221: Information Retrieval Winter 2010

Finding dense clusters in web graph

Prof. Donald J. Patterson Scribe: Minh Doan, Ching-wei Huang, Siripen Pongpaichet

1 Overview

In the assignment, we studied on top 500K wiki-pages. We created the web graph of those pages, and implemented an algorithm of finding dense clusters in that large web graph which was proposed in [1]. In addition, We also proposed a map-reduce design approach which is best fit to the algorithm. The experience on the created graph showed that algorithm works well and the its results provided some interests of the top 500K pages.

2 Motivation

Considering the web graph of top 500k wiki pages, we were curious about groups of pages which are clustered with the respect of some particular relationship such as finance, computer, math, travel, etc. In term of graph, they may group together as strong connected subgraphs.

Finding dense subgraphs in a graph is a challenge, especially in a very large graph. It has been studied well and solved in different definitions of density of a subgraph. We are very interested in this opportunity to identify strong clusters from this large graph instance (500k wiki web graph).

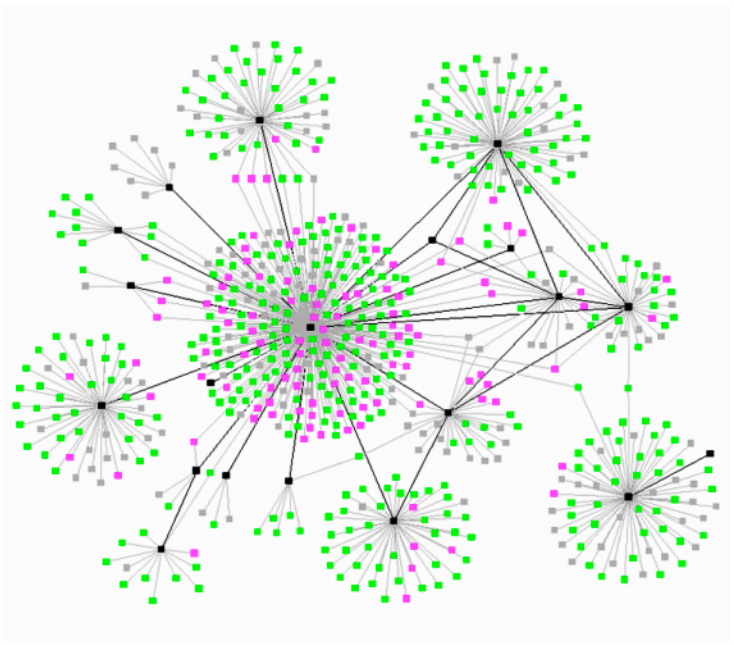
Finding dense subgraphs should have some applications in social network, attacking spammer in Google ads, etc. Assuming Facebook might be going to release some service or application which is properly used in strongly-connected communities, we are to find those communities in their graph which have 400 millions active users. As for Google adsense program, there are many bad publishers trying to take advantage of the program to spread out their revenue. A well-known way to do is that they create multiple accounts which advertise Google ads in their fake webpages generated by bots which attempt to click on ads of the fake webgpages. Due to being created from a user or group of users, those accounts may have some attributes in common and group together as a strongly-connected subgraph in the whole graph of publishers in those attributes.

3 Project proposal

In this work, we fullfiled the following tasks:

- Create a web graph of top 500k wiki pages.
- Identify dense subgraphs from the created graph.
- Find some interests from the result.

4 Creating web graph for top 500k wiki pages



For this step, we use the hadoop framework to create the web graph. With the support of a given list of wikipage URLs, we can easily design a MapReduce algorithm to create the graph. As a result, the MapReduce output is represented as adjacency list of the web graph.

More further, as from our view, it's difficult to crawl all wiki pages or create the web graph of the whole wikipages from a certain starting point.

4.1 Map reduce algorithm design

The mappers each takes input as a line containing $(link, docid)$ retrieved from the URL list-file, where $docid$ is the ID of the webpage represented by the $link$. Each mapper fetches the webpage of the $link$. The outputs of mappers are the collection of pairs $(docid, docid_i)$, where $docid_i$ is the ID of the webpage whose link is found from the fected page.

