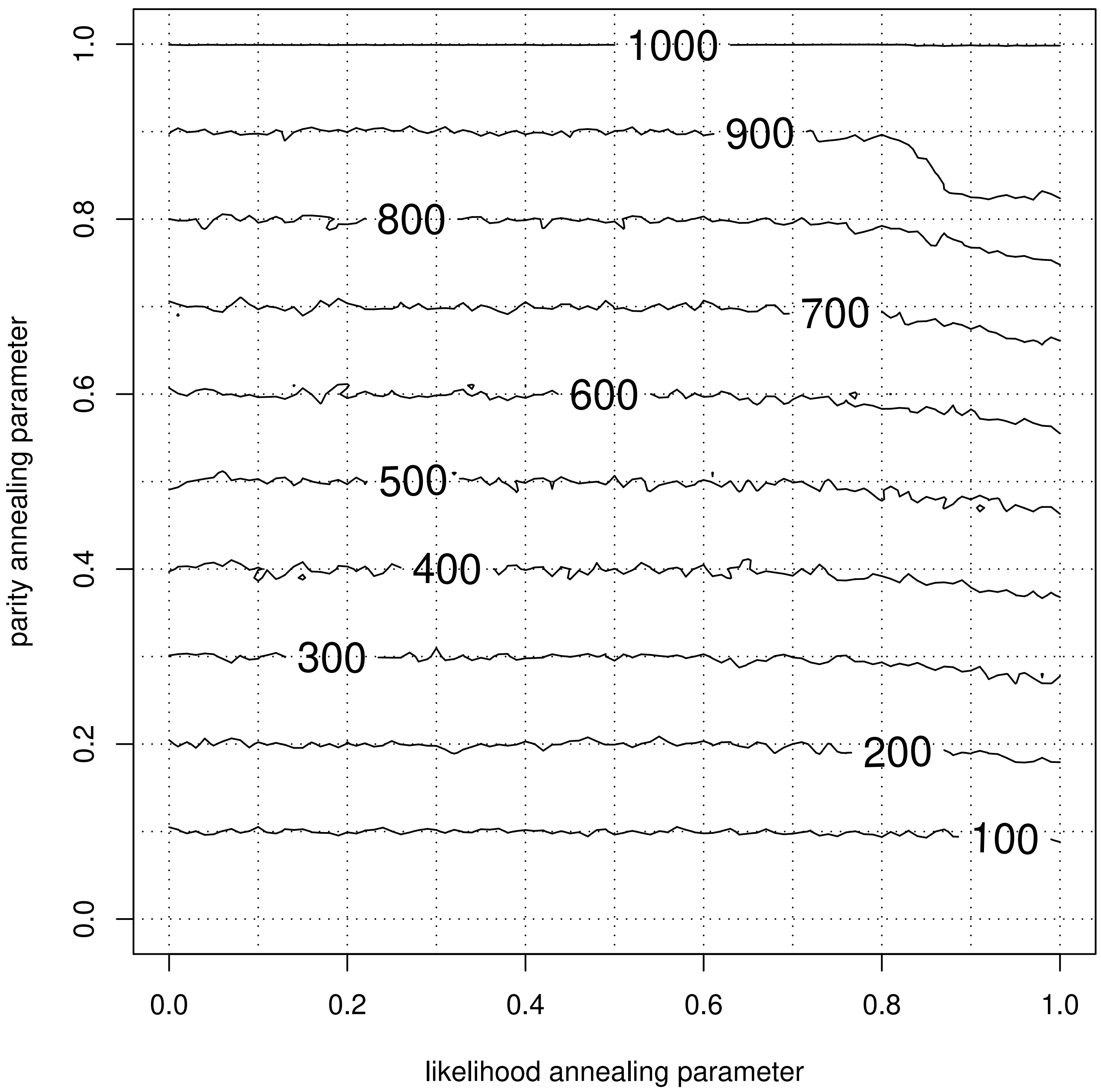


The correct decoding differs in 208 bits.

