基于全拼输入法的中式键盘排列的最优化探究

嘉定一中 王紫

摘要

最近一篇文章写到，目前的美式键盘不论在英文打字输入或中文输入都有其不合理性[1]，据调查，国外许多国家的键盘都结合本国需要作出键盘排列修改，而我国仍使用美式键盘，中式键盘的探究是很有必要的。结合汉字单字出现频度表，编程对拼音单个字母出现频度及相邻字母出现频度进行加权统计，排行制表。为了科学设计键盘，通过多人多次间歇实验的方式计算出键盘不同键位的敲击速度，配合已有拼音字母频率及相邻拼音字母频率，设计多种键盘排列，通过三个基于时间的最优化标准，严谨地求出最优化键盘排列，最终通过随机测试检验模型，验证了重新设计的键盘的高效，节约原先 9%的时间。这种设计键盘的方法，也可以适用于手机等键盘的设计，从而可以提高我们交流信息的效率。

A recent article[1] states that the keyboard of the QWERTY layout we use today is actually a very inefficient keyboard layout because the keyboard was first invented for typewriter, which slowed a typist down so as to reduce the frequency of the typewriter's typebars wedging together and jamming the machine. It's necessary to develop a new layout based on each country's language since we don't need to worry about a typewriter jam. First of all, the Chinese letters' frequency statistics were collected, and an experiment to record each key's speed was done. With the help of computer programming, we established a mathematical model and found out the most efficient layout of the keyboard by brute force search with pruning from three criteria with proof. To examine its efficiency, we ran a simulation program and arrived at the conclusion that the new layout could save 9% time of the old one. It's not only the most efficient Chinese keyboard layout but also a method to find the most efficient keyboard layout of all the languages in the world.

关键词 键盘排列 最优化键盘 全拼

前言

作为计算机的重要输入设备——键盘，并非像我们想象中的，是发明键盘的人经过严格推理，合理安排出的排列，反而，由于键盘在发明之初是为打字机服务，为了防止“卡键”，便设计出现在这样的最低效键盘。英国打字机博物馆馆长、《打字机世纪》一书的作者威尔弗雷德·A·比彻声称，“这种所谓‘科学安排’以减少手指移动距离的说法，是彻头彻尾的谎言。”[1] 而对于中文打字，美式键盘排列的打字效率似乎变得更低了。事实上，在许多国家，为了适应本国文字的需要，都会做出相应的对键盘排列的改变[2]，而我国仍在使用美式键盘排列，对于中文键盘，仅存在王永民先生的五笔输入法；但是，据不完全统计，目前的上网人群中 90%的人都热衷于全拼输入法[3]，因为它的重码率较低，并且对人来说，全拼字母的反应较快，会说便会打，而五笔输入码在人脑中的反应速度较慢，很难在年轻人间流行开，未来在键盘输入没有被脑电波控制取代之前，全拼输入法才是我们中国网民的主流。因此，我把研究的重心设为基于全拼输入法的中式键盘排列的最优化探究。

首先，我通过多人多次间歇实验的方式用测速软件测试出键盘不同键位的平均敲击速度；同时，我还需要对拼音单字母及相邻字母的频率进行统计，结合网上搜集到的用字频度表，我自学了简单的基础 C++课程，加权（原字出现频度的权）统计各字母在前五百个字

中的出现频度。接下来根据逐步完善的三个优化标准，在设计出的多个键盘排列中进行比较筛选，计算得出最优化的键盘排列。最后为了进一步验证该键盘的高效性及合理性，我用软件把大规模语料转化为拼音，编程测试新键盘与美式键盘的最小输入时间。根据模型检验，新键盘的全拼输入效率较美式键盘提高了 9.1%，而英文的输入效率也提高了 5%。这不仅验证了新键盘的高效，也进一步说明了美式键盘的确在应用于打字机时便是很低效的排列。

普通键盘如果要转变为我们新的布局其实是很方便的事，只要用键盘修改器软件把每个键位的输入值改变，键盘就可以完全转化，而不需要重新再买另一个键盘。个人认为，由于人的惰性关系，该键位布局恐怕无法得到大多普通的电脑使用者的青睐，但对键盘要求更高的电脑达人们也许会在使用新键盘布局时，发现他们的打字效率有很大提升。这也是我的研究的价值所在。

模型假设

1、除字母键以外的标点符号等按键不考虑

2、本文主要针对全拼输入法，因为五笔输入法本身按照最优化键位排列，二者并不冲突。由于简单易学，目前大部分上网人群都使用全拼输入法。

3、《成人用字表》[4]排行前 500 字累计频率达到 78.53%[5]，正文中统一使用前五百字的频度为基础进行运算。

4、每个字母键的单指敲击速度的计算：通过金山打字 2006（KINGSOFT,北京金山软件公司）对打字速度相差很大的若干人的各键敲击速度最大值进行测试，得到下表的平均近似值，速度数值保留到个位，速度单位 kpm 即 Keystrokes Per Minute，每分钟敲击键盘的次数。（其中 a’、b’等表示美式标准键盘上的 a、b 的位置。）

letter	a’	b’	c’	d’	e’	f’	g’	h’	i’	j’	k’	l’	m’
speed(KPM）	350	305	311	352	353	365	351	480	455	525	468	482	375
letter	n’	o’	p’	q’	r’	s’	t’	u’	v’	w’	x’	y’	z’
speed(KPM)	393	400	405	345	355	354	357	410	360	333	290	404	300

