Отчет по исследованию раскладки клавиатуры

Черетаев И.В

8 октября 2014 г.

1 Раскладка клавиатуры

В этом задании мы попытаемся создать программу для оценки удобства заданной раскладки клавиатуры.

За основы мы примем, что используется слепая 10-пальцевая печать и документы общего характера (художественная литература). Посмотрите расстановку рук при 10-пальцевой печати и зоны ответственности каждого пальца.

Какой же план исследования?

1.1 Нужно определить, какие параметры вы будете оценивать в качестве абстрактного "удобства".

Например, у меня они такие:

- пальцы редко убегают из среднего ряда;
- работают в основном указательные и безымянные;
- наибольшая активность сосредоточена в центре клавиатуры;
- руки эффективно чередуются;
- правая рука задействована чуть больше, чем левая.

1.2 Выразить эти параметры в численном отношении.

1.3 Завести в программе виртуального секретаря, который напечатает большой текст, например, знаменитый роман.

1.4 Оценить его "трудозатраты"по шкалам, определенным на 2 этапе.

1.5 Сделать выводы, сравнивая различные раскладки клавиатуры.

Сравнить надо 4 раскладки:

- QUERTY
- Colemak
- Дворак

на каких-нибудь текстах на английском. Например, "Госпожа Бовари"Густава Флобера. http://www.gutenberg.org/cache/epub/2413/pg2413.txt

- традиционную русскую раскладку на текстах вроде "Преступления и наказания".

2 Предварительные замечания о модели оптимизации

Были проанализированы все или почти все основные существующие модели назначения штрафов при оптимизации раскладок (таких, как Klausler, Capewell, QGMLWB и др.). Как правило, они недостаточно хорошо ортогонализированы. Т.е. одна штрафная компонента пересекается с другой, учитывающей то же обстоятельство. Пример: «прыжок» через ряд учитывается в расстоянии, проходимом пальцами; тот же прыжок штрафуется как сложная комбинация, хотя косвенно та же сложность учтена уже в расстоянии. Но дважды назначать штраф вовсе необязательно. И таких нюансов достаточно много.

После детального изучения наиболее приемлемой оказалась модель штрафов на andong.azurewebsites.net, но она не доработана. С другой стороны, очень хорошо проработана модель http://mkweb.bcgsc.ca/carpalx/, но в ней учитываются только расстояния, а не учитываются затраты на нажатие клавиш. Например, при наборе символов, находящихся в основной позиции, штраф будет равен нулю. Тогда как физическая работа не равна нулю, а скорость не бесконечна. Предварительный вывод таков: нужно использовать модель, общие затраты в которой будут складываться из затрат на перемещение пальца к нужной клавише и затрат на нажатие клавиши. На эти затраты будут накладываться поощрения и штрафы, учитывающие удобные/неудобные комбинации.

3 Вопросы построения модели для оптимизации раскладки

В чем главная сложность? Методы оптимизации (математические) хорошо отработаны, следовательно, сама оптимизация сложности не представляет. Главная сложность — составить функционал качества раскладки. Что будет выступать главным критерием? Раскладки, оптимизированные под скорость и под комфорт – вообще говоря, две отличающиеся вещи. Т.е. скоростная раскладка может быть даже вовсе не десятипальцевой. То же самое про удобство. Но в данном случае мы рассматриваем классический способ набора.

Что такое «слепой десятипальцевый метод печати (классический)»?:

1. фиксированная постановка рук;
2. фиксированные зоны – для каждого пальца свои буквы;
3. используются все десять пальцев (либо девять, в случае если пробел нажимается одним большим пальцем).

Как правильно составить модель качества раскладки?

1. нужно адекватно учесть плохие (хорошие) комбинации клавиш (отдельно определить, что есть «хорошо» или «плохо»);
2. выбрать компромисс между удобством и скоростью;
3. правильно распределить нагрузку на пальцы.

