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 Datas

- $sl_k = \begin{cases} 1 & \text{if the key } k \text{ is on the left side,} \\ 0 & \text{otherwise.} \end{cases}$
- $sr_k = \begin{cases} 1 & \text{if the key } k \text{ is on the right side,} \\ 0 & \text{otherwise.} \end{cases}$
- 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.

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, we add it to the model by using the `addLazy()` function.

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

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

5 Results

é		â		ê		«		(		»		j		z		x		f		ô		û				
	à		,		!		:		?		v		h		c		s		g		)		ù			
		.		o		u		e		a		l		t		-		r		n		b		ç		
î		è		i		y		_		;		p		’		m		d		q						

6 Improvements