表 1

可知，键盘上字母键的速度从大到小为 j’, l’, h’,k’, i’,u’,p’, y’,o’, n’, m’, f’, v’,t’, r’, s’, e’, d’, g’, a’, q’, w’, c’,b’, z’, x’

由此对键盘各键位编号，依次对应为 k_1,k_2,k_3,…,k_26。

另，根据随机抽样测试，当双手分别打字时，速度约增加至 690kpm；当单手不同手指打字时，左手平均速度约为 450kpm，右手平均速度约为 650kpm；当单手指打字时，速度约减少至原来手指最大速度的 70%。

模型分析及求解

拼音各字母频度算法

处理数据库：首先将《成人用字表》前五百字转化为拼音形式，再通过 UltraEdit-32(版本 12.20b)对拼音及其出现万分比进行整理，以方便之后程序调用。

算法：设任意字母 a 在排名前 k 个高频字中每个字的出现次数为t_i (i=1，2，…，k)，这

些高频字的万分比依次分别为 p_i (i=1，2，…，k)。

则该字母频度衡量值 $W_a = \sum_{i=1}^{k}(t_i p_i)$；

同理算得所有字母的频度衡量值，总和为 $\sum_{i=1}^{26} W_i$；

最终可得各字母加权频度 $q_i = \frac{W_i}{\sum_{i=1}^{26} W_i}$。

通过 Microsoft Visual C++ 6.0 进行编程，对上述算法在电脑上模拟，取 k=500，（具体程序见附录）

```
for(int i=0;i<strlen(pinyin);i++)
{
    char t[2]={0};
    t[0]=pinyin[i];//pinyin 是分解出的字串
    word[t]+=d_pinlv;
//放入 map 表（word 表示频度衡量值，即 W；d_pinlv 是从数据库中获取的单字频率，即 p；这些字符在前面的程序中声明）
}
```

得到表 2：

表 2_前 500 字拼音字母加权频度排序表

排行	字母	频度衡量值	加权频度
1	i	3289.211000	0.137769
2	n	2863.871000	0.119953
3	a	2315.971000	0.097005
4	g	1867.994000	0.078241
5	e	1846.799000	0.077353
6	h	1829.493000	0.076628
7	u	1744.300000	0.073060
8	o	1323.303000	0.055427
9	d	877.017000	0.036734
10	z	834.585000	0.034957
11	y	758.036000	0.031750
12	s	719.196000	0.030124
13	j	658.932000	0.027599
14	x	363.898000	0.015242
15	l	363.613000	0.015230
16	c	359.336000	0.015051

17	b	314.592000	0.013177
18	w	266.502000	0.011162
19	t	244.988000	0.010261
20	q	244.078000	0.010223
21	r	208.260000	0.008723
22	f	204.657000	0.008572
23	m	197.918000	0.008290
24	k	113.707000	0.004763
25	p	47.437000	0.001987
26	v	17.174000	0.000719

设键盘上任意两键 a，b，其中二者速度 $v_a < v_b$，出现次数 $t_a > t_b$。T 表示打字时间。

此时打字时间

$$T_1 = t_a / v_a + t_b / v_b + others$$

若将两键位置交换，使 $v_a' = v_b, v_b' = v_a$

则

$$T_2 = t_a / v_a' + t_b / v_b' + others = t_a / v_b + t_b / v_a + others$$

$$\because T_2 - T_1$$

$$= t_a / v_b + t_b / v_a + others - (t_a / v_a + t_b / v_b + others)$$

$$= \frac{t_a - t_b}{v_b} + \frac{t_b - t_a}{v_a}$$

$$= (t_a - t_b)(\frac{1}{v_b} - \frac{1}{v_a}) < 0$$

$$\therefore T_2 < T_1$$

∴优化标准一：26 个不同出现频率的字母与速度不同的键位依次组合，满足字母统计频率的一篇文档的打字时间的最小值是字母频率从大到小与键位速度从大到小依次排列所得的组合。

由此可得初步键盘排列顺序，记为 P1，下图为 P1 的键盘图：

图 1_P1 键盘图

但是，这个方案并未对使用频度高的任意两个相连字母做出合理安排。因此我采用与统计拼音单字母加权频度相类似的算法，并且使 ab 与 ba 的计数效果相同（具体程序见附录），从而得到表 3。

```
for(int i=0;i<strlen(pinyin);i++)
{
    char t[3]={0};
    char s[3]={0};
    t[0]=pinyin[i];
    if(pinyin[i+1]==0)
        break;
    t[1]=pinyin[i+1];

    s[0]=t[1];
    s[1]=t[0];

    if(word.find(t)!=word.end())
        word[t]+=d_pinlv;
    else if(word.find(s)!=word.end())
        word[s]+=d_pinlv;
    else
        word[t]+=d_pinlv;
}
```

表 3_前 500 字相邻拼音字母加权频度排序表(部分)

排行	相邻字母	频度衡量值	加权频度
1	ng	1356.1790000000	0.084894
2	an	1273.2170000000	0.079701
3	ai	938.8490000000	0.058770
4	sh	629.5220000000	0.039407
5	ji	604.9690000000	0.037870
6	en	570.0720000000	0.035685
7	zh	543.4110000000	0.034016
8	ni	540.9790000000	0.033864
9	uo	518.3400000000	0.032447
10	hi	457.1060000000	0.028614
11	he	454.2890000000	0.028437
12	on	419.0760000000	0.026233
13	de	388.4570000000	0.024317
14	hu	369.6530000000	0.023139
15	ao	366.2070000000	0.022924
16	ha	353.6930000000	0.022140
17	ua	309.1220000000	0.019350
18	xi	307.7810000000	0.019266

