The Minesweepers' Bayesian Guide to Survival

Pál Ruján*

Fachbereich Physik und Institut für Chemie und Biologie des Meeres

Carl von Ossietzky Universität

Postfach 2503, D-26111 Oldenburg, Germany

Abstract

This paper presents some excerpts from the book with the same title and is a serious attempt at introducing statistical mechanics, phase transitions, coding and decision theoretical concepts to people who would rather play computer games.

Avant-propos

In 1993 W. H., the most intriguing student I ever had[1], asked me if a neural network could learn to play 'Minesweep'. As usual, his question was misplaced on several accounts:

1. I strongly discourage my students to play computer games or chat over the net instead of working on their project,

2. I tried several times to convince him that neural networks are not 'living' things but merely clever programs. People grown up on Star Trek (the Next generation) seem to be immune against such arguments.

3. Neural networks should not play games. That is the domain of AI and we are against it (even though they are not any longer against us...).

Anyhow, this work IS dedicated to W.H. Why this is so is the topic of a long Essay from which I can offer at this time only a few excerpts.

*e-mail: rujan@neuro.uni-oldenburg.de

[1] W. H. is not a product of my fantasy. I still think he could make millions would the software industry recognize his gift. W. H. does not find bugs. Bugs find W. H.

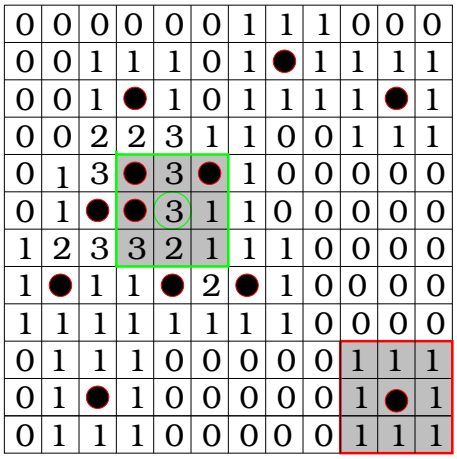

Figure 1: *Minesweep - coding the game board.*

1 The Minesweep Game

The computer game 'Minesweep' or 'Mines' is a puzzle which become so popular that Microsoft provides it with Windows'95 (the game seems to be bug-free). The main idea is to hide *at random* on a quadratic lattice a given number of bombs. The bomb-free squares (called slots) are *coded* by inscribing in them a number which equals the total number of bombs in their von Neumann neighbourhood. An example for such a coded lattice is shown in Fig. 1. The square in the center shows that in the neighbourhood of the encircled 3 there are in total 3 bombs.

After coding, the game board is covered up, except perhaps for the 0-slots and their neighbourhood. The player can choose sequentially any covered square and has then basically two options: either guess that there is bomb hidden under that square, or that there is no bomb. If one goes for a bomb but there is none, nothing dramatic happens - no additional information is won. Hence, this step is not more than the player's mark of a place where a bomb is eventually hidden. The real action is to find a a bomb-free square. If the guess is correct, the square is exposed and one gains some information about the bombs lurking in the neighbourhoods. If the guess is wrong, the Minesweeper dies. Most implementations do not allow for correcting this mistake and that is good so.

This game has a certain esthetic value, its principles are simple and intuitive. Playing requires logical thinking but escapes boredom. I even claim that not-so-sophisticated versions of this game are relevant to a wealth of physical and biological problems. Consider quenched lattice defects. A valid physical question is how to reconstruct their position from local distortions measured using the Mössbauer-effect, for example. Consider a population of cells, each exposing 'information' on their surface. How does the global amount of such information

change when the cells form a tissue? From a theoretical point of view, this problem is interesting because it sheds light on a very large class of 'natural codes' and their properties.

My hope is also that such a 'prepared' computer game might provide the right vehicle for bringing statistical physics, information, and coding theory closer to the real interests of our students.

2 Mathematical Formulation

Before answering the original question: can a neural network learn how to play this game ? (beyond which lurks the *really* interesting question of *how do* **we** *learn to do it ?*), let me consider first the optimal playing strategy. As we shall shortly see, the Bayesian Survival Guide is a Greek tragedy: beyond a certain critical bomb density everybody dies[2]. In order to discuss more formally our problem several definitions are introduced below.

