# *On algorithms for construction line diagrams of concept lattices and the set of all concepts*

Sergey A. Yevtushenko, August 7, 2001

Scientific advisor – Prof. Dr. Tatiana Taran

# Outline of a talk

- Formal Concept Analysis (reference information)
- Motivation
- Algorithm description
- Comparison with other algorithms
- Software system “Concept Explorer”

# Objectives of master's thesis

- to develop algorithms for calculation set of all concepts, construction of Hasse diagrams and their visualization, finding a base of implications, which holds in context
- to explore computational complexity of developed algorithms
- to develop a software, which implements aforementioned functionality

# Formal Concept Analysis (FCA)

Was proposed by Rudolf Wille in 1981 and actively developed from this time, mainly by Darmstadt research group on Formal Concept Analysis

FCA is based on the theorem of Gareth Birkhoff, that from each binary relation complete lattice can be yielded.

# FCA – basic notions

- Context – triple (*G*, *M*, *I*), where *G* – set of objects, *M* – set of attributes, $I \subseteq G \times M$ - incidence relation

  $gIm \Leftrightarrow$ object g has attribute m

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | X | | | | | | |
| 2 | X | X | | | | | |
| 3 | X | X | X | | | | |
| 4 | X | X | | X | | | |
| 5 | X | X | X | | X | | |
| 6 | X | X | X | X | | X | |
| 7 | X | X | X | X | X | X | X |

# FCA – basic notions

- Derivation operators
  - $A'=\{ m \mid \forall g \in A \; gIm\}, A \subseteq G$
  - $B'=\{ g \mid \forall g \in B \; gIm\}, B \subseteq M$

  Operators $A''$ and $B''$ are closure operators

- Formal concept of context (G, M, I) is a pair (A, B), where $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$.
  Set A is called **extent** and set B **intent** of concept (A, B).
  Also concept can be denoted as $(B', B'')$ or $(A'', A')$

- Set of all concepts of context (G, M, I) is denoted $\underline{B}(G,M,I)$

# FCA – basic notions

- One part of basic theorem of FCA tells, that <u>B</u>(G,M,I) is a complete lattice, where infinum and supremum are defined as

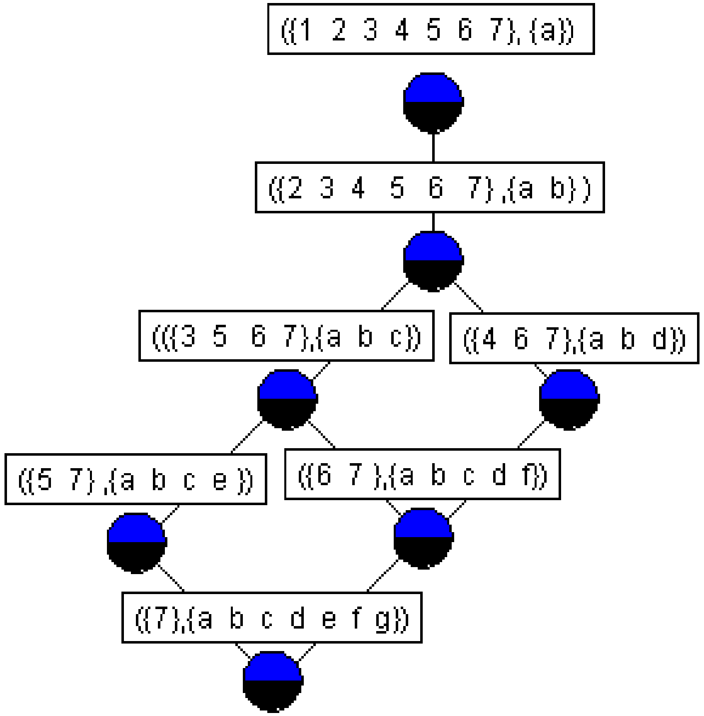
$$\bigwedge_{t \in T} (A_t, B_t) = \left( \bigcap_{t \in T} A_t , \left( \bigcup_{t \in T} B_t \right)'' \right)$$