19	yi	282.0830000000	0.017658
20	ch	275.7760000000	0.017263

整理数据，由于相同手指打字最慢，因此应该把出现频度高的相邻字母放在不同手指键位处。根据以上表格中数据的分析，其中位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
3	ai	938.8490000000	0.058770
7	zh	543.4110000000	0.034016
10	hi	457.1060000000	0.028614
15	ao	366.2070000000	0.022924
16	ha	353.6930000000	0.022140
19	yi	282.0830000000	0.017658
25	ho	194.7520000000	0. 012191
29	yo	135.8110000000	0.008501
36	ge	105.1170000000	0.006580
38	za	96.9540000000	0.006069
39	ya	89.9850000000	0.005633
52	zi	65.6860000000	0.004112
73	zo	24.8650000000	0.001556

按照单字母加权频度表，尽可能地将排序相邻的字母键位交换，由上表可以看出矛盾主要出现在右手食指中指，这样减小了调整范围，可以在人力下依照排行逐一解决。

优化标准二：

需要交换键位字母时应选择速度大于该字母速度 70%的键位交换。

若出现矛盾的 k 组相邻字母中有 m 个不同的字母，其中的任意字母 $r_j(j=1,2,...,m)$，在出现矛盾的 k 组相邻字母中出现的次数为 $s_{ji}(i=1,2,...,k)$，且这 k 组相邻字母的加权频度为 $d_i(i=1,2,...,k)$，使 $\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)$ 最小。

P1 中 m=8，k=13，$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P1}=0.457528$；

初步调整：a 与 g 交换，记为 P2，则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
7	zh	543.4110000000	0.034016
10	hi	457.1060000000	0.028614
19	yi	282.0830000000	0.017658
25	ho	194.7520000000	0.012191
29	yo	135.8110000000	0.008501
35	go	110.3790000000	0.006909
52	zi	65.6860000000	0.004112

73	zo	24.8650000000	0.001556

P2 中 m=6，k=8，$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P2}=0.227114$；

现在只有右手食指键位出现矛盾，并且在这个字母中出现最多的为h（频度和0.074821），所以应该首先考虑 h 键位的调整。由于 h 在前 500 字拼音字母加权频度排序表中位于第 6 位，所以首先应考虑与位于 k_5 的 e 或 k_7 的 u 交换。由于 he 在前 500 字相邻拼音字母加权频度排序表中位于第 11 位，故不予考虑；而 u 与 i，o 等字母都有相邻，故也不予考虑。同理，h 也无法与 k_1，2，3，4，8，10，11，13，14，15 交换。

P3：交换 h 与 k_9 的 d；则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
19	yi	282.0830000000	0.017658
29	yo	135.8110000000	0.008501
30	di	133.6560000000	0.008367
52	zi	65.6860000000	0.004112
60	do	50.0890000000	0.003135
73	zo	24.8650000000	0.001556

P3 中 m=5，k=6，$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P3}=0.086658$

P4：交换 h 与 k_12 的 s；则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
19	yi	282.0830000000	0.017658
29	yo	135.8110000000	0.008501
52	zi	65.6860000000	0.004112
66	si	39.7520000000	0.002488
73	zo	24.8650000000	0.001556

P4 中 m=5，k=5，$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P4}=0.06863$

P4 的 $\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)$ 值与 P3 相比较小，但键位速度改变量较大，优化标准一、二无法判断孰优孰劣。

优化标准三：

出现矛盾的 k 组相邻字母中有 m 个不同的字母，其中的任意字母 $r_j(j=1,2,...,m)$，在出现矛盾的 k 组相邻字母中出现的次数为 $s_{ji}(i=1,2,...,k)$，且这 k 组相邻字母的加权频度

为$d_i(i=1,2,...,k)$，字母键位的速度为$v_j(j=1,2,...,m)$，原位于矛盾位置的交换键位的单字母 x 的频率为q_x，原速度v_x，另一交换键位的单字母 y 的频率为q_y，原速度v_y，

使$T=[\sum_{j=1}^{m}\sum_{i=1}^{k}(\frac{s_{ji}d_i}{v_j})]+(\frac{q_x}{v_y}+\frac{q_y}{v_x}-\frac{q_x}{v_x}-\frac{q_y}{v_y})$最小。

P3：代入$v_j==525\times70\%$，$v_h=410$，$v_d=400$，$q_h=0.076628$，$q_d=0.036734$

$\therefore T_{P3}=2.382366426\times10^{-4}$

P4：代入$v_j==525\times70\%$，$v_h=410$，$v_s=365$，$q_h=0.076628$，$q_s=0.030124$

$\therefore T_{P4}=2.007321283\times10^{-4}$

又$\because T_{P3}>T_{P4}$

$\therefore$P4 为更优化方案。

P4 的 5 个字母中 i 的频度和最大，但 i 的单字母频度也最大，可交换键位的范围较小，因此优先考虑交换 y。可以与 y 交换的键位有 k_16,k_18,k_21,k_22。