1. Let us call MW (the Minesweepers' World) a graph $G = (E, V)$ consisting of vertices V connected by the edges E. Each node (vertex) can be coded to be either a bomb, a stone (not a bomb but no neighbourhood information available) or a slot. The distance between two nodes is defined as the smallest number of edges connecting them. Nodes at distance one are called nearest-neighbours, at distance two next-nearest-neighbours, etc. A slot contains information about the number of bombs in its first neighbourhood. According to this definitions the graph shown in Fig. 1 has no stones except at the surrounding boundaries and each node has 8 nearest-neighbours.

2. The slot-code is usually a deterministic, additive code (it counts the bombs in the next-neighbourhood). Such a code will be called a *noiseless* code. If the computer is drunk and counts sometimes more or less, the code is probabilistic and is called a *noisy* slot-code.

3. The bombs are generated identically and independently from a probability density p_B, called the *a priori* distribution. In the original game the probability to generate a bomb is p_0 for each node. There are many other possibilities for generating (correlated) distribution of bombs. The player might know or not the a priori bomb-probability distribution.

4. A game is called *sequential* if the player must uncover only one node in each step. Sometimes the protocol will allow (or require) the player to uncover a whole group of nodes before using again the gained information. Such games are called *parallel*. Similarly, if the player has access to the whole

[2] In the thermodynamic limit, with probability one.

MW the game is *global*. If only local information is available, the game is *local*.

5. Stepping on a mine is certainly a bad decision. However, as most of us know, life is a long series of bad decisions without fatal consequences[1]. Hence, uncovering an explosive node will be associated with a cost but the game will be allowed to go on. A strategy avoiding mines is minimizing the incurring cost. In other situations it might be more advantageous to find all mines as fast as possible. Such a strategy minimizes the playing time without regarding costs.

Once we have defined the lattice G, the main control parameter is the average density of bombs. Depending on the lattice structure, boundary conditions, and information loss due to noisy coding and the presence of stones, as the density of bombs increases the value of the 'surface' information displayed at the borders of uncovered vs. covered parts of the lattice decreases. The size of the unexplored field and the probability of finding a bomb rises, so that the chance of survival decreases exponentially.

3 Level 1: The Look-up Table approach

Let us consider first the case of small bomb density. The mean distance between two bombs exceeds then the range of nearest neighbourhoods and each bomb is practically surrounded by slots exposing number 1. Assuming noiseless coding, one single nearest-neighbour window (nnw) allows one to identify the bomb, as shown in Fig. 2. Therefore, at very small bomb-densities the code is redundant, the code rate is 1/8, or $1/\langle N_{nn}\rangle$ in general. $\langle N_{nn}\rangle$ is the number of nearest neighbours averaged over the MW. Hence, the code rate is defined as the inverse of the redundancy, the average number of times the *same* information is duplicated. At this stage, one could find the bombs even if the coding is probabilistic. One particularly simple implementation would be to take a majority vote on all nnw's surrounding an uncovered node. In fact, at low bomb-densities the Minesweeper Game implements a majority error-correcting code.

The situation changes as the bomb density increases. Fig. 3 shows the start configuration of the game once zero-slots have been exposed. The underlying bomb configuration obeys the rules of site-percolation. In the actual case each bomb has some 'interaction' range and the clusters seen in Fig. 3 (top) will lead to a percolating cluster well below p_c^{site}, the site-percolation critical probability. However, the information displayed on the cluster surface is redundant enough so that by moving a nnw-over all border slots solves the puzzle (see bottom of Fig. 3). The situation is more complex when the code is noisy but this case will be discussed later.

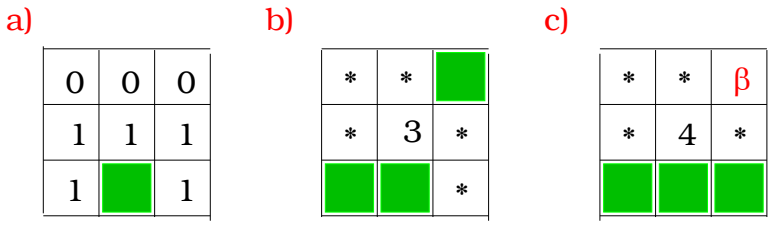

Figure 2: *The nearest-neighbour table as decoding tool. The sum of the uncovered nodes and defused mines must add up to the center sum. β denotes defused mines, $*$ any other exposed slot.*

I developed a version of the xminesweep program (©Ashley Roll) implementing the nearest-neighbour table strategy (and more). This program is by itself a rather good introduction to different data-structures and data-manipulation techniques most physicists are not aware of. In addition, new graphical tools, in particular the Tcl/Tk toolkit can be discussed and exercised at this point.