Во всех этих критериях есть некоторый субъективизм, но есть и утверждения, принимаемые всеми как истинные, подтвержденные при помощи различных программ. Например: нажатия клавиш разными руками, быстрее по скорости, чем одной рукой; набор на одном ряду чаще быстрее, чем на разных и т.д. Необходимо собрать все возможные случаи, проанализировать их, т.е. расставить по порядку предпочтительности. В то же время критериев не должно быть слишком много.

Далее нужно выбрать численные оценки качества. Допустим, каждой комбинации клавиш назначаем некоторое усилие в баллах. Чем меньше, тем лучше. Здесь появляется субъективизм.

- 1-ый субъективный выбор – какие комбинации клавиш хорошие (плохие)?
- 2-ой субъективный выбор – насколько лучше (хуже) та или иная комбинация?

Есть один объективный критерий – расстояние, проходимое пальцами от исходной позиции. Но тут нужно учитывать, что двигаются не только пальцы но и кисть, т.е. нажатие каждой клавиши происходит не обязательно из исходной позиции. Например: комбинация «ЗТ» — кисть движется к верхнему ряду, а потом к нижнему. Т.е. расстояние, проходимое до «Т» больше, чем из исходной позиции. Многие модели оптимизации минимизируют именно расстояние, но это не единственная составляющая, причем даже далеко не первая по важности. Основная часть работы приходится не на горизонтальные перемещения пальцев, а на нажатия клавиш. А число нажатий уменьшить нельзя (без автозамен). Поэтому основной путь совершенствования раскладки – минимизация доли неудобных комбинаций с учетом распределения нагрузок по пальцам каждой из рук.

Затраты на нажатие каждой клавиши, согласно предыдущему, будут складываться из двух составляющих: на горизонтальное перемещение и на нажатие. Для каждого пальца задаются коэффициенты затрат на одно нажатие и единицу перемещения. Допустим, что мы выбрали коэффициенты каким-либо образом. Далее нужно придумать систему поощрения хороших (быстрых и удобных) комбинаций и штрафования плохих. Допустим, мы это также сделали. Теперь у нас есть математическая модель затрат при наборе текста. Чтобы оценить качество раскладки по выбранной модели, нужно опробовать (прогнать) раскладку на каком-либо тексте.

Выборка текстов должна хорошо отражать статистическую структуру русского языка и включать: 1) достаточный объем (от нескольких мегабайт); 2) разные жанры (проза, фантастика, научные тексты, публицистика), так как используются различные сочетания букв и их комбинации; 3) возможно, даже разные эпохи (классика, современность). Далее этот текст прогоняется по модели оценки качества раскладки, и определяется число баллов. Раскладка с минимальным (или максимальным, в зависимости, от того, как построена модель) числом баллов – лучшая.

4 Клавиши, учитываемые при оптимизации и система штрафов

Для учета Shift'ов при оптимизации раскладки оказалось, что данных диграмм в некоторых случаях не хватает, и первичный алгоритм сбора статистики нуждается в модификации, либо нужно восстанавливать недостающие данные, перебирая все триграммы и т.д. Диграммы с пробелами учитываются. Это тоже нагрузка при наборе. И она скажется при оценке того, насколько выгоднее та или иная раскладка. Т.е. пробелы должны понижать выигрыш любой альтернативной раскладки, так как они нажимаются в любом случае и для всех раскладок одинаково (если рассматривать классический десятипальцевый метод набора). За основу была взята модель с сайта andong.azurewebsites.net, но с различными дополнениями. Общие усилия (или затраты) складываются из двух составляющих – на нажатие клавиши каждым пальцем и на единицу пути, проходимого каждым пальцем. Затраты по нажатиям для мизинцев немного уменьшены, т.к. в исходном варианте они кажутся несоразмерно высокими.