P5：y 与 k_16 的 c 交换，则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
52	zi	65.6860000000	0.004112
66	si	39.7520000000	0.002488
72	ci	25.7500000000	0.001612
73	zo	24.8650000000	0.001556
80	co	17.0300000000	0.001066

$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P5}=0.021668$，$T_{P5}=6.173582656\times10^{-5}$

P6：y 与 k_18 的 w 交换，则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
52	zi	65.6860000000	0.004112
66	si	39.7520000000	0.002488
69	wo	34.6460000000	0.002169
73	zo	24.8650000000	0.001556

$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P6}=0.02065$，$T_{P6}=5.994441351\times10^{-5}$

P7：y 与 k_21 的 r 交换，则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
52	zi	65.6860000000	0.004112
66	si	39.7520000000	0.002488
67	ri	36.3270000000	0.002274
73	zo	24.8650000000	0.001556

$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P7}=0.02086$， $T_{P7}=6.210149896\times10^{-5}$

P8：y 与 k_22 的 f 交换，则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
52	zi	65.6860000000	0.004112
66	si	39.7520000000	0.002488
73	zo	24.8650000000	0.001556

$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P8}=0.016312$， $T_{P8}=5.218199816\times10^{-5}$

$\because [\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{p8}<[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P6}<[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P7}<[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P5}$

且 $T_{P8}<T_{P6}<T_{P5}<T_{P7}$

∴P8 是较佳方案。

P9：z 与 k_16 的 c 交换，则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
66	si	39.7520000000	0.002488
72	ci	25.7500000000	0.001612
80	co	17.0300000000	0.001066

$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P9}=0.010332$， $T_{P9}=3.385382107\times10^{-5}$

P10：z 与 k_18 的 w 交换，则此时位于相同手指键位的相邻字母有

排行	相邻字母	频度衡量值	加权频度
66	si	39.7520000000	0.002488
69	wo	34.6460000000	0.002169

$[\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P10}=0.009314$， $T_{P10}=3.258916669\times10^{-5}$

$$\because [\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P9} > [\sum_{j=1}^{m}\sum_{i=1}^{k}(s_{ji}d_i)]_{P10} \text{ 且 } T_{P9} > T_{P10}$$

∴P10 是较佳方案。

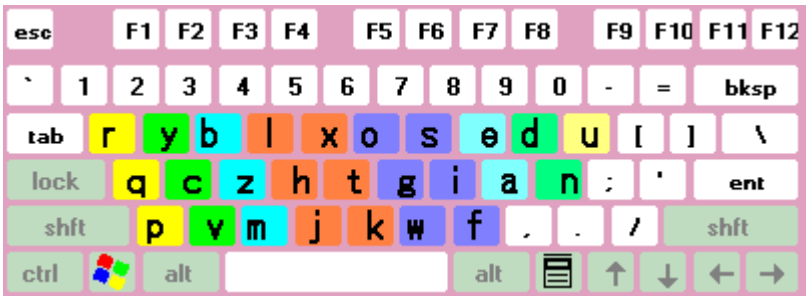

图 2_P10 的键盘图

同时，在同一手指键位或不在前 500 字相邻拼音字母加权频度排序表的字母范围内依照优化标准一，调整键位排序，记为 P11，键盘示意图如下：

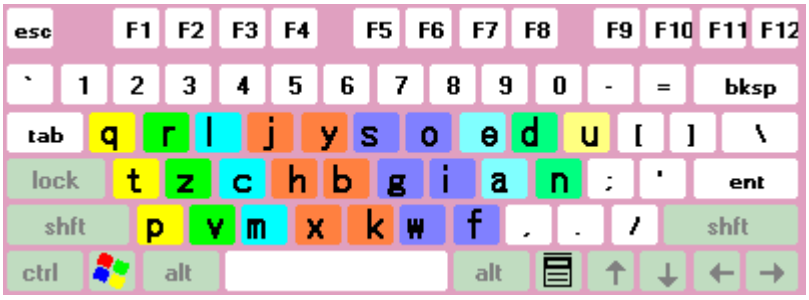

图 3_P11 键盘图

中式键盘排列的最优化方案即是从上往下第一排自左向右依次为 qrljysoedu，第二排为 tzchbgian,第三排为 pvmxkwf。这种方案下只有 0.93%的拼音字母在同一手指范围内。

模型检验

根据模型假设中的速度假设用Microsoft Visual C++ 6.0编程对之前所得的最优键盘排列进行检验（详细代码见附录）。核心代码：

```
for(int i=0;i<fsize;i++)
    {
        cur_char = file_buf[i];
        if(!isalpha(cur_char) )
        {
            pre_char = cur_char;
            pre_pk = pk;
            continue;
        }
        pk = find_key_seat(cur_char,method);
        if(pk == NULL)
            continue;
```

```
        //以下表示字母速度的选取（具体字符假设详见附录）
        if(pre_char ==0)
            t = t+(double)1/pk->v;
        else if(!isalpha(pre_char) )
            t = t+(double)1/pk->v;
        else if(pk->line == pre_pk->line)
            t = t+(double)1/(0.7*pk->v);
        else if(pk->hand == 0 && pk->hand == pre_pk->hand)
            t = t+(double)1/450;
        else if(pk->hand == 1 && pk->hand == pre_pk->hand)
            t = t+(double)1/650;
        else if(pk->hand != pre_pk->hand)
            t = t+(double)1/690;
        count++;
        pre_char = cur_char;
        pre_pk = pk;
    }
```

