Technical Note: Sharma-Mittal Generalized Entropy and 'Code Jar' Mastermind

Matthias Hofer*, Jonathan D. Nelson*

May 30, 2016

Abstract. We consider a variation of the well-known game of Mastermind to explore entropy-based approaches to the design of effective playing strategies. Our main goal is to conduct computer simulations to explore different entropy measures from a unified mathematical formalism, called the Sharma-Mittal generalized entropy framework. Ultimately, our goal is to translate potential theoretical findings to the domain of human psychology to investigate human intuitions about the informativeness of different queries when playing Mastermind.

Contents

*Max Planck Institute for Human Development, Berlin

1 The Mastermind Game

1.1 Standard version

In the original game, the *code maker* composes a hidden code c_h of fixed length ℓ using a combination of pegs (or marbles) of n different colors (repetitions are allowed). Here we will use the set of integers $\mathcal{A} = \{1, ..., n\}$ to represent the different-colored pegs. Call $\mathcal{E} = \{\mathcal{A}\}^\ell$ the set of all possible *combinations* of symbols from $\mathcal{A}$ of length ℓ, including the hidden code c_h.

Each round the *code breaker* takes a guess at the code and receives feedback about the number of pegs in the right position b (black token) and, additionally, the number of pegs that are of the right color but at the wrong place w (white token). Let c_i denote the code breaker's guess and $r_i = (b_i, w_i)$ the corresponding feedback, respectively, at round i.

Using the feedback provided after each guess, the code breaker's goal is to guess the hidden code c_h in as few rounds as possible. The game ends when $c_i = c_h$, which is equivalent to $r_i = (\ell, 0)$. In formal notation, a standard Mastermind game can be represented by the tuple

$$\mathcal{G}_{standard} = \Big(\mathcal{A}, \ell\Big),$$

with $n = |\mathcal{A}|$ and $\mathcal{E} = \{\mathcal{A}\}^\ell$. Note the constraints on receiving feedback:

- $b + w \leq \ell$
- if $b = \ell - 1$, then $w = 0$

Call $\mathcal{R}_\ell$ the set of all possible combinations $r = (b, w)$ for feedback. For $\ell = 3$, for instance:

$$\begin{aligned}\mathcal{R}_3 = \{&(0,0),\\ &(0,1),(0,2),(0,3),\\ &(1,1),(1,2),\\ &(2,0),\\ &(3,0)\}\end{aligned}$$

Or, more generally, $\mathcal{R}_\ell = \{(0,0),(0,1),...,(0,\ell),(1,1),...,(1,\ell-1),...,(\ell-1,0),(\ell,0)\}$, where $|\mathcal{R}_\ell| = |\mathcal{R}_{\ell-1}| + \ell$ with $|\mathcal{R}_1| = 3$.

1.2 Code jar version

Here we consider a variation of the standard game that involves introducing a *code jar* from which the hidden code is sampled (with replacement) by the code maker. A code jar J is an arbitrary set of pegs of different colors that (implicitly) defines a probability distribution $\mathscr{P}_J$ over the set of different-colored pegs $\mathscr{A}$. The code jar version is defined by the triple

$$\mathcal{G}_{codejar} = \Big(\mathscr{A}, \ell, J\Big),$$

with $n = |\mathscr{A}|$, $\mathscr{E} = \{\mathscr{A}\}^\ell$ and $\mathscr{P}_J(\mathscr{A})$ directly following from the quantities specified in $\mathcal{G}_{codejar}$. Rules for constraints on $\mathcal{R}_\ell$ are precisely as in the standard version (see ablove).

Note that the standard game can be seen as a special case of the code jar version when the code jar defines a uniform distribution over all possible symbols (pegs). This can be achieved by, for instance, setting $J = \{\{1\}, \{2\}, ...\{n\}\}$, where the code car contains exactly one peg for each of the n colors.

2 Important concepts

Mastermind is a dynamic constraint optimization problem. It is dynamic because the constraints determining the solution space are not known a priori and depend on the course of the game (i.e., the combinations played and the hidden code). One key notion is feasibility. The *feasible set* can be determined deductively by considering which combinations are consistent with the guesses made so far and the received feedback. We denote the feasible set after guess i with $\mathscr{F}_i$. Feasibility is a transient property. Initially, all combinations in $\mathscr{E}$ ($=\mathscr{F}_0$) are feasible. A combination looses its feasibility when it is inconsistent with any of the guesses made by the code breaker. The notions of consistency and feasibility (see below) hold for both the standard as well as the code jar version of the game.