The reducers each takes an input as an output of mappers which have a collection of pairs $docid, docid_i$. Each reducer enumerate all adjacency $docid_i$, and output $docid$ with the collection of all $docid_i$.

The algorithm design is figured in table 1.

4.2 Creating web graph

4.2.1 Statistics of web graph

The following stat gives us some information about the created graph.

- The number of nodes: 500,000 (including nodes with outlink degree 0).

Table 1: MapReduce design for creating web graph from a given list of URLs

		Mappers		
$(link_1, docid_1)$	$\Rightarrow$	**Mapper**	$\Rightarrow$	$\{(docid_1, docid_{11}), (docid_1, docid_{12}), ...\}$
$(link_2, docid_2)$	$\Rightarrow$	**Mapper**	$\Rightarrow$	$\{(docid_2, docid_{21}), (docid_2, docid_{22}), ...\}$
$(link_3, docid_3)$	$\Rightarrow$	**Mapper**	$\Rightarrow$	$\{(docid_3, docid_{31}), (docid_3, docid_{32}), ...\}$
		Reducer		
$\{(docid_1, docid_{11}), (docid_1, docid_{12}, ...)\}$	$\Rightarrow$			$(docid_1, \{docid_{11}, docid_{12}, ...\})$
$\{(docid_2, docid_{21}), (docid_2, docid_{22}, ...)\}$	$\Rightarrow$	**Reducer**	$\Rightarrow$	$(docid_2, \{docid_{21}, docid_{22}, ...\})$
$\{(docid_3, docid_{31}), (docid_3, docid_{32}, ...)\}$	$\Rightarrow$	**Reducer**	$\Rightarrow$	$(docid_3, \{docid_{31}, docid_{32}, ...\})$

- The max degree: 1,789.
- The min degree: 0.
- Average degree: 68.
- The number of nodes having degree more than 200: 29762.
- The number of nodes having degree more than 500: 3835.

4.2.2 Statistics of computation performance

We run the hadoop program with the configuration which is specified 400 mapper tasks and 400 reducer tasks. The running time was about 2 minutes.

We also implemented a linear program to create the web graph and it took more than a day to finish.

5 Identify dense subgraphs from the created graph

5.1 Problem definitions

Given a large graph which is represented adjacency list, we are to find the strongly-connected clusters in the graph. Before coming up with approaching to solve the problem, we need to know several density definitions of a subgraph.

Let m be the number of edges in the subgraph, n be the number of nodes in the subgraph.

The naive definition of the subgraph is $\frac{m}{n}$. This definition has disadvantage from our interest view of finding strongly-connected subgraphs. We would prefer the subgraph of 10 nodes and 40 edges to the subgraph of $1,000,000$ nodes and $10,000,000$ edges. That formula does not work for us in this case.

The *clique* density definition should be fit to our interest. A *clique* subgraph is a fully-connected subgraph. Its density is defined as $\frac{2m}{n.(n-1)}$. We would mention that the problem of finding the largest subgraph in a graph with respect to *clique* density definition is a NP-hard problem.

Another density definition is "neighborhood similarity" which does have an abstract definition. The idea is originated from *clique* definition in which 2 nodes belonging to a clique share all nodes in the clique. In this "neighborhood similarity" definition, 2 nodes sharing many adjacency nodes have high probability of belonging to a dense subgraph. We use this definition in our work and evaluate result using the *clique* definition with the parameter 1.8 which is $\frac{2m}{n^{1.8}}$.

The diameter density of a subgraph is defined as its diameter in the subgraph. In this definition, the subgraph should be connected and the diameter is defined as the max distance of all pair distances between nodes in the subgraph. And problem is to find subgraphs with diamater less than a threshold.

5.2 Algorithms

In this section, we showed the algorithm proposed by David Gibson in VLDB 2005.

