



# **Data Structures and Algorithms for Pretopology: the JAVA based software library PretopoLib**

Vincent Levorato, Marc Bui

## **► To cite this version:**

Vincent Levorato, Marc Bui. Data Structures and Algorithms for Pretopology: the JAVA based software library PretopoLib. (I2CS), Jun 2008, Fort de France, Martinique. pp.122-134. <hal-00460695>

**HAL Id: hal-00460695**

**<https://hal.science/hal-00460695v1>**

Submitted on 2 Mar 2010

**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.



HAL Authorization

# Data Structures and Algorithms for Pretopology: The JAVA based software library *PretopoLib*

Vincent Levorato and Marc Bui

Cognitions Humaine & ARTificielle

École Pratique des Hautes Études - Paris 8

41 rue G. Lussac, F-75005, Paris, France.

Email : {vincent.levorato, marc.bui}@ephe.sorbonne.fr

## Abstract

*PretopoLib is a library of data types and algorithms based on pretopology theory. Its interest is its data structures representing mathematical sets and all the operations related to them. The main features of the library are a clear separation of computation and visualization, efficient sets data types, and its ease of use to build model or application prototypes in the complex systems simulation framework.*

## 1: Introduction

Modeling complex networks is a complicated task. However, this is very important to find appropriate models to analyze the structure and the dynamics of these networks, as there are lot of scientific domains in which they are involved, like social sciences for instance [5]. Nowadays, there are some existing models, but there are not conclusive to represent all the dynamics of complex networks, as exposed in one of our previous work [7].

To model complex networks, we propose pretopology theory which is a mathematical modeling tool for the concept of proximity. It provides the powerful tools for the structure analysis and classification. It ensures the follow-up of the process's development of dilation, alliance, adherence, closed subset, acceptability.

On one hand, some authors with graph theory have provide a large amount of algorithms, while on the other hand nearly nothing exists in pretopology. Some works and programs have been done, but only for specific purposes, never in a general way [4] [6]. To build and implement models for specific problems, if researchers start from scratch each time they want to build a pretopological application, tools developed cannot be re-used for further purposes. Starting from these observations, it is clear that a need for a standard library with the data structures and algorithms of pretopology theory exists and can be used for differents kind of networks like in social sciences.

We started the project *PretopoLib* to build a small, but growing library of data types and algorithms in a form which allows them to be used by non-experts and to elaborate simulations using pretopology. This library is splitted into two distinct parts:

- the process engine
- the visualization part

This library can be used to develop models prototypes for researchers familiar with pretopology concepts, but also as a teaching tool useful for students. We are able to graphically represent networks differently compared to actual graphs visualizations and new results on known structures taken from the web communities like emails, forums or blogs. This paper presents our contribution with the realisation of the library which main features are:

- Data types and data structures based on pretopology entities. The data types currently available are mathematical set, element, relation between singletons, link, and ball. The most difficult decision which we faced, was the choice of implementation for sets. For the efficiency of many data structures it is crucial that some operations may have one of the best possible complexity.
- A comfortable data type `PretopoNodes` (mathematical set). Standards operations on sets are implemented like union and intersection. It is possible to add and delete elements, to test if an element is contained, offering hash table design for low complexity. The data type `PretopoNodes` permits to write programs for pretopological problems in a form close to the way the algorithms are usually presented in text articles.
- Ease of use and portability: `PretopoLib` is written in JAVA. All data types and algorithms are stored in a JAR library. All examples given in this paper show their executable codes.
- A visualization module easy to set up for quick rendering application based on the *Prefuse* package (<http://www.prefuse.org>) for fast and good visual rendering.

This paper is structured as follows: in section 2, we introduce pretopology theory concepts, in section 3, we explain which data structures we used in the library, in section 4, we show how visualization is set and we end on the section 5 by samples applications followed by a conclusion.

## 2: Pretopology Concepts

Pretopology permits to define what is a complex network in a general way, providing a specification for elements, groups of elements and their evolution to model the dynamics of the network.

We introduce in this section pretopology concepts basing our network analysis on the question *what is the definition of a network* with a pretopological approach (cf. [2]).