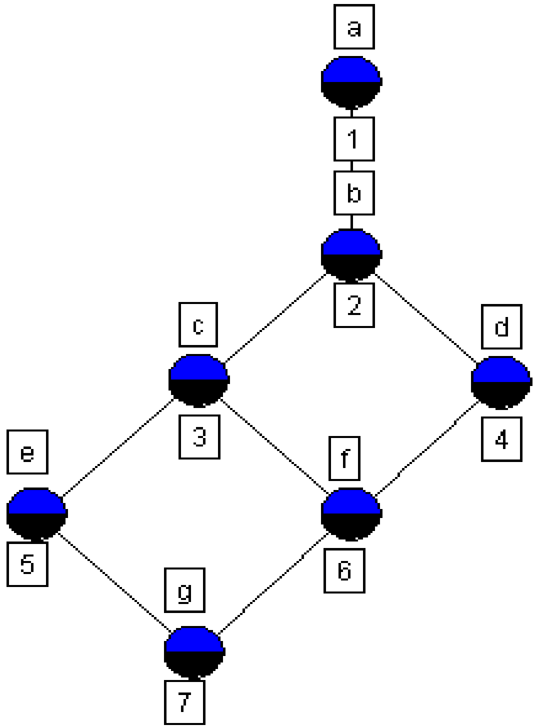
$$\bigvee_{t \in T} (A_t, B_t) = \left( \left( \bigcup_{t \in T} A_t \right)'' , \bigcap_{t \in T} B_t \right)$$

- Between concepts partial order is defined

$$(A_1, B_1) \leq (A_2, B_2), \text{ if } A_1 \subseteq A_2 \Leftrightarrow B_1 \supseteq B_2$$

# FCA – basic notions



# Motivation

Why do we need algorithms for finding set of concepts of context?

Applications:

- Classification
- Data mining ( JSM method, associations rules, …)
- Conceptual information systems
- Information retrieval systems
- …

Why do we need to develop as efficient algorithms, as possible ?

Size of concept lattice in worst case – $2^{|G|}$ for context $(G, G, \neq)$

# Motivation

In many applications of FCA line diagrams have *primary importance* and set of concepts only secondary one.

– Conceptual information systems
– From pruned line diagram rule base for approximate associations rules can be easily extracted

So, we also need efficient algorithms for construction of line diagrams

# Top-down approach for construction of Hasse diagram

```
Find unit element (G, G’) of concept lattice
if (G, G’) ≠ (M’, M)
    FindPredecessors( (G, G’) )

 FindPredecessors((A’, A”))
   Find set of Lower Neighbours of (A’, A”)
   for each Curr  ∈ Lower Neighbours
           if Curr wasn’t calculated earlier, then
                FindPredecessors(Curr)
           else
               Curr = findLatticeNode(Curr)
           Connect((A’, A”), Curr)
```

# Crucial operations

- Determination, that concept $(A_2,B_2)$ is a direct predecessor of a concept $(A_1, B_1)$
- Determination, whether concept was calculated earlier
- Finding earlier calculated concept

# Algorithm of Tkachev I

```
C=(G,G′)
BuildLattice((G′, G))

BuildLattice((A′, A″))
  if A″= M return
  Desc:= (∪{gᴵ | g∈A′})\A″// properties of descendants
  if Desc=∅
      Connect((A′, A″), (M′,M)
      return
  LN = FindLowerNeigbours((A′, A″), Desc)
  for each (B′,B″) ∈ LN
      if (B′,B″) ∉ C
          C:= C ∪ (B′,B″)
          BuildLattice((B′,B″))
      Connect((A′, A″), (B′,B″))
```

# Algorithm of Tkachev II

```
FindLowerNeigbours((A′, A″), Desc)
LN = ∅
for each m ∈ Desc
      Extent := { g | m∈gᴵ & g ∈A′}
      Intent := ∩{gᴵ | m∈gᴵ & g ∈A′}
      if Intent\A″= {m} or Extent={g|Intent∩gᴵ≠∅ & g∈A′}
          LN := LN∪(Extent, Intent)
          Desc:=Desc\Intent
      else
          Desc:=Desc\m
```

# Basic ideas of new algorithm

*Determination of direct predecessors*

Suppose, that that procedure was called for some concept (A’, A”). We want to define, whether concept ((A”$\cup${m})’, (A”$\cup${m})”) is a direct predecessor of (A’, A”).