5.2.1 The shingling algorithm

As mentioned above, we use the "neighborhood similarity" to find clusters. We consider how to measure the similarity of 2 adjacency lists of 2 consided nodes.

A simple and natural measure of the similarity of two sets is the *Jaccard coefficient*, defined as the size of the intersection of the sets divided by the size of their union: $|A \cap B|/|A \cup B|$. Formally, if h is a random permutation of the elements in the ordered universe U from which A and B are drawn, then it can be shown that:

$$Pr_{h \in U}\{h^{-1}(min_{a \in A}(h(a))) = h^{-1}(min_{b \in B}(h(b)))\} = \frac{|A \cap B|}{|A \cup B|}$$

That is, the probability that the smallest element of A and B is the same, where smallest is defined by the permutation h, is exactly the similarity of the two sets according to the Jaccard coefficient. Using this observation, we compute the fingerprint of A by fixing a constant number c of permutations $h_1, ..., h_c$ of U, and producing a vector whose i-th element is $min_{a \in A} h_i(a)$. The similarity of two sets is then estimated to be the number of positions of their respective fingerprint vectors that agree.

This formulation is not yet sufficient for our needs; we require one generalization. The formulation as given may be viewed as follows: consider every one element set contained entirely in A or B, and measure agreement by the fraction of these one-element subsets that appear in both sets. Generalizing, we may instead consider every s-element set contained entirely within either set, and measure similarity by the fraction of these s-element subsets that appear in both. This is identical to measuring the similarity of A and B by computing the Jaccard coefficient of two sets , where $A_s = a_1, a_2, ..., a_s | a_i \in A$, and B_s is defined likewise. The same fingerprinting scheme applies unchanged to A_s and B_s. We will refer to each of the s-element subsets as a shingle, and to the algorithm that produces the set as an (s, c) shingling algorithm.

5.2.2 Algorithm of creating dense subgraphs in a graph

As designed in [1], we have 2 shingling steps.

Algorithm 1 $ShingleValues(\{x_1, x_2, ..., x_n\}, s, c)$

```
Input:   Set of values $\{x_1, x_2, ..., x_n\}$;
         $s$: number of "smallest" values collected per permutation.
         $c$: number of permutation functions.
Output:  Set of shingle values $\{z_1, z_2, ..., z_c\}$
Let $p$ be a large random prime.
// Permutation function $h_i(x) = (a_i * x + b_i) \mod p$.
Let $a_1, b_1, ..., a_c, b_c$ be random integers in $[1...p]$
Let $H$ be a hash function from strings to integers.
for $j = 1$ to $c$ do
  for $i = 1$ to $n$ do
    $X_i \leftarrow H(``x_i")$
    $Y_i \leftarrow (a_j * X_i + b_j) \mod p$
  end for
  Let $w_1, w_2, ..., w_s$ be $s$ minimum elements of $Y$'s
  $z_j \leftarrow H(Concate(w_1, w_2, \ldots, w_s))$
end for
return $\{z_1, z_2, ..., z_c\}$
```

In the first one, we applied the shingling algorithm to the set of adjacency nodes of each node. We would mention again that 2 nodes belonging a cluster have high probability of having many common shingle values. Then we create the inverted list of each shingle value containing all nodes whose their adjacency set each is applied to the shingling algorithm to generate that shingle value. For example, after shingling the adjacency list of *node*1, *node*2, we have the corresponding c shingle values $(s_{11}, s_{12}, s_{13}, ...s_{1c})$ for *node*1, and $(s_{21}, s_{22}, s_{23}, ...s_{2c})$ for *node*2. For easier explanation later, we assume $s_{11} = s_{21} = S1$, and $s_{13} = s_{22} = S2$. Following up, We have the inverted lists $S1 = (node1, node2, ...)$, and $S2 = (node1, node2, ...)$.