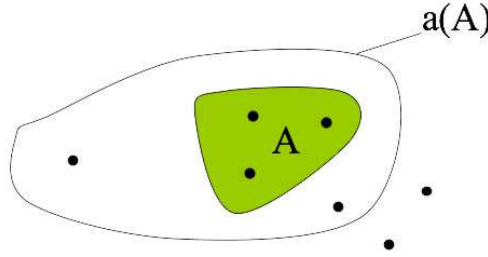
Let us consider a non-empty finite set  $E$ , and  $\mathcal{P}(E)$  which designates all of the subsets of  $E$ .

## 2.1: Pretopological space

**Definition 1** A pretopological space is a pair  $(E, a)$  where  $a$  is a map  $a(.) : \mathcal{P}(E) \rightarrow \mathcal{P}(E)$  named pseudo-closure (Fig. 1) and defined as follows :  $\forall A, A \subseteq E$  the pseudo-closure of  $A$ ,  $a(A) \subseteq E$  such that :

- $a(\emptyset) = \emptyset$  ( $P_1$ )
- $A \subseteq a(A)$  ( $P_2$ )

The pseudo-closure is associated to the *dilation process*. Thus,  $a(.)$  can be applied on a set  $A$  in sequence, to model expansions :  $A \subset a(A) \subset a^2(A) \subset \dots$ . That means we could follow the process *step by step*. Using the pseudoclosure, we model the proximity concept, very useful for aggregation process and to define the neighbourhood of a set or an element.



**Figure 1. Pseudoclosure of A**

Pretopology theory defines several space types associated to a corpus of known results [1], and the library implements some of these results as a set of basic operations.

## 2.2: $\mathcal{V}$ Pretopological space

**Definition 2** A  $\mathcal{V}$  pretopological space  $(E, a)$  is defined by :

- $\forall A, B, A \subseteq E, B \subseteq E$  and  $A \subset B$  then  $a(A) \subset a(B)$

This type of space is very powerful to describe complex properties but is based on weak assumptions.

## 2.3: $\mathcal{V}_D$ Pretopological space

**Definition 3** A  $\mathcal{V}_D$  pretopological space  $(E, a)$  is defined by :  $\forall A, B, A \subseteq E, B \subseteq E$   
 $a(A \cup B) = a(A) \cup a(B)$

Concerning the space properties, the library propose a generic framework letting the possibility for the user to redefine all characteristics space.

## 2.4: $\mathcal{V}_S$ Pretopological space

**Definition 4** A  $\mathcal{V}_S$  pretopological space  $(E, a)$  is defined by :

$$\forall A, A \subseteq E, a(A) = \bigcup_{x \in A} a(\{x\})$$

The library can be optimized depending of the space used, especially for  $\mathcal{V}_S$  pretopological space which is finite.

When transformation doesn't evolve anymore, we obtain closure, which is mainly used by classification algorithms.

## 2.5: Closure

**Definition 5**  $A \in \mathcal{P}(E)$  is closed if and only if :  $A = a(A)$

The closure of  $A \in \mathcal{P}(E)$  is the smallest closed subset containing  $A$ , noted  $F(A)$  or  $F_A$ .

In pretopology, spaces can be of different natures: metric, binary, or valued.

### 2.5.1: Metric space

$E$  has a metric, and a closed ball with a center  $x$  and a radius  $r$  is defined as follows:

$$B(x, r) = \{y \in E, d(x, y) \leq r\}$$

Pseudoclosure is defined on  $E$ :

$$\forall A \in \mathcal{P}(E), a(A) = \{x \in E, B(x, r) \cap A \neq \emptyset\}$$

### 2.5.2: Binary space

Elements of  $E$  are linked by  $n$  binary reflexives relations  $R_i$  with  $i \in \{1, \dots, n\}$ .

$$\forall i \in \{1, \dots, n\}, R_i(x) = \{y \in E, xR_iy\}$$

Pseudoclosure is defined on  $E$ :

$$\forall A \in \mathcal{P}(E), a(A) = \{x \in E, \forall i \in \{1, \dots, n\} R_i(x) \cap A \neq \emptyset\}$$

### 2.5.3: Valued space

Elements of  $E$  are linked by valued relations (graph theory extension). Let  $v: E \times E \rightarrow \mathbb{N}$ ,  $(x, y) \rightarrow v(x, y)$ .

