拼字游戏中的优化问题及解答

前言

拼字游戏是在西方的青少年中非常流行的游戏形式，游戏开始前通常会指定一本参考词典，所有在该词典中的词汇及其变形均被接纳，由主持人选中一个单词并给出特定信息，参赛者根据信息回答出主持人选中的单词。很多人为了在拼字游戏中取得成绩，甚至每天花费七八个小时去记忆那些冷门的英语单词，其中甚至包括绝大多数成年都闻所未闻的偏词[1]。其积极意义在于增进处于学语期间的青少年对单词的熟悉程度，帮助其理解单词构造，单就这一点而言，许多英语教育者已经将其运用到了课题上，在温习所学的同时活跃课堂气氛。

广义上讲的拼字游戏包含很多类型，包括广为人知的吊死鬼、拼字数独甚至传统的单词听写。本报告将讨论其中的一个特殊形式，并给出一个优秀的游戏策略及从优化方法角度讨论其数学原理。

问题叙述

本报告讨论的拼字游戏规则如下：

（1）进行游戏需要 1 位主持人与 1 位参加者。

（2）游戏开始前会公开一系列参考单词，称为词典，主持人选取其中一个单词，称为秘密，并告知参加者单词长度。

（3）参加者从词典中选取一个单词提交给主持人，该单词称为一个猜测，主持人在收到参加者的猜测后需要回答该猜测对秘密的命中程度，即猜测与秘密在几个位置上具有相同的字母。

（4）重复进行步骤（3）十次。

（5）参加者选取的十个猜测中，如果包含秘密，则参加者胜，反之，则主持人胜。

本报告将站在参加者的立场上，希望能够找到一个有尽可能大胜率的游戏策略。

问题公式化

本节内容会介绍我们对该问题的建模过程、在本报告中将使用到的数学符号及其含义。首先对于给定的单词长度 L，令 N_L 表示词典中包含的长度为 L 的单词数量，则可以将词典 D 建模为 $N_L \times L$ 的由整数构成的矩阵，即 $D \in Z^{N_L \times L}$，猜测 a 与秘密 s 被建模为在 L 维整数空间上的向量，即 $a \in Z^L, s \in Z^L$，同时显然的，存在 i 使得 $D_i = a$，D_i 表示词典 D 的第 i 个行向量，存在 j 使得 $D_j = s$，D_j 表示词典 D 的第 j 个行向量。此时，根据定义，问题叙述节的步骤（3）中所涉及到的命中程度 Hit 可以被建模为猜测 a 与秘密 s 之差的零范数相关的函数，即：

$$Hit = L - \| s - a \|_0 。 \tag{1}$$

在实际工程中我们对单词的编码方法是逐字母独立进行的，每个字母被唯一编码为 $[0,25]$ 中的整数。但为图叙述简便，在本报告的其他内容中不会针对此实现进行规约。

我们的目标则可以认为是利用尽可能少的猜测 a 求解出秘密 s，即联立尽可能少的方程，使得关于秘密 s 的方程组

$$\begin{cases} Hit_1 = L - \| s - a_1 \|_0 \\ \vdots \\ Hit_k = L - \| s - a_k \|_0 \end{cases} 。 \tag{2}$$

中 a_k 表示第 k 个猜测，Hit_k 为其对应的命中程度。

除此之外我们还定义了关于给定词典 D 的函数

$$e(D) = \sum_{i}^{N_L} \sum_{j}^{N_L} \left\| D_i - D_j \right\|_0 \tag{3}$$

我们认为上式可以用来在一定程度上衡量词典内单词的数量与单词之间的相似程度，而通常的，基于一个单词数量较少，单词之间相似程度较高的词典进行的拼字游戏会具有更低的难度。