This mean, that doesn’t exists such n, that

$$A'' \subseteq (A'' \cup \{n\})'' \subseteq (A'' \cup \{m\})'' \Leftrightarrow (A'' \cup \{m\})' \subseteq (A'' \cup \{n\})' \subseteq A'$$

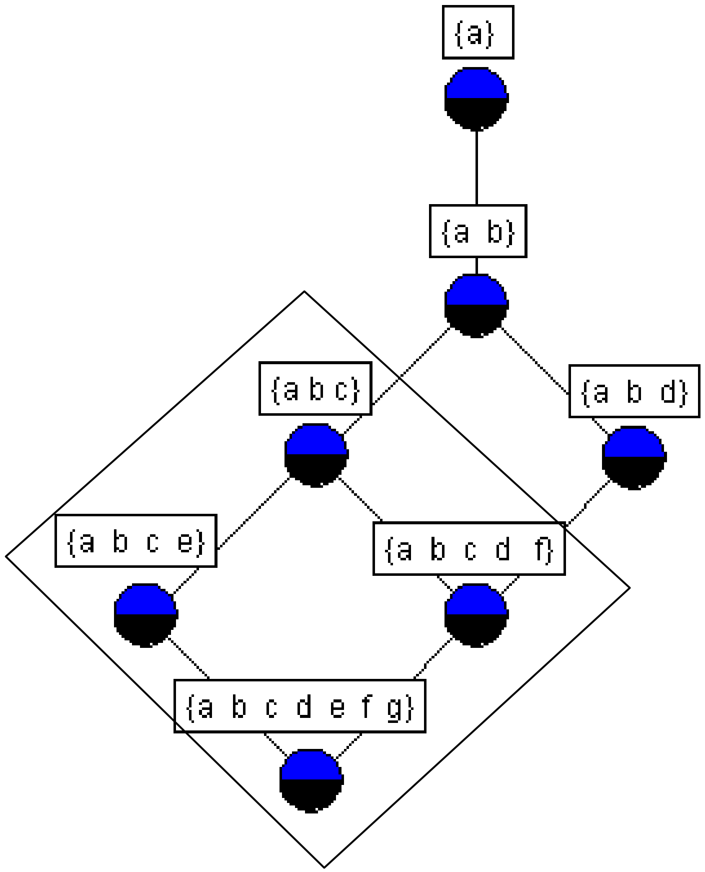
and finally we have

$$((\cup\{g^I | g \in A' \backslash (A'' \cup \{m\})'\}) \cap (A'' \cup \{m\})'') \backslash A'' = \varnothing$$

Lets mark set $\cup\{g^I | g \in A' \backslash (A'' \cup \{m\})'\} \Leftrightarrow \cup\{g^I | m \notin g^I \ \& \ g \in A'\}$ as Outer[m]

# Basic ideas of new algorithm

*Determination, whether concept was calculated earlier*



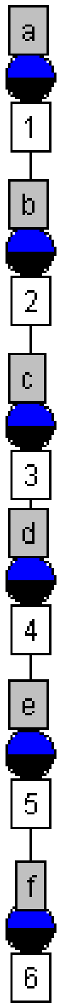

Let current concept has intent {a, b}.

Suppose, that we call procedure ***FindPredecessors*** for lower concept with intent {a, b, c}.

Than procedure will not return, till all concepts from principal ideal of {a, b, c} will be computed $\Rightarrow$ all concepts from principal ideal has {c} in their intents. So after call to ***FindPredecessors(…,*** *{a, b, c}* ***)*** , for all concepts, which are descendants of {a,b} we can use presence of {c} as indicator, that concept already was calculated

# Basic ideas of new algorithm

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| 1 | X | | | | | |
| 2 | X | X | | | | |
| 3 | X | X | X | | | |
| 4 | X | X | X | X | | |
| 5 | X | X | X | X | X | |
| 6 | X | X | X | X | X | X |




Old algorithm is inefficient on a chain.