Pseudoclosure is defined on  $E$ :

$$\forall A \in \mathcal{P}(E), a(A) = \{y \in E - A, \sum_{x \in A} v(x, y) \geq s\} \cup A$$

## 2.6: Network definition

In this paper, we focus on complex networks. In pretopology, we can define a *network* as a *family of pretopologies on a given set*  $E$  [2].

**Definition 6** *Let  $X$  be a set.*

*Let  $I$  a countable family of indexes.*

*Let  $\{a_i, i \in I\}$  a family of pretopologies on  $X$ .*

*The family of pretopological spaces  $\{(X, a_i), i \in I\}$  defines a network on  $X$ .*

Applications based on this definition are presented in section 5.

## 3: Data Types

Having introduced network formalism with pretopology, we propose to implement this framework which can be compared to graph theory (like LEDA [8]).

PretopoLib is written in JAVA language which is using a virtual machine to launch program, so we don't have to recompiled code to pass from one OS platform to another (like passing from Windows to Linux or MacOS). JAVA provides data abstraction, object based programming, operator overloading and many other features suitable for implementing data types. For details on the JAVA language see Sun documentation or a book like *Thinking in Java* [3].

A significant improvement would result, if data types we provide were made available as "package" by some users and could then be reused by others. Of course, this requires to specify the properties of the data type independently of the implementation. The package *PretopoNotions* deals with the processing part and *PretopoVisual* is used for the visual part, which be discussed in next section. Let's describe the PretopoLib specifications of the data types PretopoNodes, PretopoEdges and PretopoSpace.

### 3.1: PretopoNodes

#### 3.1.1: Description

A PretopoNodes object represents a set of elements in a pretopological space. Each element of the set is unique, using the *name* of the element as primary identifier. The name of each element (called *label*) is used in a hash table to find elements quickly into a set. Listing 1 shows a small illustration of PretopoLib's set module.

**Listing 1. Creating two sets of respectively three and four elements, then making their intersection.**

```
1 import PretopoNotions.Elm;
2 import PretopoNotions.PretopoNodes;
3
4 public class PretopoLibSample {
```

```

5
6   public static void main(String [] args)
7   {
8
9       //creating elements
10      Elmt elt_a=new Elmt("a");
11      Elmt elt_b=new Elmt("b");
12      Elmt elt_c=new Elmt("c");
13      Elmt elt_d=new Elmt("d");
14      Elmt elt_e=new Elmt("e");
15
16      PretopoNodes A=new PretopoNodes();
17      PretopoNodes B=new PretopoNodes();
18      PretopoNodes C=new PretopoNodes();
19
20      //filling sets
21      A.addElmt(elt_a);
22      A.addElmt(elt_b);
23      A.addElmt(elt_c);
24
25      B.addElmt(elt_b);
26      B.addElmt(elt_c);
27      B.addElmt(elt_d);
28      B.addElmt(elt_e);
29
30      C=A.intersection(B);
31  }
32 }

```

The program fills two sets A and B with different elements and put the result of their intersection in set C. Lines (1) and (2) import PretopoLib class packages. From line (9) to (13), we create elements (singletons) with unique label ; lines (15) to (17) create sets (A, B, and C), and lines from (20) to (27) fill sets A and B. In line (29), we finally make the intersection of A and B, storing the result in C.

### 3.1.2: Declaration of a PretopoNodes

Constructors are:

`public PretopoNodes()` which creates an empty set.

`public PretopoNodes(Elmt elt)` which creates a set with only one element.

`public PretopoNodes(PretopoNodes pn)` which creates a set by copy.

### 3.1.3: Operations on a PretopoNodes

Here are the most common operations:

`boolean contains(Object o)` tests if an element is in the set.