In the second shinging step, the inverted list of each shingle value is applied to shingling algorithm to generate the second-level shingle values. We also create the inverted list of each second-level shinge value containing all first-level shingle values whose their adjacency set each is applied to the shingling algorithm to generate that second-level shingle value. For example, the inverted lists of $S1$ and $S2$ are applied to generate $(x_{11}, x_{12}, x_{13}, ...x_{1c})$ for $S1$, and $(x_{21}, x_{22}, x_{23}, ...x_{2c})$ for $S2$, where x_{ij} are the second-level shingle values. Assuming $x_{11} = x_{21} = X1$ and $x_{12} = x_{22} = X2$, we have the inverted list of $X1$ as $(S1, S2, ...)$ (which are the first-level shingle values), and of $X2$ as $(S1, S2, ...)$.

After these 2 shingling steps, we consider that all first-level shingle values in each inverted list of each second-level shingle value are connected to each other. Here We accept indirect connectivity which means that 2 first-level shingle values may not in the same inverted list but may be connected indirectly. If A is connected to B, and B is connected to C, then so is A connected to C. We use *union-find* algorithm to find all connectivity groups. Mentioning that each first-level shingle value has an inverted list of graph nodes, and letting each inverted list as a subset, we union all subsets of all first-level shingle values in each connectivity group. The results of the unions are the dense clusters we look for.

For more detail, please take a look at the paper [1].

Table 2: MapReduce design for 2 steps shingling

Mappers				
$(node_1, adjacencylist_1)$	$\Rightarrow$	**Mapper**	$\Rightarrow$	$\{(S_{11}, node_1), (S_{12}, node_1), ...\}$
$(node_2, adjacencylist_2)$	$\Rightarrow$	**Mapper**	$\Rightarrow$	$\{(S_{21}, node_2), (S_{22}, node_2), ...\}$
$(node_3, adjacencylist_3)$	$\Rightarrow$	**Mapper**	$\Rightarrow$	$\{(S_{31}, node_3), (S_{32}, node_3), ...\}$
Reducer				
$\{(S_1, node_{11}), (S_1, node_{12}, ...)\}$	$\Rightarrow$			$(S_1, \{node_{11}, node_{12}, ...\})$
$\{(S_2, node_{21}), (S_2, node_{22}, ...)\}$	$\Rightarrow$	**Reducer**	$\Rightarrow$	$(S_2, \{node_{21}, node_{22}, ...\})$
$\{(S_3, node_{31}), (S_3, node_{32}, ...)\}$	$\Rightarrow$	**Reducer**	$\Rightarrow$	$(S_3, \{node_{31}, node_{32}, ...\})$

5.2.3 Map Reduce in algorithm

We proposed the MapReduce design which is best fit to the shingling algorithm. The first step shingling is to generate shingle values from the adjacency list of each node. We design the input of Mapper as the adjacency list. Mapper take the list, generate c shingle values, and return c pairs of $(shingle_i, N)$, where $shingle_i$ is the generated shingle value, and N is the current node.

Due to the 2 shingling steps look generic, we can apply the MapReduce approach 2 times. And the output of the first $MapReduce$ is the input of the later $MapReduce$.

The algorithm design is figured in table 2.

5.2.4 Optimization

We proved that, in the first shingling step, if the inverted list of shingle value contains only one element, that list can be removed without affecting losing any clusters. Because number of those inverted lists is significantly large, if we remove them, it efficiently improve the computation cost and storage. In our experiments, this optimization reduces the number of first-level shingles by 10 folds, and significantly reduce the storage and computation cost. This optimization also helps us eliminate "noise clusters" which are very small in size and are not interesting.