2.1 Consistency

Consistency is a non-symmetric, reflexive function of two combinations c_q and c_r. We say that combination c_q is consistent with c_r, when $\mathrm{h}(c_q, c_r) = \mathrm{h}(c_r, c_h)$, where $\mathrm{h}(c_q, c_r)$ refers to the feedback when playing combination c_q, and when c_r is the hidden code. I.e., c_q is consistent with c_r iff c_q generates the same feedback

under c_r as hidden code than c_q generates under c_h. If a combination is consistent with all past guesses c_i, it is logically consistent to assume that it is the hidden combination.

2.2 Feasibility and the feasible set

All combinations that are consistent after guess i are part of the feasible set $\mathcal{F}_i$. The feasible set will shrink after each round unless no combination c_i is repeatedly used as a guess (in which case the feasible set stays the same).

3 Determining the usefulness of a combination

A generic playing strategy for Mastermind consists of procedures for (i) identifying the set of feasible combinations $\mathcal{F}_i$ (with $\mathcal{F}_0 = \mathcal{E}$), where prior feedback is used to determine which combinations are still viable and which not; and (ii) picking the combination c_{i+1} for the next guess. While for the standard version, the usefulness of a combination depends on how many combinations it is expected to exclude in the next step, this does not necessarily hold for the probabilistic, code jar version of the game.

3.1 Graphical intuition

Since not all combinations in $\mathcal{E}$ are necessarily equally likely, we have to use a different utility function to measure how much we prefer one feasible set over another. For example, ending up with a set $\mathcal{F}^a$ with $P(\mathcal{F}^a) = \{0.6, 0.1, 0.1, 0.1, 0.1\}$ is (arguably) better than ending up with $\mathcal{F}^b$ with $P(\mathcal{F}^b) = \{0.2, 0.2, 0.2, 0.2, 0.2\}$, if we had to take a guess at the hidden code. We will use the notion of entropy to quantify this difference, i.e., to measure the uncertainty of a probability distribution. Building on this, our algorithm (introduced below) chooses the combination that provides the highest expected reduction in entropy. A graphical intuition on how to compute the usefulness of a combination c is provided in Figure 1 and a formal description follows in the next section.

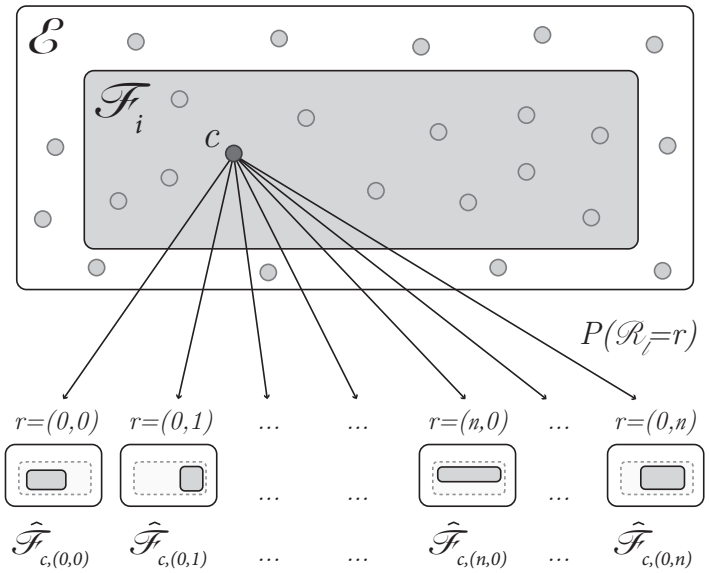

Figure 1: Graphical intuition for computing the usefulness of a query. After guess i, the current feasible set $\mathcal{F}_i$ is a subset of the set $\mathcal{E}$, containing all possible combinations. We can evaluate the usefulness of any combination c (not necessarily only combinations in $\mathcal{F}_i$) by considering all possible combinations of feedback r when playing c and their respective probabilities $P(\mathcal{R}_\ell = r)$. We can also compute the ensuing feasible sets $P(\hat{\mathcal{F}}_{c,r})$, the possible worlds we could end up in after receiving feedback r when playing combination c. The usefulness of query c is then defined as the probability-weighted sum of $H(\mathcal{F}_{c,r})$, where H measures the entropy of the set.