`boolean equals(Object o)` tests if two sets contains the same elements.

`void addElmt(Elmt elmt)` add one element into the set if this one is not in this set already.

Operation	List	HashTable
<i>contains</i>	$O(n)$	$O(1)$
<i>equals</i>	$O(n^2)$	$O(n)$
<i>addElmt</i>	$O(n)$	$O(1)$
<i>intersection</i>	$O(n^2)$	$O(n)$
<i>union</i>	$O(n^2)$	$O(n)$
<i>subtract</i>	$O(n^2)$	$O(n)$
<i>isSubset</i>	$O(n^2)$	$O(n)$

**Table 1. Simple operations costs**

`boolean intersects(PretopoNodes pn)` tests if a given set intersects the concerned set.

`PretopoNodes intersection(PretopoNodes pn)` gives the result set after the intersection operation between two sets.

`PretopoNodes union(PretopoNodes pn)` gives the result set of the union of two sets.

`PretopoNodes subtract(PretopoNodes pn)` gives a new set by removing all the elements of a given set in parameter.

`boolean isSubset(PretopoNodes pn)` tests is a set is a subset of a given set.

### 3.1.4: Implementations

There are several implementations of mathematical sets. One possible implementation is based on lists. The problem with that choice is the complexity of the standard operations introduced in the previous paragraph.

We propose an efficient data structure called *HashSet* in JAVA based on a hash table. In the table 3.1.4, operations costs in the worst case are detailed. All elements are *unordered*.

As shown is the table 3.1.4, for all operations, the gain of complexity is appreciable, which is quite important on large datasets.

## 3.2: PretopoEdges

This class represents all the possible relations of a pretopological space. It contains a set of relations, which can be of different type (metric, valued, or binary). It contains a hashtable of all the relations between the elements of a pretopological space. A relation object defines all the links between the elements and more important, it defines the *neighbourhood function* used for building the *pseudoclosure* process. In listing 2, using the previous variables of listing 1, we first build a binary relation, then another relation which is valued and a last one which is metric:

### Listing 2. Creating a binary, a valued and a metric relation.

```

1 //creating a binary relation
2 Relation r= new Relation("binary");
3 r.addLink(elt_c , elt_d);
4 r.addLink(elt_e , elt_d);
5 r.addSymLink(elt_c , elt_a);

```



```

6 r.addLink(elt_b , elt_a );
7
8 //creating a valued relation
9 RelationV rv= new RelationV("valued");
10 rv.addSymLink(elt_a , elt_e ,2);
11 rv.addSymLink(elt_b , elt_e ,1);
12 rv.addSymLink(elt_b , elt_d ,4);
13
14 //creating a metric relation
15 RelationB rb= new RelationB("metric");

```

In line (2), a binary relation is created, which is the basic relation class. From line (3) to (6), links are added between elements, sometimes in an oriented way `addLink( source , target )` and sometimes in an unoriented way `addSymLink( source , target )`. In line (9), a valued relation is created, and from line (10) to line (12), valued links are added. In line (15), a metric relation is created. Note that no links are used for this last relation because it's using directly the coordinates of each elements to know if they are close each other or not (with a given radius using ball concept).

The neighbourhood function is implemented in *Relation* class. Here is how a neighbourhood function like  $R(x) = x \in E, xRy$  is coded into the *Relation* class.

### Listing 3. Implemented neighbourhood function.

```

1 public boolean NeighbFunction(Elmt elt , PretopoNodes A)
2 {
3     boolean b=false;
4
5     ArrayList<Link> l=(ArrayList<Link>) rel_list.get(elt);
6     if(l!=null)
7         for(int i=0;i<l.size();i++)
8             if(A.contains(l.get(i).target))
9                 b=true;
10
11
12     return b;
13 }

```

The listing 3 shows how the algorithm is working: it examines the table of list of links between the element given in parameter ( line (5) ) and checks if the tested element has a link with an element of A, the set on which we are applying the pseudoclosure on line(8). It returns true if the element actually is into the neighbourhood of A, false else ( line (12) ).