By increasing the lattice size one can obtain a rather good estimate of the bomb-density below which this strategy functions well ($p_0 \leq 0.2$). Most implementations of this game stay well below this bomb density. The game can be solved most of the time by applying a 3×3 look-up table. A neural network having 9 inputs and one output could then 'learn' (better said: load) this table given enough examples and that's all to it. Some of us will also learn to play the game in this way. If one plays long enough, however, one makes easily trivial mistakes and the outcome is merciless. Most of us will take that as a challenge, concentrate, and play again.

4 Level 2: The Enumeration Method

When further increasing the bomb-density p_0, we encounter situations where the simple look-up table strategy cannot solve the puzzle any longer (see Fig. 4).

Notice that such clusters tend to occur along the (free) boundaries of the board. This reflects the fact that the information useful for our decisions is concentrated at the *surface* of the uncovered clusters. This is a deeply philosophical observation: due to their hierarchical structure many biological systems must display information at the interface between *inside* and *outside* - at every level of organization. And isn't always the boundary between known and unknown the most fertile - and perhaps the only possible - ground for the scientific inquiry? [1].

Since in statistical physics we are interested on very large lattices (thermodynamic limit) I use periodic boundary conditions in that case. That will reduce

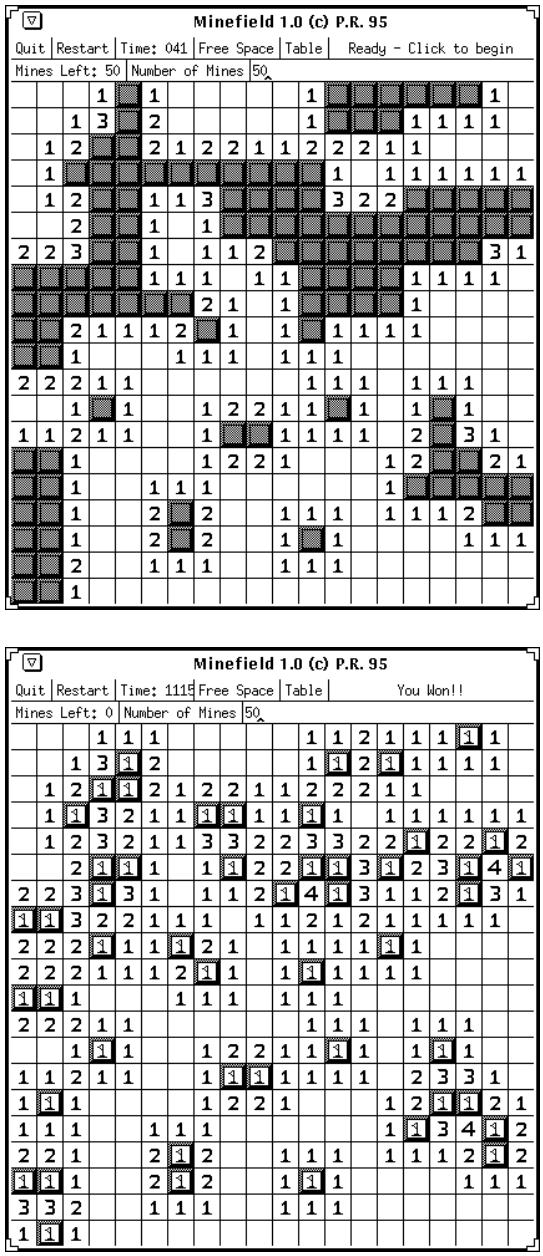

Figure 3: *A* 20×20 *square lattice at* $p_0 = 1/80$. *The uncovered site clusters percolate. However, applying the nearest-neighbour table sequentially solves the puzzle, as shown on the bottom picture.*

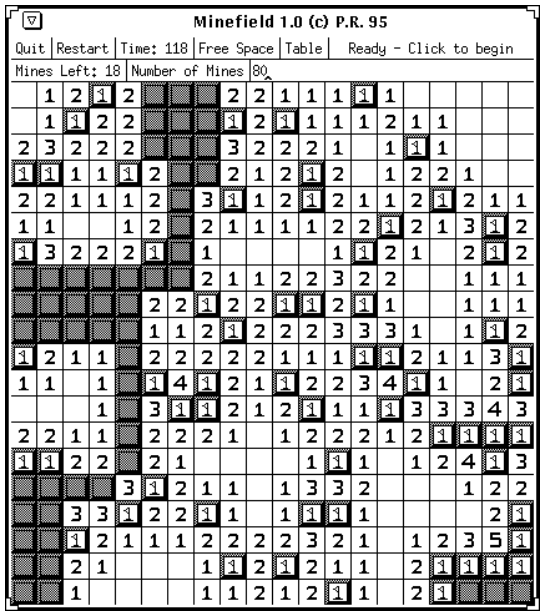

Figure 4: *Uncovered clusters which cannot be solved by the simple look-up table strategy.*

both finite size effects and boundary nucleation of such clusters.