3.2 Formal treatment

Let us denote the expected utility (or usefulness) of playing combination c in the next round with $eu(c)$. The generic formula for computing $eu(c)$ is

$$eu(c) = \sum_{r}^{\mathcal{R}_\ell} P(\mathcal{R}_\ell = r) \cdot H^{SM_{r,t}}\Big(P(\hat{\mathcal{F}}_{c,r})\Big), \tag{1}$$

that is, for each possible feedback r, the product of the probability of receiving feedback r, $P(\mathcal{R}_\ell = r)$, and the entropy associated with the ensuing feasible set $\hat{\mathcal{F}}_{c,r}$ when playing combination c and receiving feedback r. We will look at each of these quantities and how to compute them below.

3.2.1 Probability of feedback

The general idea behind the formula above is to consider all possible outcomes (feedback) when playing combination c and to ask how much we 'like' receiving a particular feedback. Let us first consider how to compute the probability of different feedback outcomes.

When taking into account that same combinations might already have been eliminated from the feasible set, the potential feedback will be a subset of $\mathcal{R}_\ell$. To compute $P(\mathcal{R}_\ell = r)$, the probability of receiving some feedback r from set $\mathcal{R}_\ell$, we simply look at all the combinations in $\mathcal{F}_i$ that would lead to feedback r when they were the hidden code. For some combination c,

$$P(\mathcal{R}_\ell = r) = \frac{\sum_{c_j}^{\mathcal{F}} P(c_j) \cdot \gamma_r(c, c_j)}{\sum_{c_i}^{\mathcal{F}} \sum_{c_j}^{\mathcal{F}} P(c_j) \cdot \gamma_r(c_i, c_j)},$$

where $\gamma_r(c, c_j) = 1$ if $h(c, c_j) = r$ and 0 otherwise to ensure we only include combinations that are feasible with c under r. The probability of any combination of pegs or marbles $c = m_1 m_2 ... m_\ell$ can be computed as

$$P(c = m_1 m_2 ... m_\ell) = \mathscr{P}_J(\mathcal{A} = m_1) \cdot \mathscr{P}_J(\mathcal{A} = m_2) \cdot ... \cdot \mathscr{P}_J(\mathcal{A} = m_\ell) \qquad (2)$$

for the code jar version, and $P(c = m_1 m_2 ... m_\ell) = (\frac{1}{n})^l$ for the standard version.

3.2.2 Determining hypothetical feasible sets

The other term of Equation 1 is $H^{SM_{r,t}}(P(\hat{\mathcal{F}}_{c,r}))$, which requires us to compute hypothtical feasible sets $\hat{\mathcal{F}}_{c,r}$ (also see Figure 1). Given the current feasible set $\mathcal{F}_i$, a combination c_i we want to evaluate, and hypothetical feedback r, we simply need to exclude all combinations $c_j \in \mathcal{F}_i$ for which $h(c_j, c_i) \neq r$; that is, all combinations c_j that are not consistent with feedback r.

3.2.3 Sharma-Mittal generalized entropy

Lastly, in order to measure how much we 'like' one such hypothetical set, i.e., to assign a utility to a feasible set $\mathcal{F}$, we use the notion of entropy. We use the

generic Sharma-Mittal entropy framework to compute the entropy of a probability distribution defined over set $\mathcal{F}$, $P_{\mathcal{F}}(c)$. For each combination $c \in \mathcal{F}$

$$P_{\mathcal{F}}(c) = \frac{P(c)}{\sum_{c_j}^{\mathcal{F}} P(c_j)},$$

where the nominator $P(c)$ is computed according to Equation 3.2.1 and the denominator is a normalization term. The Sharma-Mittal entropy of set $\mathcal{F}$ can then be computed as

$$H^{SM_{(r,t)}}(\mathcal{F}) = \frac{1}{t-1}\left[1 - \left(\sum_{c}^{\mathcal{F}} P_{\mathcal{F}}(c)^r\right)^{\frac{t-1}{r-1}}\right],$$

where $0 < r, t < \infty$.