### 3.3: PretopoSpace

This class represents a pretopological space itself. It is composed of a set of elements and a set of relations. It contains the pseudoclosure method and all the algorithms which are relying on it, like closure or open set. Three types of pretopological spaces are usually manipulated. By default, binary space, valued space and metric space are supported. One

of the fundamental operation in the library is the pseudoclosure.

**Listing 4. Part of the pseudoclosure code concerning the valued space type.**

```

1 public PretopoNodes pseudoclosure(PretopoNodes set, String rel_label, Object param)
2 {
3
4 PretopoNodes pseudoCset=new PretopoNodes(set);
5 PretopoNodes E=new PretopoNodes(Nodes);
6 E=E.subtract(set);
7
8     ...
9
10 if(param.getClass().equals(Double.class)) //valued relations space
11 {
12     RelationV r=(RelationV) Relations.getRelation(rel_label);
13     Iterator<Elmt> itr=(Iterator<Elmt>) E.getNodeSet().iterator();
14     while(itr.hasNext())
15     {
16         Elmt elt=itr.next();
17         if(r.NeighbFunction(elt,set,(Double)param))
18             pseudoCset.addElmt(elt);
19     }
20 }
21
22     ...
23
24 return pseudoCset;
25 }

```

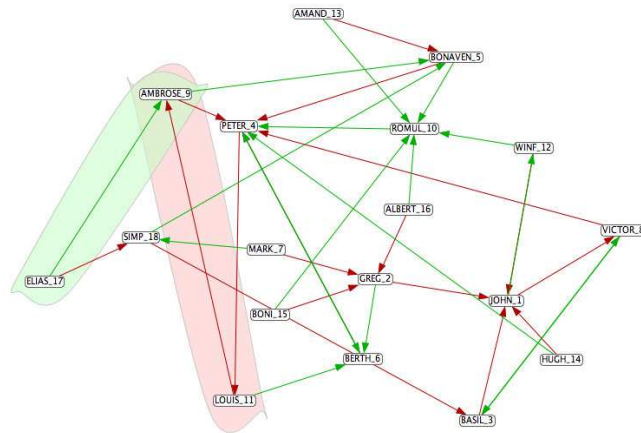
In the listing 4, we can see the generic overload flexibility of the implementation. It takes three input parameters: the set on which we want to apply the pseudoclosure, the name of the relation, and a parameter used for the neighbourhood function. In this listing, only the part about the valued space is given as example. On line(12), we extract the desired relation, and we look through E the elements which are neighbours of the given set ( lines(13) to (19) ). Finally, it returns the set representing the pseudoclosure of the given set line(24). They are several interesting characteristics with the library implementation:

- It is possible to constitute neighbourhoods and groups.
- Inter-relations can be visualized.
- User can interact by manipulating elements and groups.

## 4: Visualization

The Visualization Part is based on *Prefuse*: it has shown some very good visual rendering results. Being used to represent graphs, it also allows us to represent convex polygons that we use to represent sets.

To display a pretopological space, we have to use the object *PretopoVisualization*. One can see how to make a display window into a *JPanel* JAVA structure (display window).



**Figure 2. Visualization sample**

**Listing 5. Building of a visualization.**

```

1 PretopoVisualization pv = new PretopoVisualization(ps.getNodeSet(), rel,
2 PretopoVisual.PretopoVisualization.NORMAL);
3
4 JPanelDisplay.add(pv);

```

In listing 5 line(1) the PretopoVisualization object is created based on the set E of the pretopological space, a list of relations and a constant allowing the user to choose the type of visualization:

- *NORMAL* : Standard radial view
- *GRID*: Grid view
- *RANDOM*: Random view
- *FIXED*: Fix the elements using their coordinates

Line (4), we only have to add the PretopoVisualization object to a JPanel for displaying it. The figure 2 shows an example of display.

Useful functions are incorporated into PretopoVisualization class:

`void hidelinks()` hides links between elements.

`void showlinks()` shows links between elements.

`void rearrange_display(int type)` rearranges display view with a chosen type: NORMAL, GRID, RANDOM, or FIXED.

