Second part report

Discrete optimization

Fabrice Servais (S111093)
Laurent Vanosmael (S114351)

Masters in Computer Science and Engineering - 1st year

December 7, 2014

1 Introduction

2 Data

The data listed below are the ones from the Integer Model that we have used in the first report.

- $sl_k = \begin{cases} 1 & \text{if the key } k \text{ is on the left side,} \\ 0 & \text{otherwise.} \end{cases}$
- $sr_k = \begin{cases} 1 & \text{if the key } k \text{ is on the right side,} \\ 0 & \text{otherwise.} \end{cases}$
- fr_k : probability of the apparition of the letter l in the considered language.
- $v_l = \begin{cases} 1 & \text{if the letter } l \text{ is a vowel,} \\ 0 & \text{otherwise.} \end{cases}$
- V : number of vowels in the language.
- $w_{i,j}$: probability that the letter j follows the letter i in a word.
- ks_k : strength of the finger associated to key k ($ks_k \in [0, 1]$).
- d_k : distance that the finger attributed to the key k has to cross to reach that key.

2.1 Data's collect

Concerning the data related to the keyboard (sl_k, sr_k, d_k), we have based our analysis on the keyboard present on the BEPO graphic in the assignment paper. The distance is calculated as the number of "moves" from the home row the finger has to do. There is no diagonal moves.

For the strength of the fingers, we have arbitrarily decided that, on a scale of 1, the index will be 0.75, the middle finger 1, the ring finger 0.5 and the little finger 0.1. We have also matched each key with a finger. With the strength of the finger associated, we can thus construct ks_k.

To collect the frequencies of the appearance of the letters and the bi-grams, we have used the NGramExtraction tool from Deception. To obtain the alphabet that we used, we just take the 47 characters that have the highest number of occurrence (47 is the number of keys in our keyboard). Then we divide the number of occurrence of each letter by the total number of occurences to have the frequency of each letter. We also "normalize" the bi-gram so that the sum of each bi-gram frequency starting by a specified letter makes always 1.

2.2 Data file organisation

The file of data (named `data`) is structured to be also human readable, the first word is the name of the variable, which is followed by its content (note that it has to be in a specified order for the parser program). The data do not have necessarily the same format as in the model, for example for sl and sr, it's the number of the key that is indicated. The parser translates it into binary data. We have done this in that way to avoid errors and to keep the data file readable. fs represent the finger strength and $kb < x >$ the key associated to a certain finger (there is no key for the thumbs).

3 Model

3.1 Variables

- $kb_{k,l} = \begin{cases} 1 & \text{if the letter } l \text{ is on the key } k, \\ 0 & \text{otherwise.} \end{cases}$
- $vl = \begin{cases} 1 & \text{if the vowels are on the left side,} \\ 0 & \text{otherwise.} \end{cases}$
- $a_{i,j} = \begin{cases} 1 & \text{if to type } i \text{ and then } j, \text{ it is not the same hand that is used,} \\ 0 & \text{otherwise.} \end{cases}$

3.2 Constraints

- $\sum_l fr_l \cdot (\sum_k kb_{k,l} \cdot (sr_k - sl_k)) \geq 0$
- $\sum_k kb_{k,l} = 1, \forall l$
- $\sum_l kb_{k,l} = 1, \forall k$
- $\sum_l v_l \cdot (\sum_k kb_{k,l} \cdot sl_k) = V \cdot vl$
- $a_{i,j} = a_{j,i}, \forall i, j$
- $a_{i,i} = 0, \forall i$
- $a_{i,j} \leq \sum_k kb_{k,i} \cdot sl_k + \sum_k kb_{k,j} \cdot sl_k$
- $a_{i,j} \geq \sum_k kb_{k,i} \cdot sl_k - \sum_k kb_{k,j} \cdot sl_k$
- $a_{i,j} \geq \sum_k kb_{k,j} \cdot sl_k - \sum_k kb_{k,i} \cdot sl_k$
- $a_{i,j} \leq 2 - \sum_k kb_{k,i} \cdot sl_k - \sum_k kb_{k,j} \cdot sl_k$

3.3 Objective function

$$\min \left[\sum_l fr_l \cdot \left(\sum_k ks_k \cdot d_k \cdot kb_{k,l} \right) + \sum_i \sum_j w_{i,j} \cdot (1 - a_{i,j}) + \sum_i \sum_j w_{i,j} \left(\sum_k d_k \cdot kb_{k,i} + \sum_l d_l \cdot kb_{k,j} \right) \right]$$

4 Method

To obtain an optimal keyboard related to our model by using the **row generation** method. To implement that method, we use 2 concepts that are present in Gurobi : the lazy constraint and the callback.

The lazy constraints are the constraints that are less likely to be violated, so they are not contained in the set of the beginning but are added as we go along the execution of the solver. In order to do that, we use the callback system. We provide a class containing a `callback()` function which is called at key steps of the resolution. In our implementation, the function is called when the solver found a solution. This function checks if one of the lazy constraints is violated or not. If yes (and is not yet added), we add it to the model by using the `addLazy()` function.

The constraints that are considered lazy are these one : $\forall i, j$,

- $a_{i,j} \leq \sum_k kb_{k,i} \cdot sl_k + \sum_k kb_{k,j} \cdot sl_k$
- $a_{i,j} \geq \sum_k kb_{k,i} \cdot sl_k - \sum_k kb_{k,j} \cdot sl_k$
- $a_{i,j} \geq \sum_k kb_{k,j} \cdot sl_k - \sum_k kb_{k,i} \cdot sl_k$
- $a_{i,j} \leq 2 - \sum_k kb_{k,i} \cdot sl_k - \sum_k kb_{k,j} \cdot sl_k$

It represents a total of $4 * 47 * 47 = 8836$ constraints.

5 Results

5.1 Clavier français

- Sample : "Madame Bovary", Gustave Flaubert.

The result is shown FIGURE 1.

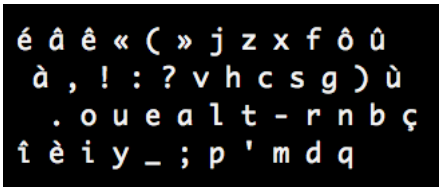

Figure 1: Keyboard : French

There are the numbered results :

Objective function	123.286
Row 1	3.67 %
Row 2	16.89 %
Row 3	60.65 %
Row 4	18.75 %
Time	1024 sec

6 Improvements