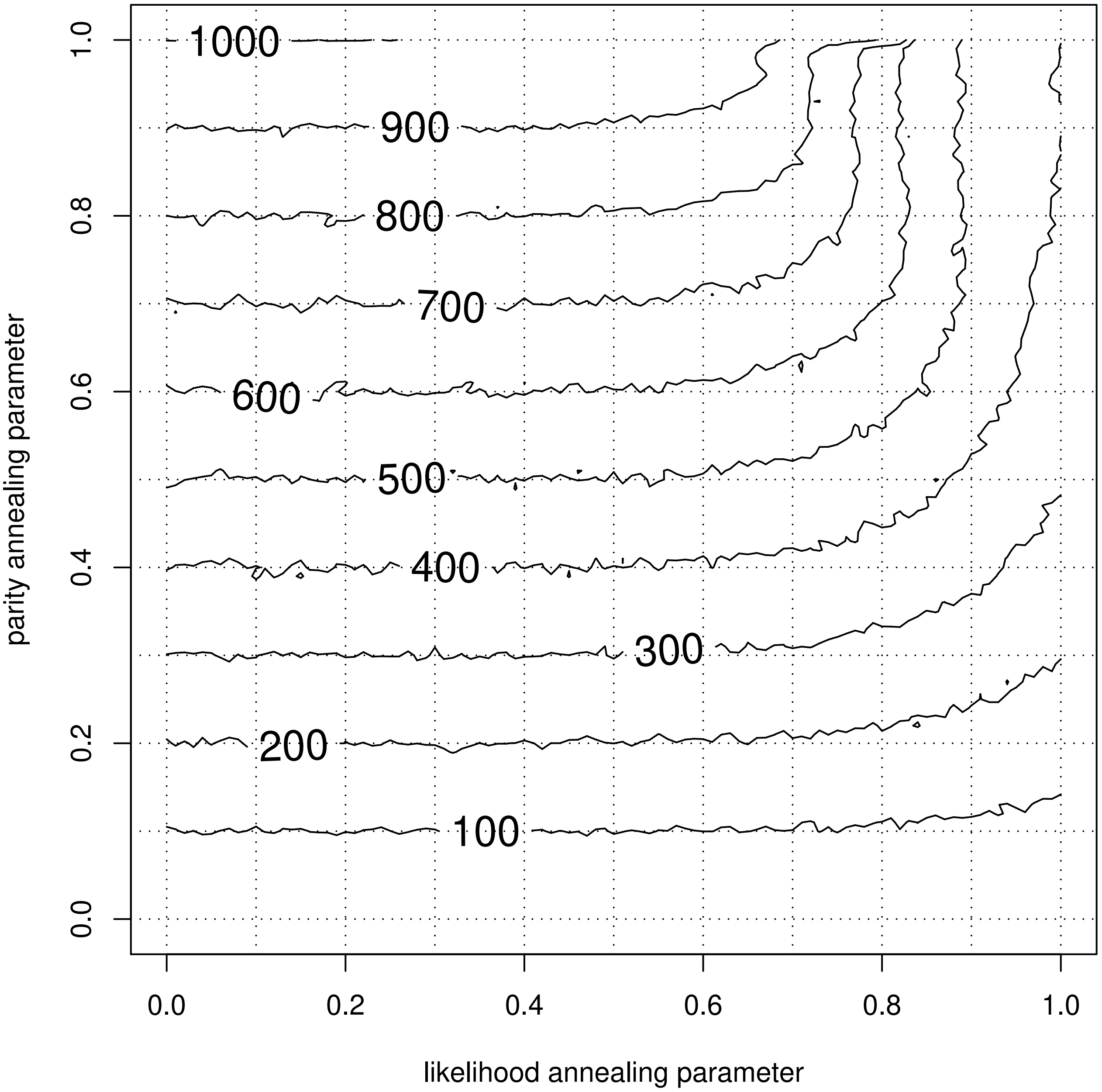
Number of Bits Free to Vary in a Subcode Sampling Update