`void resetgroup()` remove all the visual sets.

`void setgroup(java.util.HashSet <java.lang.String> group_nodes, int igrp)` creates a visual set of elements using a palette color.

## 5: Examples of application

To illustrate the use of our library, let us consider two samples of data. The first is about a community of monks studied by the sociologist Samuel F. Sampson [10], who described the various social relations among the novices who were preparing to enter the clerical orders. Although small, this network has a large number of different relationships, which enable us to test the various features of our tool. The second sample, from Valdis Krebs (<http://www.orgnet.com/>) concerns the Web which is describing a network of books on American policy sold on Amazon.com. The relationship between books represent co-purchase done by the same customers, as indicated on the site by *"people who bought this product also purchased."*

The results in Fig. 2 show a pseudoclosure on an element (in this case AMBROSE\_9), with green relations representing friendship and red relations representing esteem. Here, the threshold of the neighbourhood is set in such a way that only those who have the greater esteem and friendship are concerned. Regardless of relations, the final set which is the pseudoclosure of {AMBROSE\_9} in this case is equal to {AMBROSE\_9 LOUIS\_11 ELIAS\_17}.

Using visualization facilities of the library, we show that the user can move the elements in his own way, as well as sets while retaining the coherence of visualization.

For the second sample data, there is only one type of binary relation. However, even if there are many more elements (105) and connections (441), it is a small network, nevertheless sufficient for the demonstration. In this case, we use one of the provided algorithm of the library to isolate the group which is locally most connected. This group of books are client links to discover all the other books, and moreover, the group is the most popular books. Once identified, it has become clear that the group consists of 38 books and has a degree of 164, contains a large number of books dealing which is a hot topic of discussion in the time period the experiment was conducted.

Concerning visualization, if display becomes fuzzy even after we have extracted components out of other elements (Fig. 3), simply hide the relationship to clarify the display.

## 6: Conclusion

PretopoLib is a library of efficient data types and algorithms. At present, its strength relies on pretopological algorithms and the data structures related to them, which continue to evolve. This is the first project which aim for the goals described in the introduction. The advantages of PretopoLib are indeed:

- a clear separation between notions and visualization
- a straight forward implementation of the specification of the algorithms
- the inclusion of many data structures and algorithms



## References

- [1] Z. Belmandt. *Manuel de prétopologie et ses applications: Sciences humaines et sociales, réseaux, jeux, reconnaissance des formes, processus et modèles, classification, imagerie, mathématiques*. Hermes Sciences Publications, 1993.
- [2] Monique Dalud-Vincent. *Modèle prétopologique pour une méthodologie d'analyse des réseaux: concepts et algorithmes*. PhD thesis, Université Claude Bernard - Lyon 1, Lyon, 1994.
- [3] Bruce Eckel. *Thinking in Java*. Prentice Hall PTR, 1998.
- [4] Christine Largeron and Stéphane Bonnevey. A pretopological approach for structural analysis. *Information Sciences*, 144:169 – 185, 2002.
- [5] Emmanuel Lazega. *Réseaux sociaux et structures relationnelles*. Presses Universitaires de France, Versailles, 1998.
- [6] Thanh Van Le, Nadia Kabachi, and Michel Lamure. A new clustering method associated pretopological concepts and k-means algorithm. In *Proceedings of the International Conference on Research Innovation and Vision for the Future*, Hanoi, 2007.
- [7] Vincent Levorato and Marc Bui. Modeling the complex dynamics of distributed communities of the web with pretopology. In *Proceedings of the 7th International Conference on Innovative Internet Community Systems*, Munich, Germany, 2007.
- [8] Kurt Mehlhorn and Stefan Näher. Leda : A library of efficient data types and algorithms. *GI Jahrestagung*, 1:35–39, 1989.
- [9] Mark Newman, Albert-László Barabási, and Duncan J. Watts. *The Structure and Dynamics of Networks*. Princeton University Press, 2006.
- [10] S.F. Sampson. *A Novitiate in a Period of Change. An Experimental and Case Study of Social Relationships*. PhD thesis, Cornell University, 1968.