检测得到以下几个表格，其中时间衡量值与真实时间值倍数相差较大，但可以反应真实时间值间的比例关系。

《我的名字叫红》88,021 字 拼音字母 227,608 个

键盘方案	时间衡量值
优化方案	378.445
原键盘方案	412.949

《万物简史》115,058 字 拼音字母 306,115 个

键盘方案	时间衡量值
优化方案	502.58
原键盘方案	555.79

《比我老的老头》47,102 字 拼音字母 125,333 个

键盘方案	时间衡量值
优化方案	208.603
原键盘方案	226.264

《浮生六记》35,309 字 拼音 89151

键盘方案	时间衡量值

优化方案	173.652
原键盘方案	183.226

综合语料 拼音字母 746131 个

键盘方案	时间衡量值
优化方案	1259.84
原键盘方案	1374.53

《Harry Potter 6》字母 739,516 个

键盘方案	时间衡量值
优化方案	1473.64
原键盘方案	1538.69

《百年孤独（英文版）》字母 634,759 个

键盘方案	时间衡量值
优化方案	1252.75
原键盘方案	1317.19

《A Short History Of Nearly Everything》字母 766,012 个

键盘方案	时间衡量值
优化方案	1506.41
原键盘方案	1574.2

中文打字平均效率提高约 9.1%。英文打字平均效率提高 5%。

这说明，本键盘作为中文拼音最优化的最优化键盘的同时，在打英文上也不会比原来的美式键盘慢。

模型实际应用

这个设计键盘的模型不仅可以适用于基本键盘的设计，还可以应用在手机等键盘设计上。对于手机键盘，我们只需要把原来电脑键盘上的八组按键通过小幅调整，赋给手机上的八个字母输入键。

此外，普通键盘如果要转变为我们新的布局其实是很方便的事，只要用键盘修改器软件把每个键位的输入值改变，键盘就可以完全转化，而不需要重新再买另一个键盘。我觉得，由于人的惰性关系，该键位布局无法得到大多普通的电脑使用者的青睐，但对键盘要求更高的电脑达人们也许会在使用新键盘布局时，发现他们的打字效率有很大提升。

讨论

本模型精确严谨，采用键盘各键位分离记录速度参数的方法，使键盘设计更加确切，达到真正的最优化效果，通过随机测试可以看出，本键盘设计的速度达到了很好的水平，中文全拼打字平均节省时间 9%左右，英文打字也可以节约 5%左右的时间，因此不论中英文打字都能节省时间，并且全拼打字时，节时达到最大。但是不足之处是该键盘与原有键盘差异较大，不易很快掌握，因此不易推广。

已有相关研究比较：

《电脑键盘字母的优化排列》[6]松坪学校中学部

作者：马曦宇 蒲金山 吕方 陈涛

指导教师： 汤先君

“我们在英国牛津大学出版社出版的《书虫》——《LOVE OR MONEY》一书中统计了原文 7607 个字母（将近 2000 个英语单词）中 26 个英语字母分别出现的次数，计算出它们在字母总数中所占的频率，并画出了统计图。”

“按照字母的频率高低和食指优先、右手优先、中档键优先的原则，他们重新排列出键盘字母：从左至右上行 Q、K、C、S、G、M、R、L、B、J，中行 V、W、N、T、O、A、E、I、P，下行 Z、X、U、D、Y、F、H。为了验证新键盘的优越性，四位中学生进行模拟操作，实验证明按字母频率高低重新排列后的新键盘优越性十分明显，手指移动次数减少了一半。他们的研究成果《电脑键盘字母的优化排列》不久前获得今年深圳市科技创新大赛一等奖，专家对中学生这种敢于挑战传统、追求真理的行为给予肯定。”[7]

该研究语料相对涉及非常少，几乎没有代表性，并且没有依据相邻字母频率高的字母键位分开排列的原则，缺乏严谨性，在模型检验上由于通过人工测试，也没有普遍适用的基础。而我的探究的汉字频率源于网上可信大规模语料统计，统计语料总字数为 86405823 个，字种数为 6763 字，并且使用 C++编程，使大规模频率统计数据库转化为拼音单字母频率和相邻拼音双字母频率整理，非常严密并具有代表性，之后用严谨的数学推理得出最优化方案。在模型检验方面，我利用键位速度的测试进行编程，任何语料经拼音转化后都可以测试出打字时间衡量值，进行比较，效率提高且精确。

文献

1、http://news.ifeng.com/history/1/jishi/200808/0804_2663_693554_2.shtml

http://en.wikipedia.org/wiki/Typewriter#Keyboard_layouts:_.22QWERTY.22_and_others

2、http://www.mobile3g.cn/article/497/497969.shtml

3、http://www.tianya.cn/publicforum/content/funinfo/1/1539068.shtml

4、通过大规模语料统计得到的成人用字表（总表为 6763 个字，网上发布的字表为前 4400 字），该《成人用字表》由王良辰先生对 ChenShuyuan 先生转载清华大学统计资料进行加

工得来，统计语料总字数为 86405823 个，字种数为 6763 字（国标字符集）。

5、根据《中国语言生活状况报告（2007）》（已由商务印书馆出版发行，http://www.moe.gov.cn/edoas/website18/97/info1226566539390597.htm），高频词语中，一字词出现的总频次占全部高频词语频次的 49.04%，排名第一；二字词语的词种数最多，占全部词种的 70.53%，在频次上所占比例为 47.86%；一至二字词语在词种上占了 88.40%，在频次上占了 96.90%。说明在实际应用中，主要以一至二字词语为主。