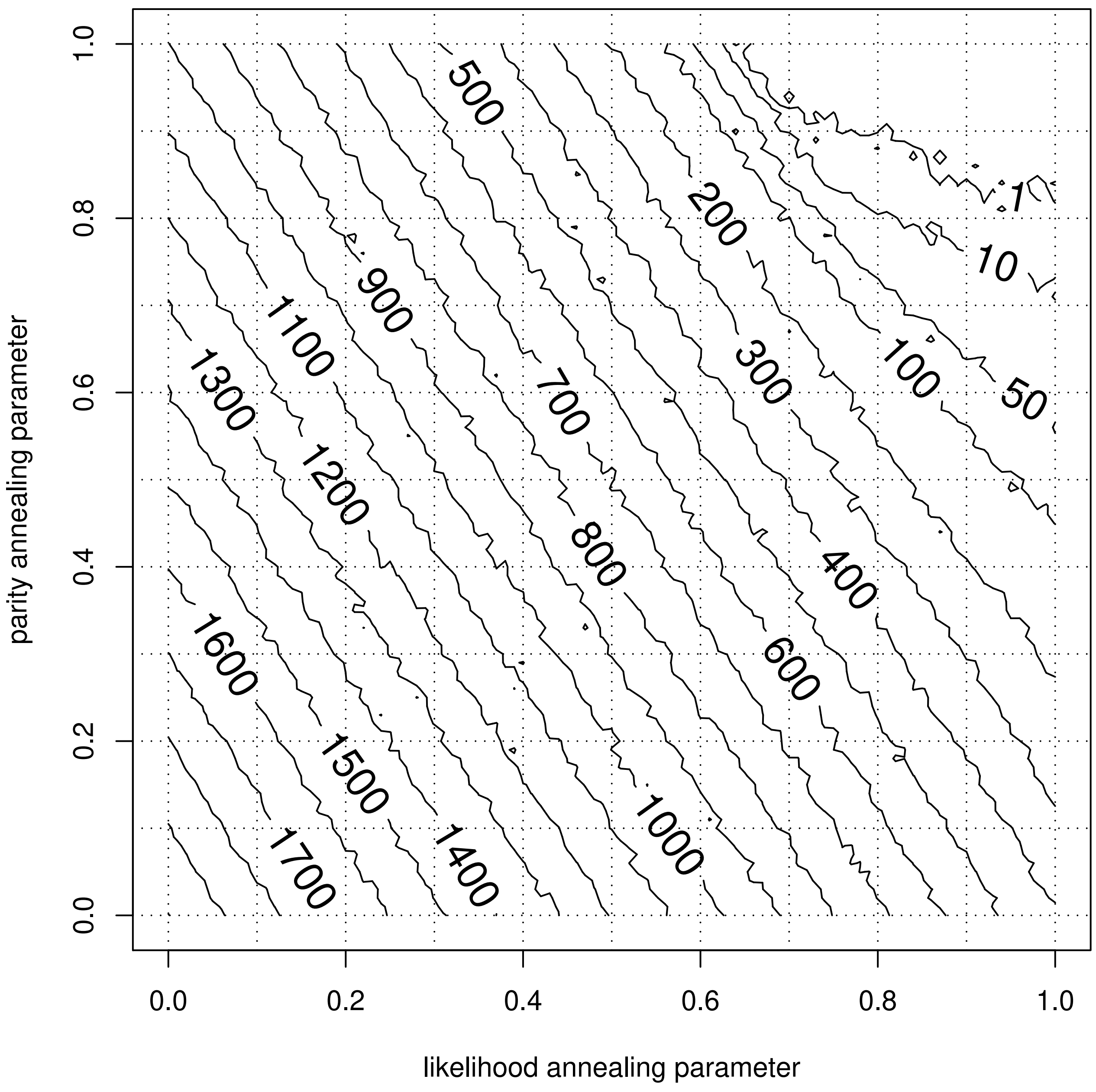
Number of Active Parity Checks in a Subcode Sampling Update



Number of Parity Checks Used in a Subcode Sampling Update



Number of Bits Set Randomly in a Subcode Sampling Update



Tests on a Rate 1/2 Code, BSC

Gibbs sampling alone with annealing or tempering for parity check constraints only was tried on a BSC with varying noise level.