Reasons:

- calculation of all concepts with intent of kind $(A'' \cup \{m\})''$ when calculating sets of direct predecessors
- Movement from concept to concept only by edges of Hasse diagram
- No mechanism of using information about relations between different attributes, which are defined during calculation

Cure:

- Set Outer can be used for determination of relation between attributes – if for some attribute m attribute n $\notin$ Outer[m] $\Rightarrow$
  $$(A'' \cup \{m\})'' \subset (A'' \cup \{n\})''$$
- Allow to move not only by edges of Hasse diagram ( in our version, used only for construction of concept set )

# Basic ideas of new algorithm

*Finding earlier calculated concept*

Two strategies can be applied, depending on memory requirements:

1. Using already generated part of concept lattice for search (can be realised with complexity - $O(m^2)$)
2. Storing for each nodes predecessors(not only direct), which were generated for the first time during call of procedure *FindPredecessors*, in a tree structure – search can be realized by $O(m)$ operations. Drawback – additional memory requirements for storing a tree.

# Algorithm (for calculating concept set)

```
C=(G,G′)
if M≠G′
   CalcPredConcepts((G, G′), ∅)

CalcPredConcepts((A′, A″), Prohibited)
   Prohibited = Prohibited ∪ A″
   for each m∈ M\Prohibited
        NewIntent = M ∩{∩gᴵ|g∈A′ & m∈gᴵ}
        NewExtent = {g|g∈A′ & m∈gᴵ}
        Outer = {∪gᴵ | g∈A′ & m∉gᴵ}
        if (NewIntent∩Prohibited)\A″=∅
              C=C ∪ (NewExtent, NewIntent)
              CalcPredConcepts((NewExtent, NewIntent), Prohibited)
              Prohibited = Prohibited ∪ (M\Outer)
```

# Theoretical complexity of algorithm

Theoretically possible to achieve complexity of algorithm – $O((m+n)nH)$ operations

(S. Kuznetsov)

Complexity of algorithm in the worst case is $O(m^2 nH)$, where $m=|M|$, $n=|G|$, $H=|B(G,M,I)|$

Amount of memory needed for a work of algorithm is $O(m(n+m))$

# Algorithms for finding set of concept

There are a lot of algorithms for calculating set of concept / construction of line diagram

- Chein
- Norris
- Close by One (Kuznetsov)
- Next Closed Set (Ganter)
- Bordat
- Godin
- Lindig
- Nourine & Raynaud
- Titanic (Stumme,Taouil, Pasquier, Lakhal, Bastide)
- …

# Main properties of algorithms

- Calculation strategy – batch or incremental
- Method of generation of new concepts
- Method of checking of earlier generation of concept

# Properties of algorithms

| | Calculation strategy | Checking earlier generation of intents | Method of calculating new concepts |
|---|---|---|---|
| Ganter | Batch | Lexical order | Intersecting objects intents |
| Norris | Incremental | Set of earlier added objects | Intersecting object with earlier generated concepts |
| Nourine | Incremental | Lexicographical Tree | Intersecting object with earlier generated concepts |
| Chein | Batch | Levelwise approach | Using earlier generated concepts |
| Lindig | Batch | RB-Tree | Extending earlier generated concept (by adding new attribute/object) |
| Bordat | Batch | Trie | Extending earlier generated concept(by finding new object, minimal by inclusion) |
| CBO | Batch | Lexical order | Extending earlier generated concept (by adding new attribute/object) |
| Grail | Batch | Set of earlier visited attributes | Extending earlier generated concept (by adding new attribute/object) |
| Titanic | Batch | Levelwise approach | Using earlier generated concepts and support heuristic |
| Godin | Incremental | Cardinality heuristic | Intersecting object with earlier generated concepts |

# Strategies for construction of line diagram

- Just in time – when concept is calculated for the first time, all his direct predecessors are calculated (Bordat, Lindig, our)
- After calculation of concept set for each element determine direct predecessors and find them between generated concepts (Nourine, Ganter, …)

*Some algorithms can be used with both strategies*

- Incremental calculation and updating of line diagram (Godin)

# Methodic of comparison of algorithms

- Software system for comparison of algorithms was developed (console java application)
- Comparison was performed on randomly generated contexts of different sizes with different percent of fill cells per row, and on contexts of kind $(G, G, \neq)$ on which worst case is achieved .
- All algorithms comparisons( as for calculation of concepts' set, as well for construction of line diagram) were performed on the same set of contexts
- For every non-square context comparison was also performed on transposed context, to which corresponds isomorphic concepts' lattice.