The lack of succeeding with the table look-up strategy does not mean that there are no other ways to determine in a *deterministic* fashion where bombs are hidden! However, this requires now more complex strategies. The best one can do is to determine the probability of finding a bomb for each covered node and then act accordingly (choose the site with smallest probability as the next one to be turned up). From Fig. 4 it is clear that the uncovered nodes can be classified into 'surface' and 'bulk' nodes, depending on their distance from the explored world. For instance, the lower-right cluster is a bulk-cluster because it is isolated from the known world by a barrier of identified bombs. Furthermore, the graph of surface nodes might fall into disjoint components. Each such component will be called a 'surface'-cluster.

Level 2 playing requires to determine the probability of finding a mine for every node of the surface clusters. First, the calculation of the surface cluster graph itself is done using a simple graph-labeling algorithm. Again, I am not going *here* into details of how this is done. However, I think that such algorithmic issues are very important for both physicists or computer scientists. Good programs are both simple and optimal (no, not needless to say it: and bug-free).

Once a list of connected surface nodes is available, one must consider all bomb configurations compatible with the given 'surface' constraints. This problem is a generalization of the nearest-neighbour exclusion lattice-gas models in statisti-

cal mechanics. A direct approach has been developed, which first generates all *allowed* local configurations generated by a single nnw. These lists are merged together and the contradictory configurations are removed. In the end, one has an exact enumeration of all bomb configurations *allowed* by the boundary constraints. As different global configurations might involve a different total number of mines, the *a priori* probability with which a mine has been generated is used to weigh further the different possibilities. Finally, one computes for each node the number of times a bomb appears and the frequency of this event. This is the information used by the (sequential) decision process. If the node-probability is zero, there is no bomb at that site. If it is one, we found a bomb. In principle, this method generalizes the table-look up approach and is *almost* the optimal solution to the problem.

There are a few lessons here. First, one is better off with more constraints. When it comes to decision (or search), constraints are useful! It is simply impressing to see how the program can handle large surface-clusters. A simpler enumeration program generating first all possible 2^N configurations and then discarding the unrealizable ones simply fails. The basic principle of optimal algorithms: *Thou shall not search the emptiness* [1] - known as 'importance sampling' in physics - shines here in its full might.

Second, it is now easy to understand how different goals lead to different decision rules. If the goal is to minimize the number of steps in uncovering the whole field (and stepping on a bomb is not punished), the best strategy to follow is to uncover nodes whose bomb-probability is as close to 1/2 as possible! In this way one gains in every step the maximal amount of information (1 bit) about the uncovered world. As Alfred Rényi put it, asking a girl to marry you is exciting only when you do not have the faintest idea of what she is going to say.

Third, it is perhaps evident that once the bomb-density is high enough, we are left with many surface clusters and there is a definite chance to blow up at each choice. The survival probability will consist of the product of bomb-probabilities along a certain decision path. Even with great luck, our chance to survive tends to zero in the thermodynamic limit.

Table I shows some results concerning the total number of moves vs. the cost of solving the puzzle at a given bomb density. The enumeration method described here can be used also when designing parallel moves. In general, that would also require the calculation of node-node correlation functions. Although the enumeration method is *almost* optimal, it has its limitations. The number of generated configurations grows as a^N with the number N of nodes, and $1 < a < 2$. Hence, at higher bomb-densities one needs a lot of RAM and fast computers. Furthermore, the method does not handle the question of bulk spins (which are assumed to have the same bomb-probability). Last but not least, the gains made by restricting the search to a subset of all possible configurations disappears for noisy coding, when the constraints are probabilistic in nature.