算法流程与关键难点

算法的基础框架及可行性分析

为了达成上述目的，我们分析了拼字游戏的一般思路，总结了参加者方的思维逻辑，并基于此搭建了我们算法设计的基础框架，如图 1 所示。我们设计每个猜测 a_k 会基于当时的词典 D_k，并从其中选取，在得到 a_k 的命中程度 Hit_k 后，对 D_k 中的所有行向量 D_{ki} 进行测试，所有不满足

$$L - \left\| D_{ki} - a_k \right\|_0 = Hit_k \tag{4}$$

的行向量被从 D_k 中删除，这样设计的好处在于可以使得程序能在有限步内停止并输出 s，我们称满足这个特性的算法是可行的。由于至少 $D_{ki} = a_k$ 时式（4）不被满足（否则即有 $s = a_k$，输出即可），故在每个迭代过程中词典的规模都得到了缩小且对于所有词典，$s \in D_k$ 都成立，故显然程序能在有限步内停止并输出 s，即算法可行，关键点在于如何设计猜测 a_k 使得迭代的步数尽可能少。

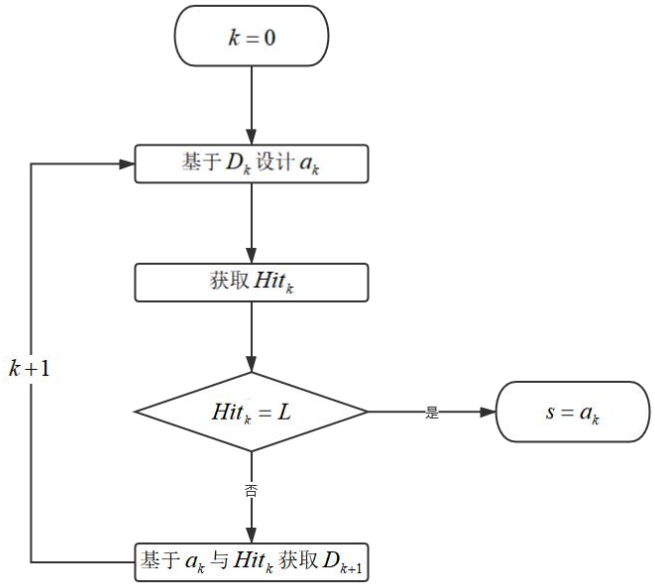

图 1. 算法设计的基础框架

基于词典的问题难度分析及猜测向量的设计

除了上述内容所述的有限步内停机约束，即可行性约束外，我们还希望经过每次迭代后都能获得一个更容易求解 s 的词典，换句话说每次迭代都将原始问题转化为了一个更易解答的问题，我们在问题公式化部分提出了基于词典衡量问题难度的方法（式（3））， 即设计的 a_k 应该是使得 $\min e(D_{k+1})$ 成立的最小值点。从而把设计 a_k 转化成了一个优化问题，然而由于式（3）中包含零范数，难以优化求解，只能通过 $a_k \in D_k$ 的约束枚举尝试。由于式（1）中 s 未知，所以对于给定 a_k， $e(D_{k+1})$ 的值只能基于期望给出，不妨记该期望为 E，其具体计算方法如下。

$$E=\sum_{Hit=0}^{L}P(D_k,Hit,a_k)\cdot \sum_{i}^{N_L}G(D_{ki},Hit,a_k)\sum_{j}^{N_L}G(D_{kj},Hit,a_k)\left\|D_i-D_j\right\|_0 \tag{5}$$

其中 $P(\cdot)$ 是 a_k 对应的命中程度 Hit 的概率函数， $G(\cdot)$ 是反映输入向量是否需要从当前词典中删除的门函数，二者的具体计算方法如下：

$$G(D_{ki},Hit,a_k)=\begin{cases}1, L-\left\|D_{ki}-a_k\right\|_0-Hit=0\\ 0, else\end{cases} \tag{6}$$

$$P(D_k, Hit, a_k) = \sum_{i=0}^{NL} G(D_{ki}, Hit, a_k) \Big/ NL \tag{7}$$

需要说明的是，在实际工程中，可以通过预先根据 Hit 分类并缓存的方法规避式（5）中的门函数 $G(\cdot)$，从而提高计算性能。我们设计的算法会在每次设计 a_k 时遍历 D_k，从而找到使得式（5）的最小值点，即 a_k。算法的具体伪代码见附录。