# Methodic of comparison of algorithms

- In order to ensure independence from garbage collector between runs of different algorithms all references to data, used and generated by previous algorithm, were freed and garbage collector was called
- Before starting comparisons one test run on small context was performed, in order to ensure presence of all used classes in memory
- No use of virtual memory was allowed
- Most efficient implementations, known to author, were used
- Comparison was performed on Intel Celeron 700 machine with 512 MB of RAM, with OS Windows NT 4.0 (Service Pack 6), otherwise idle.

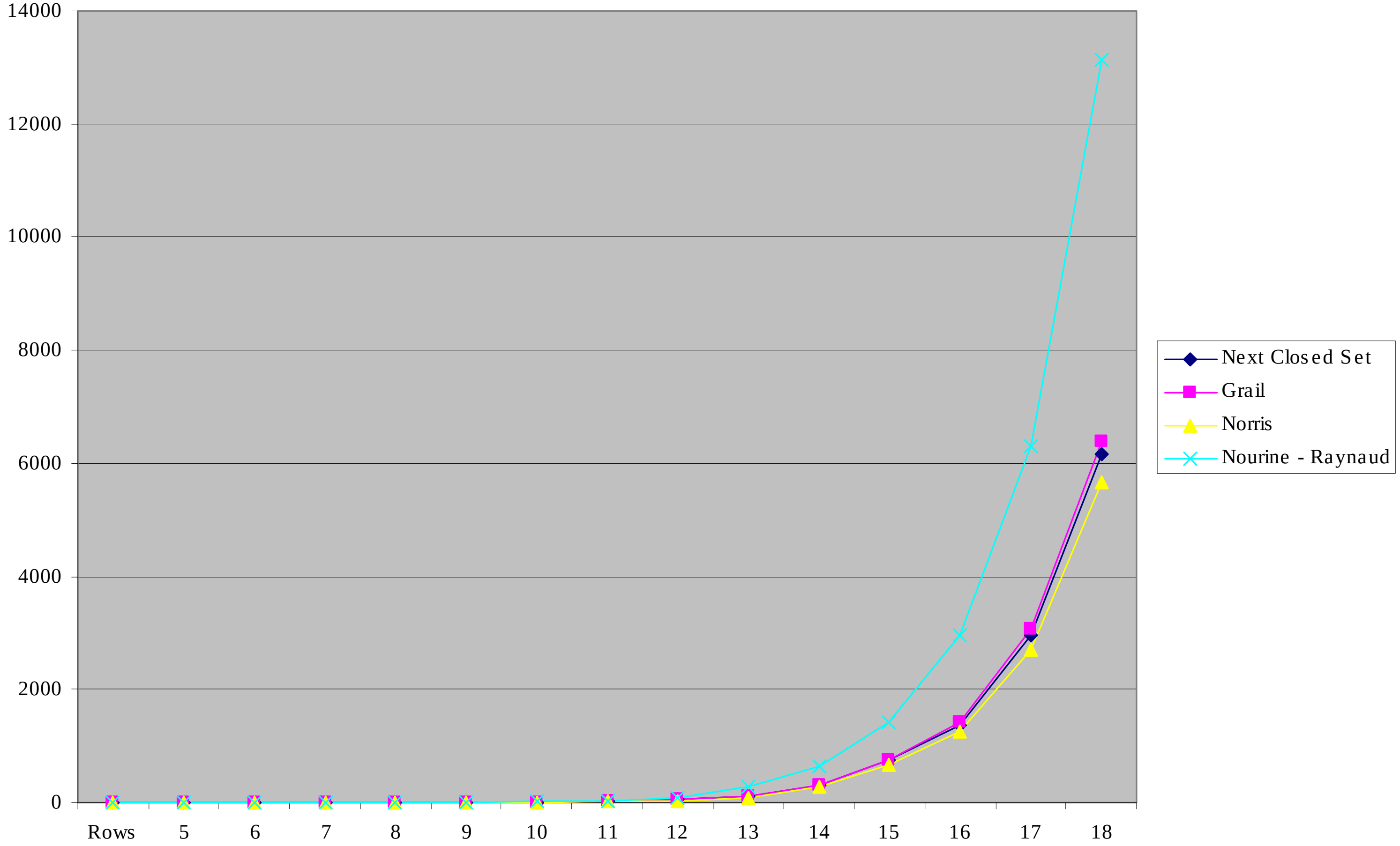
# Contexts, on which comparison was performed