6、http://spzx.spxx.com/UploadedImages/f2a33f84-a0f8-4065-8508-fd81f3ce065e.doc

7、http://zhidao.baidu.com/question/4739718.html

8、http://news.ifeng.com/history/1/jishi/200808/0804_2663_693554_2.shtml

使用软件

1、Microsoft Visual C++ 6.0

2、金山打字 2006（KINGSOFT,北京金山软件公司）

3、实用汉字转拼音 v4.2

（版权所有：Ken Guong，http://d.download.csdn.net/down/158287/xiaotian_ok）

4、UltraEdit-32(版本 12.20b)

附录

```
#include<iostream>
#include<string>
#include<map>
#include<vector>
#include <algorithm>
#include <functional>
#include <STDIO.H>
using namespace std;
#define MAX_INPUT_KEY 3

int model_analysis();
int pinlv();
int xianglin();
```

```
int main()
{
	while(1)
	{
		char input_key;
		cout<<"\n\n";
		cout<<"单字拼音字母加权频度统计————————1\n";
		cout<<"单字拼音相邻字母加权频度统计————2\n";
		cout<<"模型检验————————————————3\n";
		cout<<"退出————————————————————4\n";
		cout<<"选择（1，2，3）：";
		cin>>input_key;
		cout<<endl;
		int method = input_key - '0';
		if(method==0|| method > MAX_INPUT_KEY) break;
		switch(method)
		{
		case 1:
			pinlv();
			break;
		case 2:
			xianglin();
			break;
		case 3:
			model_analysis();
			break;

		}

	}
	return 0;
}


#include<iostream>
#include<string>
#include<map>
#include<vector>
#include <algorithm>
#include <functional>
#include <STDIO.H>
using namespace std;
class wordcount;


class wordcount
{
public:
	string word;
	double times;
	wordcount(string s,double t)
	{
		word=s;
		times=t;
	}
	int operator < (wordcount &w){return times<w.times;}
	int operator > (wordcount &w){return times>w.times;}
};
int UDgreater ( wordcount elem1, wordcount elem2 )
{
	return elem1 > elem2;
}

void takein(char* filename);
bool statistic_singleletter(char* filename);
bool statistic_xianglin(char* filename);
```

```
bool singlesplit(char* string,char* pinyin,char* begin,char* end);


double pin_sum = 0;
map<string,double> word;
vector<wordcount> word_count;


int pinlv()
{
//	string s;
//	while(cin>>s)//ctrl+z结束
//		word[s]++;

	pin_sum = 0;
	word.clear();
	word_count.clear();
	if (statistic_singleletter("E:\\常用字拼音\\前500字字母频率\\前500拼音.txt")==false)
		return 0;

	for(map<string,double>::iterator itr=word.begin();itr!=word.end();itr++)
	{
		wordcount w(itr->first,itr->second);
		word_count.push_back(w);
		pin_sum += itr->second;
	}
	sort(word_count.begin(),word_count.end(),UDgreater);
	for(vector<wordcount>::size_type i=0;i<word_count.size();i++)
	{
		cout<<word_count[i].word<<" "<<word_count[i].times<<endl;
	}
	takein("E:\\常用字拼音\\前500字字母频率\\前500拼音字母频率统计.txt");
return 0;
}

//加权统计拼音字母频度
bool statistic_singleletter(char* filename)
{
	FILE *sfp = NULL;
	if((sfp=fopen(filename,"r"))==NULL)
	{
	    cout<<"open file "<<filename<<" error"<<endl;
	    return false;
	}

      while(!feof(sfp))
	{
		char pinyin[100]={0};//将word内存初始化为0
		char pinlv[100]={0};
		char Info[1000]={0};//将Info内存初始化为0
		fgets(Info,1000,sfp);//获得一行
		if(singlesplit(Info,pinyin,",",",";") == true)//分解字串
		{
			if(singlesplit(Info,pinlv,";",""")==true)
			{
				double d_pinlv=0;
				d_pinlv=atof(pinlv);
				for(int i=0;i<strlen(pinyin);i++)
				{
					char t[2]={0};
					t[0]=pinyin[i];
					word[t]+=d_pinlv; //放入map表
				}
			}
		}
	}
```

```
        fclose(sfp);
        return true;
}



int xianglin()
{
        pin_sum = 0;
        word.clear();
        word_count.clear();
        if (statistic_xianglin("E:\\常用字拼音\\前500字字母频率\\前500拼音.txt")==false)
                return 0;
        for(map<string,double>::iterator itr=word.begin();itr!=word.end();itr++)
        {
                wordcount w(itr->first,itr->second);
                word_count.push_back(w);
                pin_sum += itr->second;
        }
        sort(word_count.begin(),word_count.end(),UDgreater);
        for(vector<wordcount>::size_type i=0;i<word_count.size();i++)
        {
                cout<<word_count[i].word<<" "<<word_count[i].times<<endl;
        }
        takein("E:\\常用字拼音\\相邻拼音排序qian500.txt");
return 0;
}


//相邻拼音字母加权频率统计
bool statistic_xianglin(char* filename)
{
        FILE *sfp = NULL;
        if((sfp=fopen(filename,"r"))==NULL)
        {
            cout<<"open file "<<filename<<" error"<<endl;
            return false;
        }

       while(!feof(sfp))
        {
                char pinyin[100]={0};//将word内存初始化为0
                char pinlv[100]={0};
                char Info[1000]={0};//将Info内存初始化为0
                fgets(Info,1000,sfp);//获得一行