结果与分析

为简单起见，我们在构造测试集时选择忽略了英语构词规则，设计了一套随机生成单词的程序生成了一系列不同单词长度、不同规模的词典，并利用其检验了我们设计的算法性能。

对于每一个生成的词典，我们从中选取 20 个单词依次作为秘密进行模拟，统计 20 次模拟中求解秘密所需要的平均猜测数目（图 2）。

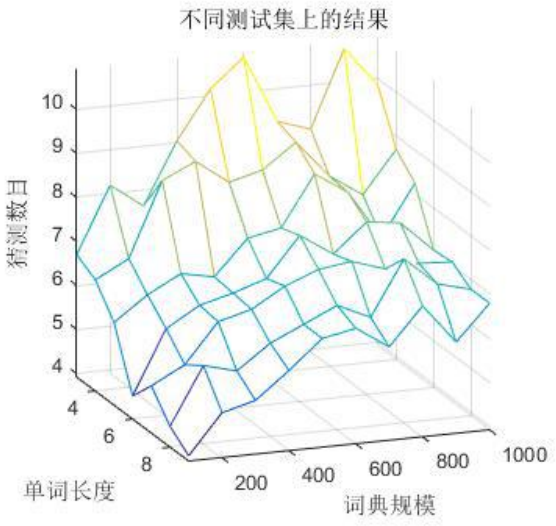

图 2. 在不同测试集上求解秘密所需的平均猜测次数

注意到随着单词长度增大，确定秘密需要的平均猜测次数是减小的。我们通过分析迭代过程中词典数目下降的速度，认为这可能是由于长的单词在命中程度上的概率函数较小，导致每次迭代都能把词典划分为更小的子词典，从而经过较小的猜测次数即可得到秘密。

虽然可以观察到在词典数据量较大的时候，解答秘密需要的平均猜测次数很接近拼字游戏要求的 10 次，但是考虑到实际英语单词会遵循一定的构词规则，换句话说在实际游戏的词典中我们用以衡量问题难易程度的函数式（3）会比测试集中完全随机生成的词典具有更小的值，故我们认为我们提出的算法在实际中会具有比在测试中更优秀的性能，足以在实际中使用。

结论

本报告讨论了广为流行的拼字游戏中的一个特殊形式，并给出一个优秀的游戏策略及从

优化方法角度讨论其数学原理，并基于自己构建的测试集对其进行了测试。测试结果表明，本报告提出的游戏策略可以在该游戏中使用，且具有较优秀的性能。

参考文献

[1] 王盈.拼字游戏备受美国青少年青睐[J].基础教育参考,2007(04):29.

附录

可以在 https://github.com/South-Walker/OptimizationMethod 中访问到本文中所涉及的具体代码及原始数据。

Algorithm 1 Search for the secret

Input: D: dictionary; n: number of words in D; l: number of letter in a word; $Host()$: an function that return the hitnumber between secret and input;

Output: secret s

while $|D| > 1$ **do**
    Select the optimal guess a;
    $Hit = Host(a)$
    if $Hit = l$ **then return** $s = a$
    end if
    $D = \{d \in D | l - \|d - a\|_0 = Hit\}$
end while

Algorithm 2 Select the optimal guess

Input: D: dictionary; n: number of words in D; l: number of letter in a word;

Output: optimal guess a^*

for $i = 1$ to l **do**
    initialize a dictionary D_i;
end for
for each $a \in D$ **do**
    for each $d \in D$ **do**
        $Hit = l - \|a - d\|_0$;
        $D_{Hit} = D_{Hit} \cup \{d\}$;
    end for
    for each D_i **do**
        $P_i = |D_i| / \sum_{j=1}^{l} |D_j|$;
        for each $d_j \in D_i$ **do**
            for each $d_k \in D_i$ **do**
                $E_i = E_i + \|d_j - d_k\|_0 * P_i$;
            end for
        end for
        if $E_i < E_{min}$ **then**
            $E_{min} = E_i$;
            $a_{min} = a_i$;
        end if
    end for
end for
return $a^* = a_{min}$;