- *Sparse contexts* - with number of rows from 100 to 900 (step 100) and 100 columns, with 4 filled cell in a row in randomly generated positions and transposed ones
- Contexts with number of rows from 20 to 100 and 20 columns, when were filled from 10 % to 70% of cells in a row and transposed ones
- (G, G, $\neq$) for |G| from 5 to 19

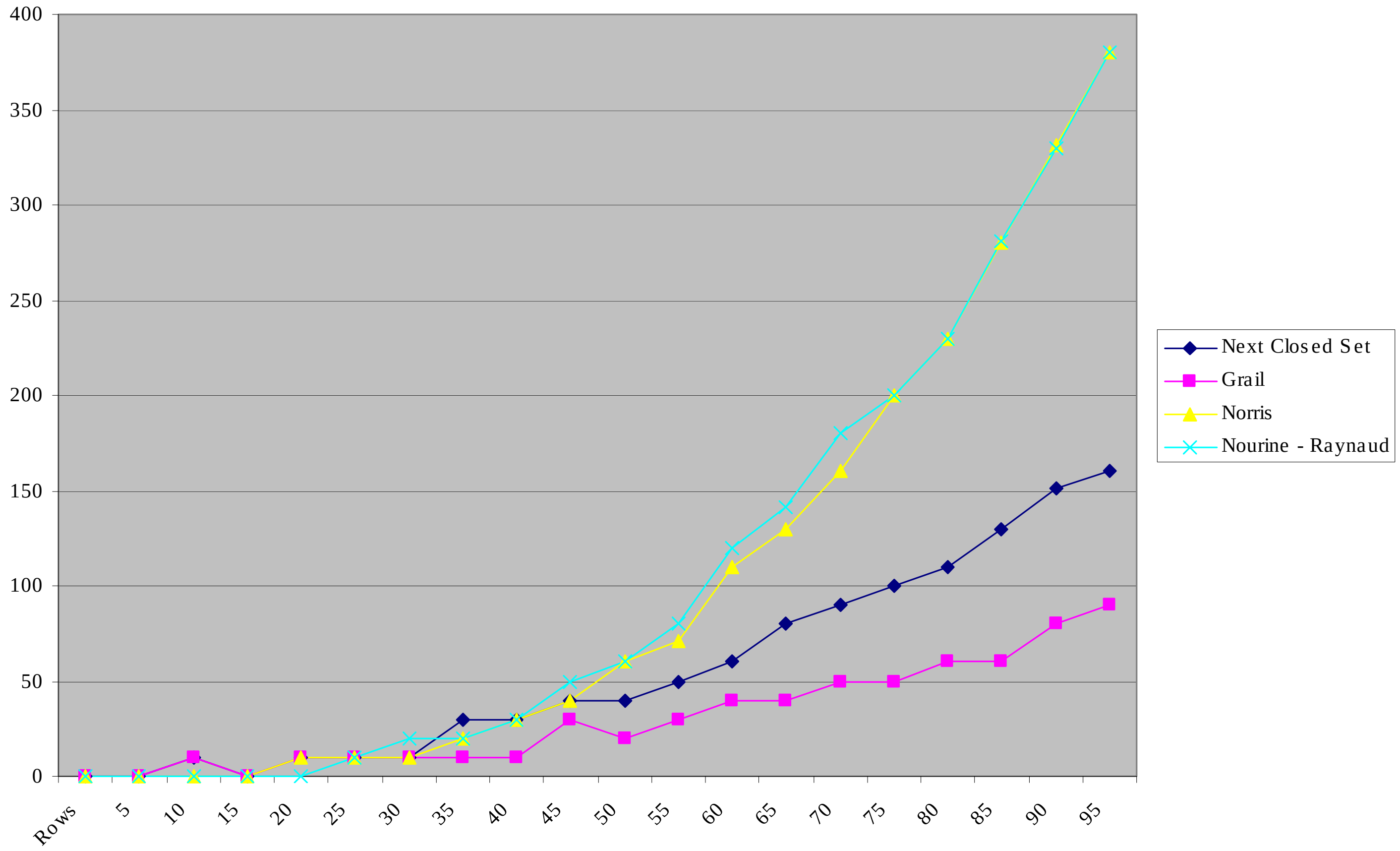
# Compared algorithms for calculation set of concepts