4.1 Затраты на одно нажатие:

- 2,10 левый мизинец
- 1,45 левый безымянный
- 1,05 левый средний
- 1,05 левый указательный
- 1,00 правый указательный
- 1,00 правый средний
- 1,40 правый безымянный
- 2,00 правый мизинец
- 1,00 большие пальцы

4.2 Затраты на единицу перемещения (за единицу взята сторона клавиши):

- 1,60 левый мизинец
- 1,00 левый безымянный
- 0,55 левый средний
- 0,75 левый указательный
- 0,70 правый указательный
- 0,50 правый средний
- 0,90 правый безымянный
- 1,50 правый мизинец

Как мы видим, коэффициенты несколько больше для левой руки, что будет поощрять использование правой руки в процессе оптимизации. Т.е. данный вариант — скорее для правшей. Для левшей коэффициенты для правой и левой рук необходимо поменять. Отдельно стоит оговорить асимметрию загрузки рук. На мой взгляд, асимметрия по количеству нажатий в 5-10% (т.е. если одна рука делает в 1,05-1,1 раза больше нажатий, чем другая) вообще не играет роли.

Видим, что затраты на нажатия приняты минимальными для указательного и среднего пальцев, т.к. они примерно одинаково развиты. Возможно, коэффициенты для безымянного и мизинца следует еще уменьшить. Затраты на перемещения минимальны для среднего пальца, т.к. он длиннее остальных. Отдельный вопрос о соотношении затрат на перемещение для безымянных и мизинцев. Мизинцы — самые слабые пальцы, но они более подвижные, т.к. это крайние пальцы. Для больших пальцев затраты на горизонтальные перемещения отсутствуют.

Т.е. для каждой нажимаемой клавиши затраты складывались из двух составляющих: на горизонтальное перемещение и на нажатие. Горизонтальное перемещение может быть нулевым, но суммарные затраты никогда не будут нулевыми (как и в реальности), чем и объясняется выбор такой модели в качестве основы. Для представления выборки текстов

было решено использовать статистику диграмм. С одной стороны, их значительно меньше, чем триграмм, что должно ускорить процесс оптимизации, с другой стороны, даже диграммы позволяют учесть основные быстрые/медленные комбинации, например, чередование рук.

Дополнительные бонусы и штрафы

Какие дополнительные бонусы и штрафы нужно начислять (т.е. как поощрять удобные, быстрые диграммы и штрафовать плохие)? Было решено ввести следующие пункты:

1. Бонус скорости для чередования рук. Возможно, усилия и не будут меньше при наборе диграммы разными руками, но скорость в среднем будет значительно выше, чем при наборе одной рукой. Отнимаем 60% от перемещений для второй буквы диграммы (это можно оправдать еще и тем, что большая часть пути до второй буквы будет проделана одновременно с нажатием первой, т.к. в данном случае руки двигаются независимо). Пример диграммы – «ШУ».

2. Штраф на нажатие той же клавиши. Горизонтальные перемещения для второй буквы в диграмме принимаются равными нулю, поскольку палец остается над той же позицией. А усилие на нажатие увеличиваем на 20%. Пример – «СС», «ЕЕ».

3. Штраф на обратную комбинацию одной рукой. Если диграмма набирается одной рукой в направлении от центра клавиатуры к краю, то усилия для второй буквы как на перемещения, так и на нажатие увеличиваются на 20% (подразумевается уменьшение удобства и скорости). Пример – «ВЫ», «ОЛ».

4. Штраф на однопальцевые комбинации (исключая повторное нажатие той же буквы). Усилия на перемещение и нажатие увеличиваются на 30% (подразумевается, что может значительно уменьшиться скорость). Пример – «МА», «АК», «ОГ», «ОТ» — в стандартной раскладке таких комбинаций очень много, т.к. 50% нажатий делают указательные пальцы.

5. Штрафование нижнего ряда. Считается, что на стандартных клавиатурах доступность нижнего ряда хуже, чем верхнего, поэтому необходимо ввести коэффициент сложности. Для любого символа нижнего ряда затраты как на перемещение, так и на нажатие увеличиваются на 25%.

Некоторые бонусы и штрафы могут действовать совместно. Например, может быть согласованное перемещение на нижний ряд, но диграмма набирается в обратном порядке. Т.е. получаем бонус за согласованное перемещение и штрафы за нижний ряд и за обратную комбинацию другой рукой. Пример – «СЧ».