Table I

The average survival probability, number of lives, and number of probabilistic decisions over different initial conditions for a square lattice with periodic boundary conditions

lin. size	strategy	bomb prob	survival prob	# lives	# questions
250	min p	0.18	0.035	2.6	7.6
250	$p \sim 0.5$	0.18	0.041	2.8	7.6
250	min p	0.20	0.0036	6.0	12.2
250	$p \sim 0.5$	0.20	0.0019	6.2	11.6
200	min p	0.22	5×10^{-3}	7.5	16.8
200	$p \sim 0.5$	0.22	10^{-4}	7.4	14.6
200	min p	0.25	10^{-10}	14	58.5
200	$p \sim 0.5$	0.25	10^{-13}	21.8	51

5 Level 3: The Transfer Matrix alias Dynamic Programming

The next level in playing mines is to implement the transfer matrix method. In many respects, the Mines Game is similar to a quenched, random spin system. On the other hand, the fact that the interesting constraints are all at the border of the uncovered cluster simplifies the problem.

For yet undisclosed reasons, let us consider noisy coding. At each uncovered spot we find now a discrete probability distribution for the total number of bombs in its nearest-neighbourhood being 0,1,...,N_{nn}. In fact, one can consider this constraint as the interaction energy between the involved surface 'spins' (alias nodes). Therefore, as the game proceeds, one has to solve dynamically defined statistical mechanical problems. The surface clusters are chain-like systems but can consist of several loops, dangling bonds, etc. Therefore, the method presented below can be used also for polymers and other similar physical problems.

As discussed in Level 2, our goal is to calculate the local magnetization of all spins (a bomb is a spin pointing up, a slot a spin pointing down). For one-dimensional regular chains there are known methods for performing this calculation. These methods could be applied here as well but since the interactions change dynamically, they require a very complex data structure. In what follows I present a variant of the truncation method.

Since the local magnetization is defined as

$$m_i = \frac{\sum_{\text{all spin configs}} s_i \prod(\text{all local constraints})}{\sum_{\text{all spin configs}} \prod(\text{all local constraints})} \qquad (1)$$

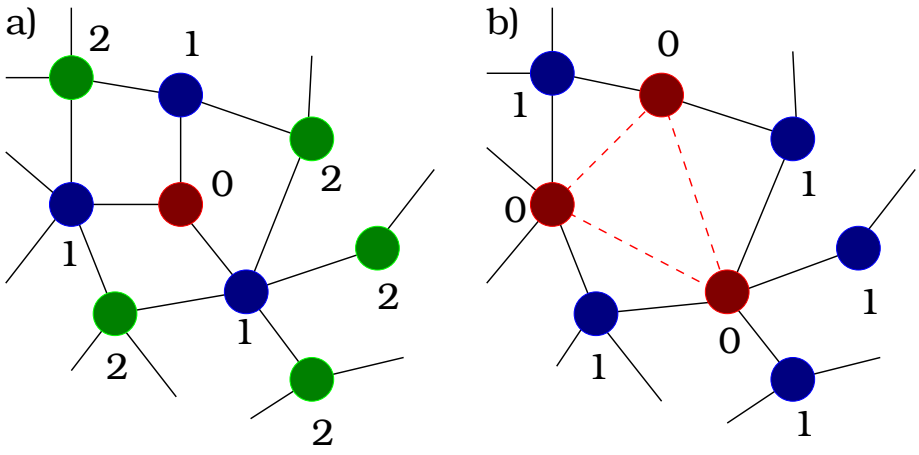

Figure 5: *a) Graph showing the local interaction structure of a surface cluster around spin s_0. The edges DO NOT correspond to two-spin interactions. All possible interactions are allowed between nearest-neighbours. After summing up spin s_0 one obtains the graph shown in b). The dashed lines represent new interactions generated by the summation. In the next step all spins denoted by a 0 are summed up, etc.*

we will carry with us the two sums, one for the normalization constant (the denominator = the partition function), and one for the numerator. The idea is to perform the sums sequentially starting at the 'center' spin s_i and then follow the interaction graph.

The first two steps are illustrated in Fig. 5. The 'interaction' between the spin 0 and the three nn-spins is described in general by a vector of 2^4 components $\vec{a} = (a_0, a_1, \ldots a_7)$ representing the coefficients of the orthogonal expansion in all spin products

$$f(s_0, s_1, s_2, s_3) = \sum_{n_i=\{0,1\}} a_{n_0+2n_1+4n_2+8n_3} s_0^{n_0} s_1^{n_1} s_2^{n_2} s_3^{n_3} \tag{2}$$

Summing up for the denominator Eq. (2) means that only those $\vec{a}$ components survive, which correspond to $n_0 = 0$. For the numerator one keeps only the $n_0 = 1$ components.