- *Next Closed Set* (Ganter) – version, working in top-down way (dual to original one)
- *Grail* – my algorithm
- *Norris*
- *Nourine-Raynaud*

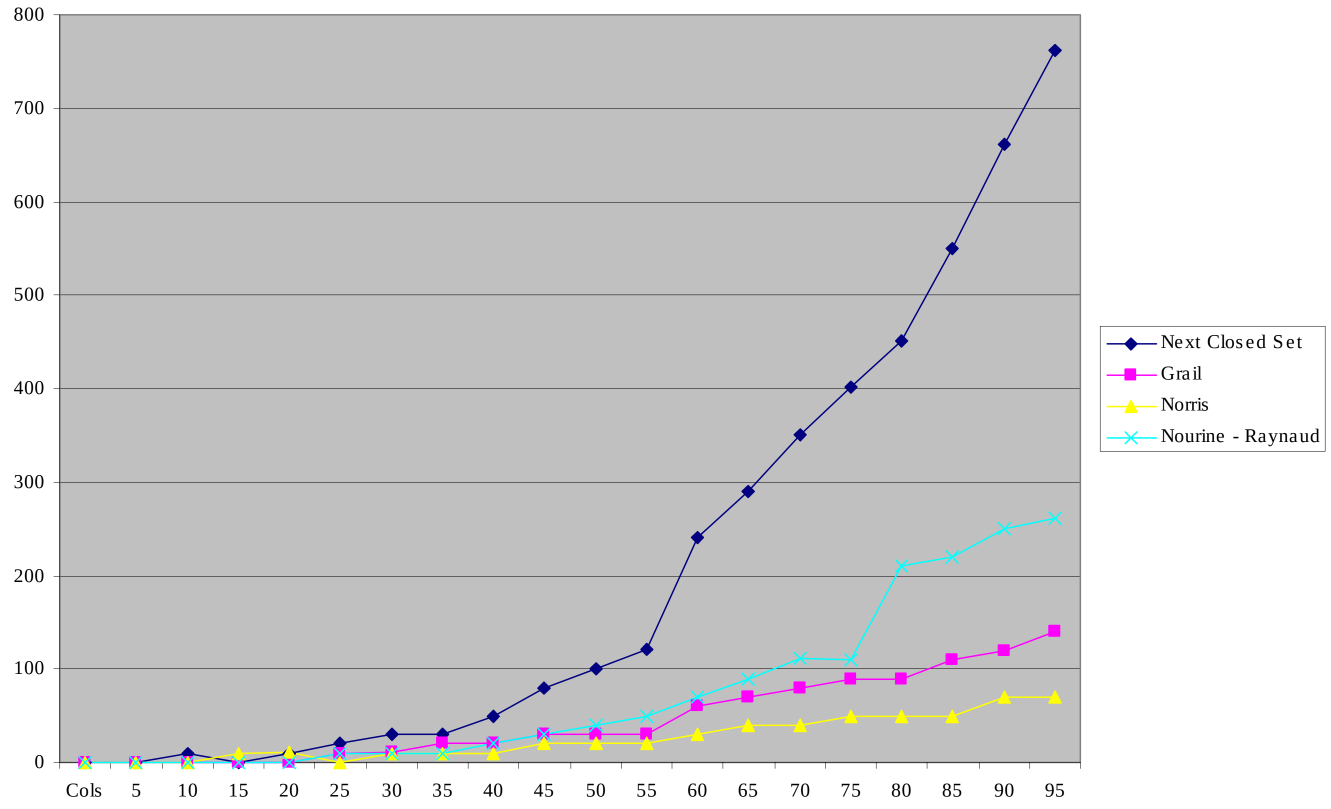
# Calculation of Concept Set (G, G, ≠)