                if(singlesplit(Info,pinyin,",",";") == true)//分解字串
                {
                        if(singlesplit(Info,pinlv,";","")==true)
                        {
                                double d_pinlv=0;
                                d_pinlv=atof(pinlv);
                                for(int i=0;i<strlen(pinyin);i++)
                                {
                                        char t[3]={0};
                                        char s[3]={0};
                                        t[0]=pinyin[i];
                                        if(pinyin[i+1]==0)
                                                break;
                                        t[1]=pinyin[i+1];

                                        s[0]=t[1];
                                        s[1]=t[0];

                                        if(word.find(t)!=word.end())
                                                word[t]+=d_pinlv; //放入map表
```

```
                                else if(word.find(s)!=word.end())
                                        word[s]+=d_pinlv; //放入map表
                                else
                                        word[t]+=d_pinlv;


                        }
                }
        }
    }

    fclose(sfp);
    return true;
}



//send info to a file
void takein(char* filename)
{
    FILE *sfp = NULL;
    if((sfp=fopen(filename,"w"))==NULL)
    {
        cout<<"open file "<<filename<<" error"<<endl;
        return ;
    }


    for(vector<wordcount>::size_type i=0;i<word_count.size();i++)
    {
        char lineinfo[500]={0};
        if(pin_sum > 0)
        sprintf(lineinfo,"%d    %s    %10.10f
    %10.6f\r\n",i+1,word_count[i].word.c_str(),word_count[i].times,(double)(word_count[i].times)/(double)pin_
sum);
        else
            sprintf(lineinfo,"%d    %s    %10.10f\r\n",i+1,word_count[i].word.c_str(),word_count[i].times);
        fputs(lineinfo,sfp);//写入一行
    }

    fclose(sfp);
    cout<<"已写入文件："<<filename<<endl;
    return;

}


//分解字串
bool splitstring(char* string,char* pinyin,char* begin,char* end)
{
    char *two = strstr(string,begin);
    if (two==NULL)
        return false;

    char *three = strstr(string,end);
    if (three==NULL)
        return false;
    if (three-two-2>0)
        memcpy(pinyin,two+2,three - two-2);
    else
    {
        int i=0;
    }
    return true;
}
```

```
bool singlesplit(char* string,char* pinyin,char* begin,char* end)
{
	char *two = strstr(string,begin);
	if (two==NULL)
		return false;

	char *three = strstr(string,end);
	if (three==NULL)
		return false;
	if (three-two-1>0)
		memcpy(pinyin,two+1,three - two-1);
	else
	{
		int i=0;
	}
	return true;
}


#include <iostream>
#include<string>
#include<map>
#include<vector>
#include <algorithm>
#include <functional>
#include <STDIO.H>
#include <sys/stat.h>
#include <sys/types.h>
using namespace std;

bool key_method(char* filename,int method);
struct KEY
{
	char letter;
	int line;
	int hand;
	int v;
};

	KEY k1[26] = {
		{'i',5,1,525},
		{'n',7,1,482},
		{'g',5,1,480},
		{'a',6,1,468},
		{'e',6,1,455},
		{'o',5,1,410},
		{'u',8,1,405},
		{'s',5,1,404},
		{'d',7,1,400},
		{'w',5,1,393},
		{'f',5,1,375},
		{'h',4,0,365},
		{'x',4,0,360},
		{'y',4,0,357},
		{'j',4,0,355},
		{'z',2,0,354},
		{'l',3,0,353},
		{'c',3,0,352},
		{'b',4,0,351},
		{'t',1,0,350},
		{'q',1,0,345},
		{'r',2,0,333},
		{'m',3,0,311},
		{'k',4,0,305},
		{'p',1,0,300},
```

```
        {'v',2,0,290},
    };
/*  KEY k3[26] = { //p1
        {'i',5,1,525},
        {'n',7,1,482},
        {'a',5,1,480},
        {'g',6,1,468},
        {'e',6,1,455},
        {'h',5,1,410},
        {'u',8,1,405},
        {'o',5,1,404},
        {'d',7,1,400},
        {'z',5,1,393},
        {'y',5,1,375},
        {'s',4,0,365},
        {'j',4,0,360},
        {'x',4,0,357},
        {'l',4,0,355},
        {'c',2,0,354},
        {'b',3,0,353},
        {'w',3,0,352},
        {'t',4,0,351},
        {'q',1,0,350},
        {'r',1,0,345},
        {'f',2,0,333},
        {'m',3,0,311},
        {'k',4,0,305},
        {'p',1,0,300},
        {'v',2,0,290},
    };*/
    KEY k3[26] = { //p10
        {'i',5,1,525},
        {'n',7,1,482},
        {'g',5,1,480},
        {'a',6,1,468},
        {'e',6,1,455},
        {'s',5,1,410},
        {'u',8,1,405},