The next step is to *merge* the new interactions (dashed lines in Fig. 5b) with the already present ones (heavy lines in Fig. 5b). This involves a *product* of sums of the type shown in Eq. (2). It requires the allocation of a new vector (with 2^7 elements) and filling in its elements. Afterwards the 'old' interaction vectors are deallocated.

After the 'merge' operation we sum up all spins in the *actual* generation 0. This consist now of choosing the $n_0 = 0$ components for all three spins (in both

numerator and denominator sums).

After each summation step, one estimates the local magnetization as the ratio $\tilde{m}_i \approx a_0^{num}/a_0^{den}$. The loop ends if the change of the estimate remains within a given error range, there is no more memory available, or when the whole cluster is summed up. Apart from the bookkeeping and dynamic memory management problems, this method is well suited for very sparse graphs. The typical neighbourhood distance over which the procedure converges is related to the correlation length of the surface cluster.

What are the advantages of the transfer matrix method[3] over the enumeration technique of Level 2? This could be easily illustrated on a linear chain consisting of N nodes. The enumeration technique will need to compute something in the order of a^N steps and allocate vectors on the same order of magnitude. The transfer matrix method would need at most a vector of 2^4 (two 0-nodes, two 1-nodes) and, in general, about the same amount of computation as enumeration. For homogeneous interactions, of course, the computational part can be also strongly reduced to $O(N^3)$ (full diagonalization of the transfer matrix) or less (largest eigenvalues and eigenvectors of the transfer matrix).

The main advantage of this method, however, lies in its ability to deal with noisy coding. Assume now that we computed the local spin magnetizations (bomb probabilities) for all surface clusters. We can now use this information recursively by considering it as probabilistic constraints imposed on the first row of bulk clusters! Note that by using the local spin magnetizations instead of the full surface cluster 'interaction vector' we lose some information. However, the 'reduction' to local probabilistic constraints allows us to use the machinery developed above for calculating recursively the node bomb-probabilities for *all* covered spins. Without reduction one has again the problem of storing a potentially large vector.

Coming back to W. H.'s original question, we can now give an educated guess regarding the existence of neural networks good at playing 'Minesweep' (including perhaps our own). Assume a brain (natural or artificial) can allocate a certain 'attention window' (sensory memory) and 'memory buffer' (processing elements) when trying to solve the puzzle. If the attention 'window' is larger than the typical correlation length in the system, receiving enough examples might allow a complex network to 'learn' something similar to the local summation technique exposed in this Section. I seldom see people using paper and pencil when playing on a computer. I think most of the times we learn some simple rules or find some solution more 'nice' than others. Perhaps it would be worth designing psychophysical experiments testing the idea that we 'learn' such probabilistic tables when playing.

[3]This method was invented in physics in the 20's and has become known in computer science as the dynamic programming method in the 60's

6 Level 4: Modeling a priori Information

Until now we have assumed that we know how the slots are coded and how the bombs are distributed. We assumed that the bombs are randomly distributed with probability p_0. When computing the interaction vectors $\vec{a}$ (see Eg. (2)) one must 'merge' the external probabilistic constraints with an additional *a priori* probability term which takes the form of the product $\prod_{i=0}^{3}(1 + ys_i)$, with $y = 2p_0 - 1$. In this way all available information about the Minesweepers' World has been used, the decision is optimal.

However, we cannot always count on the willingness of our adversaries in making their strategies public. Consider the question of what is the distribution of 'natural' images, for instance. The class of such images is obviously very small compared with that of randomly generated images but is very difficult to define mathematically. As we have seen, in order to make optimal decisions we must eventually develop *models* for both the slot-coding process and the a priori distribution of bombs. In information theory language, one needs to have a good source distribution quantizer and a good channel-noise model. In the Minesweeper Game one finds a nice example for a code whose rate changes dynamically as a function of the bomb density. The information content of a given configuration can be also estimated by computing the entropy of the uncovered clusters, *eg* using the transfer matrix formalism.

Why Nature herself is not particularly hiding its coding strategies is one of the oldest questions tormenting *homo sapiens*. Maybe is this belief only an illusion. The illusion of a well adapted species.

Acknowledgments

This work has been partly supported by the SFB 517 'Neurokognition'. I thank Ido Kanter for inviting me to the Minerva Workshop. Due to the huge number of References, none is included.

References

[1] From the Author's 'Collection of Recycled Wisdom'.