5 «Соревнование»

А теперь проведем наше небольшое «соревновение» среди раскладок клавиатуры. Итак, в конкурсе участвуют:

- QUERTY
- Colemak
- Дворак
- традиционная русская раскладка ЙЦУКЕН

Англоязычные раскладки тестируются на тексте "Госпожа Бовари"Густава Флобера. http://www.gutenberg.org/cache/epub/2413/pg2413.txt

ЙЦУКЕН тестируется на романе Льва Николаевича Толстого «Война и мир», книга 1

Результаты тестирования:

- Русская раскладка: 332.934 балла
- QWERTY: 379,913 баллов
- Дворак: 323.685 балла
- Colemak: 375.181 баллов

Приложение 1 Python

```
#coding:utf8

from math import *


def main(file, f_but, f_penalties):
    f1 = open(f_but)
    buttons = {}
    for b in f1:
        buttons[b[0]] = [float(s) for s in b[1:].split()]
        buttons[b.upper()[0]] = [float(s) for s in b[1:].split()]
    buttons['\n'] = [2, 5, 2, 7]
    buttons[' '] = [1, 1, 0, 1]
    f2 = open(f_penalties)
    diagram = {}
    penalties_for_finger = {}
    for str in f2:
        s1, s2, s3, s4 = [float(s) for s in str.split()]
        penalties_for_finger[(s1, s2)] = (s3, s4)
    penalties_diagram = {}
    f3 = open(file)
    str = f3.read()
    last_c = str[0]
    count = 0
    for c in str[1:]:
        if last_c+c in diagram.keys():
            diagram[last_c+c] += 1
        else:
            diagram[last_c+c] = 1
        last_c = c
        count += 1
    summ = 0
    for s in diagram.keys():
        if s[0] in buttons.keys() and s[1] in buttons.keys():
            p1, p2 = buttons[s[0]][0:2]
            p3, p4 = buttons[s[1]][0:2]
            penalties_diagram[s] = penalties_for_finger[(p1, p2)][0] +\
                                   penalties_for_finger[(p3, p4)][0] +\
                                   penalties_for_finger[(p3, p4)][1]*sqrt(\
                                   (buttons[s[0]][2]-buttons[s[1]][2]) ** 2 +\
                                   (buttons[s[0]][3]-buttons[s[1]][3]) ** 2)
        if buttons[s[0]][0] != buttons[s[1]][0]:
            penalties_diagram[s] -= 0.6*penalties_for_finger[(p3, p4)][1]*sqrt( \
                                   (buttons[s[0]][2]-buttons[s[1]][2]) ** 2 + \
                                   (buttons[s[0]][3]-buttons[s[1]][3]) ** 2)
        if s[0] == s[1]:
            penalties_diagram[s] += 0.2*penalties_for_finger[(p3, p4)][0]
        if s[0] != s[1] and buttons[s[0]][:2] == buttons[s[1]][:2]:
```

```
49                  penalties_diagram[s] += 0.3*(penalties_for_finger[(p1, p2)
                        ][0] + \
50                                      penalties_for_finger[(p3, p4)][0] + \
51                                      penalties_for_finger[(p3, p4)][1]*sqrt(
                                            \
52                                      (buttons[s[0]][2]-buttons[s[1]][2]) **
                                            2 + \
53                                      (buttons[s[0]][3]-buttons[s[1]][3]) **
                                            2))
54              if buttons[s[1]][2] == 1:
55                  penalties_diagram[s] += 0.25*(penalties_for_finger[(p1, p2)
                        ][0] + \
56                                      penalties_for_finger[(p3, p4)][0] + \
57                                      penalties_for_finger[(p3, p4)][1]*sqrt(
                                            \
58                                      (buttons[s[0]][2]-buttons[s[1]][2]) **
                                            2 + \
59                                      (buttons[s[0]][3]-buttons[s[1]][3]) **
                                            2))
60              summ += diagram[s]*100/count*penalties_diagram[s]
61
62          print(summ)
63          f1.close()
64          f2.close()
65          f3.close()
66
67
68      if __name__ == '__main__':
69      file = "input.txt"
70          f_but = "buttons.txt"
71          f_penalties = "penalties.txt"
72          main(file, f_but, f_penalties)
```