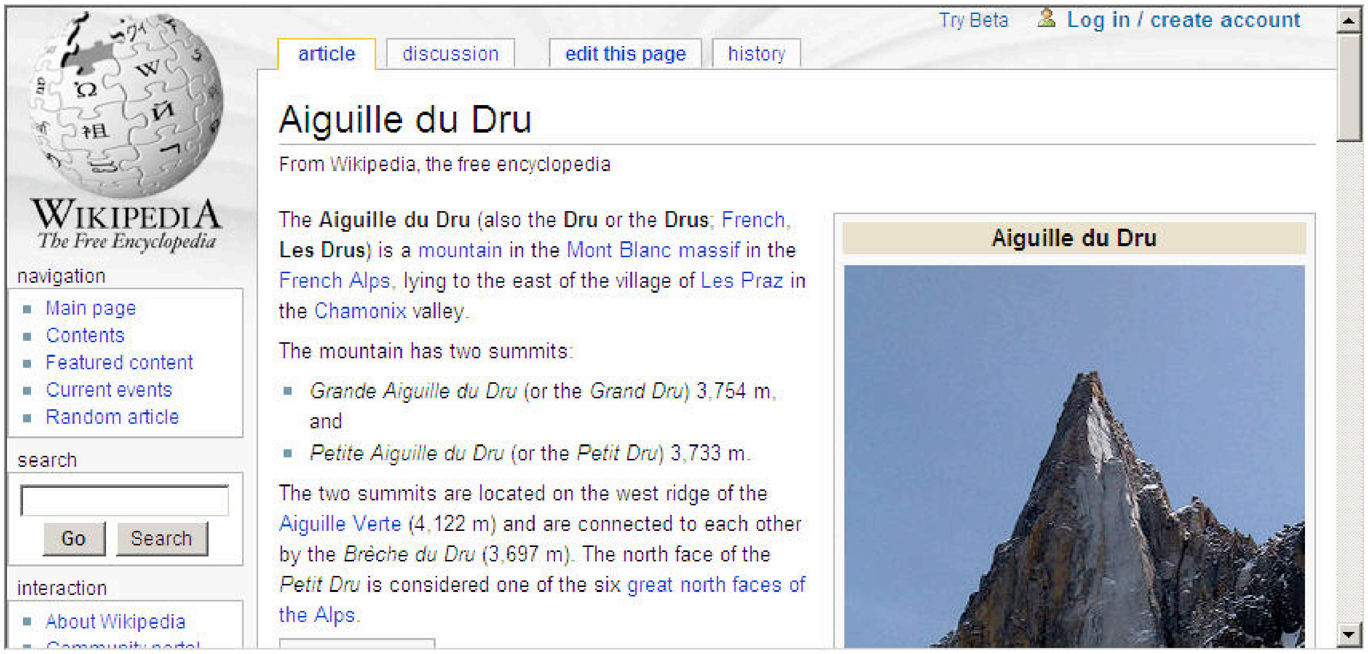
5.2.5 Results

We found 5617 raw clusters which have many clusters with size less than 10. They are in our interest. We put some constraints to filter for *good* clusters which are defined as the size greater than 10, and the density greater than a certain threshold. *Clique* density is used with parameter 1.8. With the threshold 0.3, we have found 408 clusters. The largest cluster in density has 320 nodes with 80157 edges. Average degree is 250 and the density is 2.48.

We manually verified the clusters and they look very interesting. Due to being very strongly-connected, pages in each clusters are common in some sense. They are about travel, finance, math, distribution, a group of papers, KDE open source, MS product, radio station. The largest cluster is about France.