Code: 1000 bits, 500 checks, 3 checks/bit.

Schedule: $\rho = 0.00, 0.01, \ldots, 0.99, 1.00$.

The table below shows the number of block errors in 10 transmissions.

BSC ERROR PROBABILITY:	0.060	0.065	0.070	0.075	0.080
bit flipping	10	10	10	10	10
prob. propagation 1000	0	0	0	1	4
annealed gibbs 2000	0	1	1	2	6
tempered gibbs 40 50	0	0	0	2	4
annealed gibbs 10000				1	4
tempered gibbs 100 100				1	4

Annealing with 2000 Gibbs sampling scans at each level took about 100 seconds per block; the corresponding tempering runs took from 3 to 50 seconds per block, more for higher noise levels.

Tests on a Rate 1/2 Code, AWGN

Code: 2000 bits, 1000 checks, 3 checks/bit.

Schedules:

S1: $\rho = 0.50, 0.51, \ldots, 0.99, 1.00$

S2: $(\lambda, \rho) = (0.750, 0.50), (0.755, 0.51), \ldots, (1.00, 1.00)$

The table below shows the number of block errors in 10 transmissions.

SIGMA:	0.70	0.75	0.80	0.85
Eb/N0:	3.10db	2.50db	1.94db	1.42db
bit flipping	10	10	10	10
prob. propagation 1000	0	0	0	1
tempered gibbs 50 100 S1	0	0	0	6
tempered gibbs 50 100 S2	0	0	0	7
tempered gibbs 200 100 S1				1
tempered subcode+bitflip S2	0	1		

The states in the first pass of subcode sampling were improved by simple bit flipping, as were the final states at $\lambda = \rho = 1$.

Subcode sampling was slow, taking 1 minute per block at $\sigma = 0.70$ and and 30 minutes per block at $\sigma = 0.75$.

Tests on a Rate 1/3 Code, AWGN

Code: 3000 bits, 2000 checks, 3 checks/bit.

Schedule: $\rho = 0.50, 0.51, \ldots, 0.99, 1.00$.

The table below shows the number of block errors in 10 transmissions.

SIGMA:	1.00	1.05
Eb/N0:	1.76db	1.34db
bit flipping	10	10
prob. propagation 1000	0	1
tempered gibbs 300 100	0	1

Time for the tempering runs was about 1 minute on averge for $\sigma = 1.00$ and 13 minutes on average for $\sigma = 1.05$.

Tests on a Rate 2/3 Code, AWGN

Code: 3000 bits, 1000 checks, 3 checks/bit.

Schedule: $\rho = 0.50, 0.51, \ldots, 0.99, 1.00$.

The table below shows the number of block errors in 10 transmissions.

SIGMA:	0.65	0.70
Eb/N0:	2.49db	1.85db
bit flipping	10	10
prob. propagation 1000	0	4
temper gibbs 300 100	0	6

The tempering runs took an average of 50 seconds per block when $\sigma = 0.65$, and an average of over an hour per block when $\sigma = 0.70$.

Questions Left to Answer

1) Subcode sampling seems generally slower than Gibbs sampling, but it is better for some values of the annealing parameters. Does this give any advantage in practice?

2) What is the best annealing/tempering schedule to use, and how closely spaced must the levels be?

3) Is tempering better than just doing multiple annealing runs, stopping when a good decoding is found?

4) Can one analyse theoretically when Gibbs and subcode sampling would be expected to work well?

Tentative Conclusions

1) Monte Carlo methods, even simple Gibbs sampling with tempering, can correct errors as well as probability propagation, albeit at much greater computational cost.

   Implication: Probability propagation does not exploit "global" aspects of the problem.

2) Monte Carlo methods do not work better than probability propagation.

   Implication: The flaws in probability propagation do not harm it much, and hence fixing these flaws will not help much.

3) Monte Carlo methods can be used to investigate the properties of the coding problem, which may perhaps help us find better decoding methods, or codes that are more easily decoded.