        {'o',5,1,404},
        {'d',7,1,400},
        {'w',5,1,393},
        {'f',5,1,375},
        {'h',4,0,365},
        {'j',4,0,360},
        {'x',4,0,357},
        {'l',4,0,355},
        {'c',2,0,354},
        {'b',3,0,353},
        {'z',3,0,352},
        {'t',4,0,351},
        {'q',1,0,350},
        {'r',1,0,345},
        {'y',2,0,333},
        {'m',3,0,311},
        {'k',4,0,305},
        {'p',1,0,300},
        {'v',2,0,290},
    };
    KEY k4[26] = { //y换f以下表示字母速度的选取
        {'i',5,1,525},
        {'n',7,1,482},
        {'g',5,1,480},
        {'a',6,1,468},
        {'e',6,1,455},
        {'s',5,1,410},
        {'u',8,1,405},
        {'o',5,1,404},
```

```
        {'d',7,1,400},
        {'z',5,1,393},
        {'f',5,1,375},
        {'h',4,0,365},
        {'j',4,0,360},
        {'x',4,0,357},
        {'l',4,0,355},
        {'c',2,0,354},
        {'b',3,0,353},
        {'w',3,0,352},
        {'t',4,0,351},
        {'q',1,0,350},
        {'r',1,0,345},
        {'y',2,0,333},
        {'m',3,0,311},
        {'k',4,0,305},
        {'p',1,0,300},
        {'v',2,0,290},
    };

    KEY k2[26] = {
        {'j',5,1,525},
        {'l',7,1,482},
        {'h',5,1,480},
        {'k',6,1,468},
        {'i',6,1,455},
        {'u',5,1,410},
        {'p',8,1,405},
        {'y',5,1,404},
        {'o',7,1,400},
        {'n',5,1,393},
        {'m',5,1,375},
        {'f',4,0,365},
        {'v',4,0,360},
        {'t',4,0,357},
        {'r',4,0,355},
        {'s',2,0,354},
        {'e',3,0,353},
        {'d',3,0,351},
        {'g',4,0,351},
        {'a',1,0,350},
        {'q',1,0,345},
        {'w',2,0,333},
        {'c',3,0,311},
        {'b',4,0,305},
        {'z',1,0,300},
        {'x',2,0,290},
    };

#define MAX_INPUT_CHAR 4

int model_analysis()
{
    while(1)
    {
        char input_key;
        cout<<"\n\n";
        cout<<"方案1是P11\r\n方案2是美式键盘\n方案3是P10\n方案4为P8\nQ键退出\n";
        cout<<"输入方案(1,2,3,4):";
        cin>>input_key;
        cout<<endl;
        int method = input_key - '0';
        if(method > MAX_INPUT_CHAR) break;

        key_method("E:\\常用字拼音\\A Short History Of Nearly Everything.txt",method);

    }
```

```
	return 1;

}

KEY* find_key_seat(char key,int method)
{
	for(int i= 0;i<26;i++)
	{
		if(method ==1)
		{
		if(k1[i].letter==key)
			return &k1[i];
		}
		else if(method==2)
		{
			if(k2[i].letter==key)
			return &k2[i];
		}
		else if(method==3)
		{
			if(k3[i].letter==key)
			return &k3[i];
		}
		else if(method==4)
		{
			if(k4[i].letter==key)
			return &k4[i];
		}
	}
	return NULL;
}

int get_file_size(FILE *fp)
{
	int cur_pos = ftell(fp);
	fseek(fp,0,SEEK_END);
	int file_size = ftell(fp);
	fseek(fp,cur_pos,SEEK_SET);
	return file_size;
}

bool key_method(char* filename,int method)
{
	FILE *sfp = NULL;
	int fsize = 0;
	char* file_buf = NULL;
	if((sfp=fopen(filename,"r"))==NULL)
	{
	    cout<<"open file "<<filename<<" error"<<endl;
	    return false;
	}
	fsize = get_file_size(sfp);
	if(fsize <=0)
	{
	    cout<<"get file "<<filename<<" size error"<<endl;
	    return false;
	}

	file_buf = (char*)malloc(fsize);
	if(file_buf==NULL)
	{
	    cout<<"alloc "<<filename<<" mem error"<<endl;
	    return false;
	}
	fread(file_buf,fsize,1,sfp);
	fclose(sfp);
```

```
    char pre_char = 0;//前一字节
    KEY* pre_pk = NULL;
    char cur_char = 0;
    KEY* pk = NULL;
    double t=0;
    int count=0;
    for(int i=0;i<fsize;i++)
    {

        cur_char = file_buf[i];
        if(!isalpha(cur_char) )
        {
            pre_char = cur_char;
            pre_pk = pk;
            continue;
        }
        pk = find_key_seat(cur_char,method);
        if(pk == NULL)
            continue;

        if(pre_char ==0)
            t = t+(double)1/pk->v;
        else if(!isalpha(pre_char) )
            t = t+(double)1/pk->v;
        else if(pk->line == pre_pk->line)
            t = t+(double)1/(0.7*pk->v);
        else if(pk->hand == 0 && pk->hand == pre_pk->hand)
            t = t+(double)1/450;
        else if(pk->hand == 1 && pk->hand == pre_pk->hand)
            t = t+(double)1/650;
        else if(pk->hand != pre_pk->hand)
            t = t+(double)1/690;
        count++;
        pre_char = cur_char;
        pre_pk = pk;
    }
    free (file_buf);
    cout<<"方案的时间为"<< t <<endl;
    cout<<"文档输入字母为"<< count<<endl;
    cout<<"\n\n";

    return true;
}
```