The following pictures are shown as the demo of the largest cluster whose all pages almost are about France.

```
Aiguille_du_Dru French_Prealps Principal_passes_of_the_Alps Union_Internationale_des_Associations_d%27Alpinisme
Geneva_Cointrin_International_Airport Percy_Bysshe_Shelley Davos Bohemia_national_ice_hockey_team
Czechoslovakia_national_ice_hockey_team Abseiling James_Prescott_Joule Transmitter %C3%89vian-les-Bains Yvoire Thonon-
les-Bains Annemasse Christl_Haas Federico_Bahamontes Robert_Trent_Jones Ski_mountaineering Verbier
The_World_Is_Not_Enough Col_du_Galibier Rob_Gauntlett A42_autoroute La_Balme-de-Sillingy Choisy,_Haute-Savoie %C3%
89pagny,_Haute-Savoie Lovagny M%C3%A9signy Metz-Tessy Meythet Nonglard Poisy Sallen%C3%B4ves Sillingy Faucigny
Communes_of_the_Haute-Savoie_department Abondance,_Haute-Savoie Alby-sur-Ch%C3%A9ran Alex,_Haute-Savoie All%C3%A8ves
Allinges Allonzier-la-Caille Amancy Ambilly Andilly,_Haute-Savoie Annecy-le-Vieux Anthy-sur-L%C3%A9man Ar%C3%A2ches-la-
Frasse Arbusigny Archamps Arenthon Argonay Armoy,_Haute-Savoie Arthaz-Pont-Notre-Dame Aviernoz Ayse Ballaison La_Balme-
de-Thuy Bassy La_Baume Beaumont,_Haute-Savoie Bellevaux Bernex,_Haute-Savoie Le_Biot Bloye Bluffy,_Haute-Savoie Bo%C3%
ABge Bog%C3%A8ve Bonne,_Haute-Savoie Bonnevaux,_Haute-Savoie Bonneville,_Haute-Savoie Bons-en-Chablais Bossey Le_Bouchet
Brenthonne Brizon,_Haute-Savoie Burdignin Cercier Cernex Cervens Chainaz-les-Frasses Challonges Champanges La_Chapelle-
d%27Abondance La_Chapelle-Rambaud La_Chapelle-Saint-Maurice Chapeiry Charvonnex Ch%C3%A2tel,_Haute-Savoie Ch%C3%A2tillon
-sur-Cluses Chaumont,_Haute-Savoie Chavannaz Chavanod Ch%C3%AAne-en-Semine Ch%C3%AAnex Chens-sur-L%C3%A9man Chessenaz
Chevaline,_Haute-Savoie Chevenoz Chevrier Chilly,_Haute-Savoie Clarafond-Arcine Les_Clefs Clermont,_Haute-Savoie
La_Clusaz Cluses Collonges-sous-Sal%C3%A8ve Combloux Cons-Sainte-Colombe Contamine-Sarzin Contamine-sur-Arve Copponex
Cordon,_Haute-Savoie Cornier La_C%C3%B4te-d%27Arbroz Cran-Gevrier Cranves-Sales Crempigny-Bonnegu%C3%AAte Cruseilles Cusy
Cuvat Demi-Quartier Desingy Dingy-en-Vuache Dingy-Saint-Clair Domancy Doussard Douvaine Draillant Droisy,_Haute-Savoie
Duingt %C3%89loise Entremont,_Haute-Savoie Entrevernes Essert-Romand Etaux %C3%89tercy %C3%89vires Excenevex Faverges
Feig%C3%A8res Fessy F%C3%A9ternes Fillinges La_Forclaz Franclens Frangy Gaillard Les_Gets Giez,_Haute-Savoie Le_Grand-
Bornand Groisy Gruffy Hab%C3%A8re-Lullin Hab%C3%A8re-Poche Hauteville-sur-Fier H%C3%A9ry-sur-Alby Jonzier-%C3%89pagny
Juvigny,_Haute-Savoie Larringes Lathuile Leschaux Loisin Lornay Lucinges Lugrin Lullin Lully,_Haute-Savoie Lyaud Machilly
Magland Manigod Marcellaz Marcellaz-Albanais Margencel Marignier Marigny-Saint-Marcel
```



