



HAL
open science

Relabeling nodes according to the structure of the graph

Ronan Hamon, Céline Robardet, Pierre Borgnat, Patrick Flandrin

► **To cite this version:**

Ronan Hamon, Céline Robardet, Pierre Borgnat, Patrick Flandrin. Relabeling nodes according to the structure of the graph. 2013. ensl-00878041

HAL Id: ensl-00878041

<https://hal-ens-lyon.archives-ouvertes.fr/ensl-00878041>

Submitted on 29 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Relabeling nodes according to the structure of the graph.

Ronan Hamon, Céline Robardet, Pierre Borgnat, Patrick Flandrin

1 Proposed method

1.1 Algorithm

We propose to solve this problem thanks to a two-step algorithm. The first step is based on the depth-first search algorithm and enables us to obtain a collection of independent paths. From a starting node with minimal value of closeness centrality, the algorithm jumps from another node according to the neighborhood of the considered node. The neighborhood is computed such that a node already taken into account in a path is not included. If one or more of his neighbors have a degree equal to 1, that means the neighbor node is only linked to the considered node, the node is added to the path and another neighbor is considered. If all neighbors have a degree greater than 1, the next node is chosen taking the highest value of a criterion based on the Jaccard index between neighborhood of the considered node and each of its neighbors. This criterion determines which neighbors is the most similar to the current node in order to stay in the same part of the graph. The other neighbors are stacked in a pile and the algorithm repeats the same procedure from the chosen node. When no neighbors are available, the procedure stops and the path is closed. A new path is opened and starts from the last node put in the pile and so on. At the end of step 1, there is a collection of paths which are independent i.e. no vertex is in two different paths.

The second step aims to aggregate these paths in order to minimize the cyclic bandwidth sum. The paths are considered following their decreasing lengths. The longest path is first considered and inserted into a empty list called labeling. The second longest path is then considered and inserted at all available indices in the labeling : for each insertion, a criterion based on the cyclic bandwidth sum is computed. The path is inserted definitively at the index which minimized this criterion. The algorithm goes on until the collection of paths is empty.

Algorithm 1 Minimization_Cyclic_Bandwidth_Sum

Require: $G = (V, E)$

Ensure: π a one-to-one and onto mapping of V to $\{0 \dots n - 1\}$. L , labeling two piles. Paths a heap.

```
1: for all  $u \in V$  do
2:    $\text{color}[u] \leftarrow \text{white}$ 
3:    $\pi[u] \leftarrow \text{nil}$ 
4: end for
5:  $\text{centrality} \leftarrow \text{Closeness\_Centrality}(G)$ 
6: for all Connected components  $C$  of  $G$  do
7:    $V_C \leftarrow \text{Vertices of } C$ 
8:   for all  $u \in V_C$  do
9:      $\text{Heap\_Push}(S, (\text{centrality}[u], u))$ 
10:  end for
11:  while  $S$  is not empty do
12:     $u_0 \leftarrow \text{Heap\_Pop}(S)$ 
13:    if  $\text{color}[u_0] = \text{white}$  then
14:       $P \leftarrow \text{Find\_best\_path}(u_0, C, \text{color}, \text{centrality})$ 
15:       $\text{Heap\_Insert}(\text{Paths}, (\text{length}(P), P))$ 
16:    end if
17:  end while
18:  while Paths is not empty do
19:     $\text{path} \leftarrow \text{Max\_Heap\_Extract}(\text{Paths})$ 
20:     $\text{Insert\_path}(\text{labeling}, \text{path}, C, \text{color})$ 
21:    for all  $u \in \text{path}$  do
22:       $\text{color}[u] \leftarrow \text{black}$ 
23:    end for
24:  end while
25: end for
26: for  $i \in [0, \dots, n - 1]$  do
27:    $\pi[i] \leftarrow \text{Index}(\text{labeling}, i)$ 
28: end for
```

Algorithm 2 Find_best_path($u_0, C, \text{color}, \text{centrality}$)

Ensure: P a pile. H a heap.

```
1:  $u \leftarrow u_0$ 
2: while  $u \neq -1$  do
3:   Push( $P, u$ )
4:   for all  $v \in \text{adj}[u]$  do
5:     if  $\text{color}[v] = \text{white}$  then
6:       if  $\text{degree}(v) = 1$  then
7:         Push( $P, v$ )
8:          $\text{color}[v] \leftarrow \text{gray}$ 
9:       else
10:         $j \leftarrow \text{Modified\_Index\_Jaccard}(u, v)$ 
11:         $c \leftarrow \text{centrality}[v]$ 
12:        Heap_Insert( $H, (v, c, j)$ )
13:      end if
14:    end if
15:  end for
16:   $\text{color}[u] \leftarrow \text{gray}$ 
17:  if  $H$  not empty then
18:     $u \leftarrow \text{Min\_Heap\_Extract}(H)$ 
19:  else
20:     $u \leftarrow -1$ 
21:  end if
22: end while
23: return  $P$ 
```

Lines 8-17 concerns the step 1 of the algorithm whereas lines 18-25 concerns the step 2.

Algorithm 3 Modified_Index_Jaccard(u, v)

Ensure: nb_u, nb_v two piles.

```
1: for all  $w \in \text{adj}[u]$  do
2:   if  $\text{color}[w] = \text{white}$  then
3:     nb_u, w
4:   end if
5: end for
6: for all  $w \in \text{adj}[v]$  do
7:   if  $\text{color}[w] = \text{white}$  then
8:     nb_v, w
9:   end if
10: end for
11: return  $\frac{\#(\text{nb}_u \cup \text{nb}_v)}{\#(\text{nb}_u \cap \text{nb}_v)}$ 
```

Algorithm 4 *Insert_path*(*labeling*, *path*, *C*, *color*)

```
1: best_index  $\leftarrow$  0
2: best_cbs  $\leftarrow$  Criterion(labeling, path, C, color)
3: for all  $i \in ]0, \dots, \text{length}(\text{labeling})]$  do
4:   cbs  $\leftarrow$  Criterion(Insert(labeling, path, i), path, color)
5:   if cbs < best_cbs then
6:     best_index  $\leftarrow$  i
7:     best_cbs  $\leftarrow$  cbs
8:   end if
9: end for
10: return labeling  $\leftarrow$  INSERT(labeling, path, best_index)
```

Algorithm 5 *Criterion*(*labeling*, *path*, *C*, *color*)

```
1: CBS  $\leftarrow$  0
2: n  $\leftarrow$  #V
3: for all  $u \in \text{path}$  do
4:   for all  $v \in \text{adj}[u]$  do
5:     if color[v] = black then
6:       label_u  $\leftarrow$  Index(labeling, u)
7:       label_v  $\leftarrow$  Index(labeling, v)
8:       CBS  $\leftarrow$  CBS + min(|label_u - label_v|, n - |label_u - label_v|)
9:     end if
10:   end for
11: end for
12: return cbs
```