Aiguille du Dru

From Wikipedia, the free encyclopedia

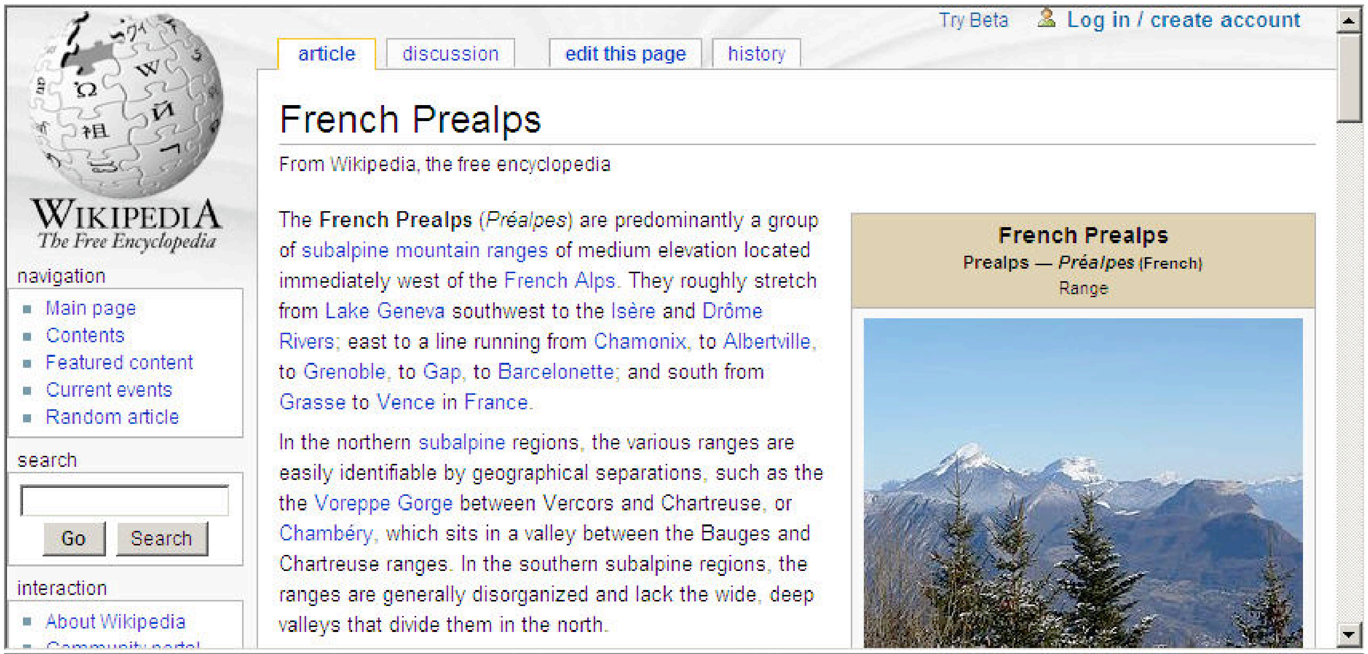
The **Aiguille du Dru** (also the **Dru** or the **Drus**; French, **Les Drus**) is a mountain in the Mont Blanc massif in the French Alps, lying to the east of the village of Les Praz in the Chamonix valley.

The mountain has two summits:

- *Grande Aiguille du Dru* (or the *Grand Dru*) 3,754 m, and
- *Petite Aiguille du Dru* (or the *Petit Dru*) 3,733 m.

The two summits are located on the west ridge of the Aiguille Verte (4,122 m) and are connected to each other by the *Brèche du Dru* (3,697 m). The north face of the *Petit Dru* is considered one of the six great north faces of the Alps.

Aiguille du Dru


Try Beta Log in / create account

article discussion edit this page history

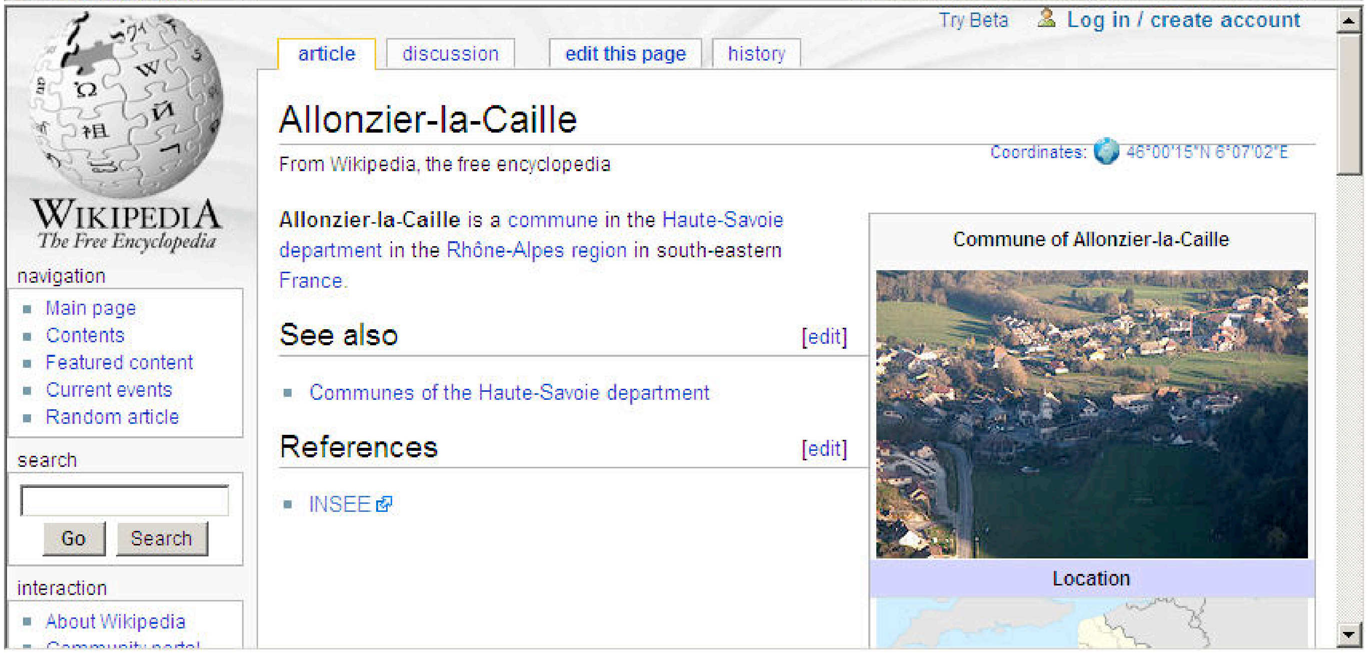
French Prealps

From Wikipedia, the free encyclopedia

The **French Prealps** (*Préalpes*) are predominantly a group of subalpine mountain ranges of medium elevation located immediately west of the French Alps. They roughly stretch from Lake Geneva southwest to the Isère and Drôme Rivers; east to a line running from Chamonix, to Albertville, to Grenoble, to Gap, to Barcelonette; and south from Grasse to Vence in France.

In the northern subalpine regions, the various ranges are easily identifiable by geographical separations, such as the the Voreppe Gorge between Vercors and Chartreuse, or Chambéry, which sits in a valley between the Bauges and Chartreuse ranges. In the southern subalpine regions, the ranges are generally disorganized and lack the wide, deep valleys that divide them in the north.

French Prealps
Prealps — ***Préalpes*** (French)
Range


Try Beta Log in / create account

article discussion edit this page history

Allonzier-la-Caille

From Wikipedia, the free encyclopedia

Coordinates: 46°00′15″N 6°07′02″E

Allonzier-la-Caille is a commune in the Haute-Savoie department in the Rhône-Alpes region in south-eastern France.

See also [edit]

- Communes of the Haute-Savoie department

References [edit]

- INSEE

Commune of Allonzier-la-Caille

Location

6 Conslusion

We have created the webgraph with respect to the top 500K wiki pages. We studied on the webgraph and implemented the algorithm of finding dense clusters in the graph. We designed a MapReduce approach which is best fit to the algorithm. I also proposed an optimization in the algorithm which speed up the program 10 times faster. The found results showed that our assumption about the strongly-connected clusters which have something in common is correct in some sense.

And we really enjoyed this assignment.

References

[1] David Gibson and Ravi Kumar and Andrew Tomkins, “Discovering Large Dense Subgraphs in Massive Graphs,” VLDB 2005