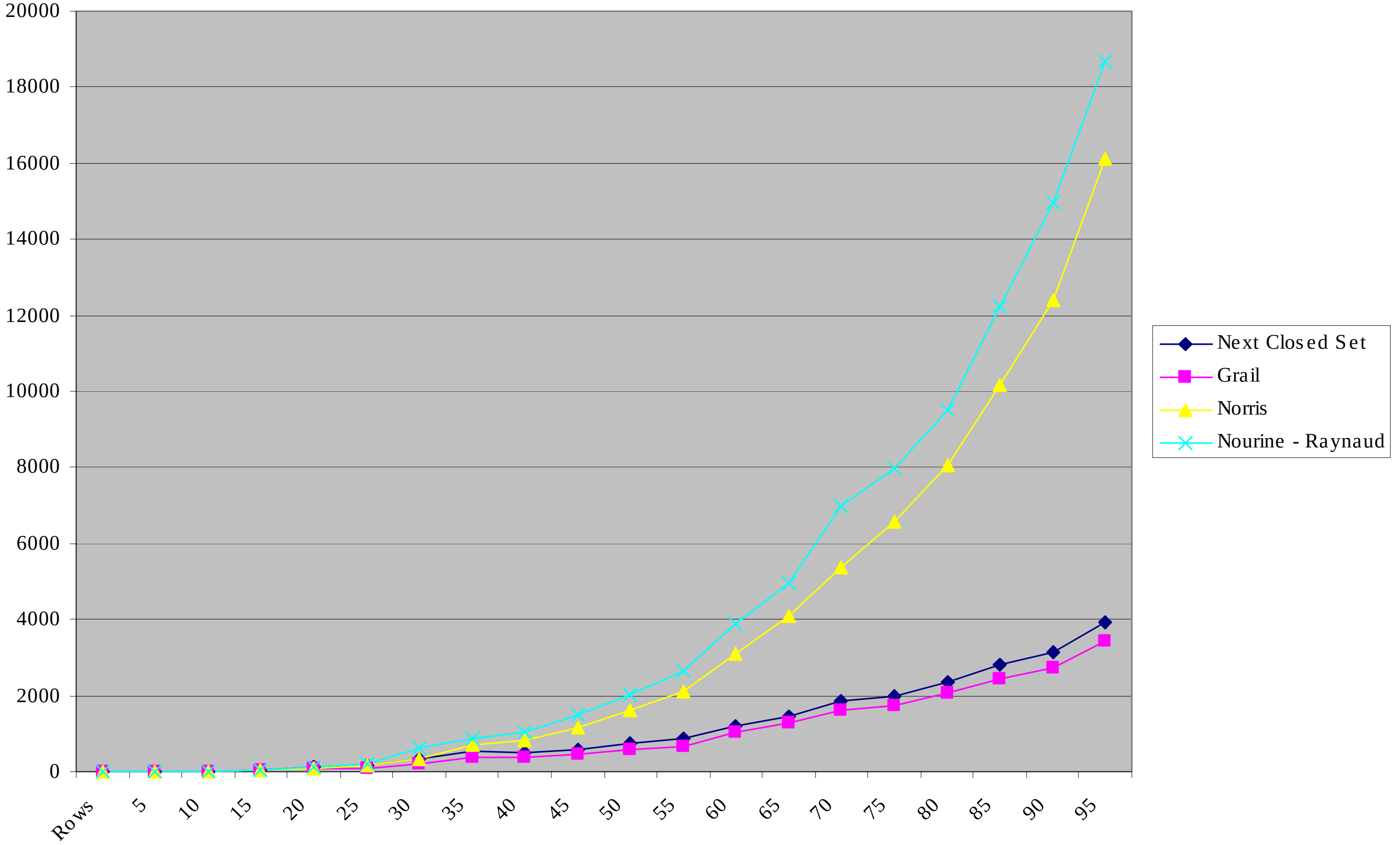
# Calculation of Concept Set |G|=20..100 |M|=20, fill factor(per row)= 0,4



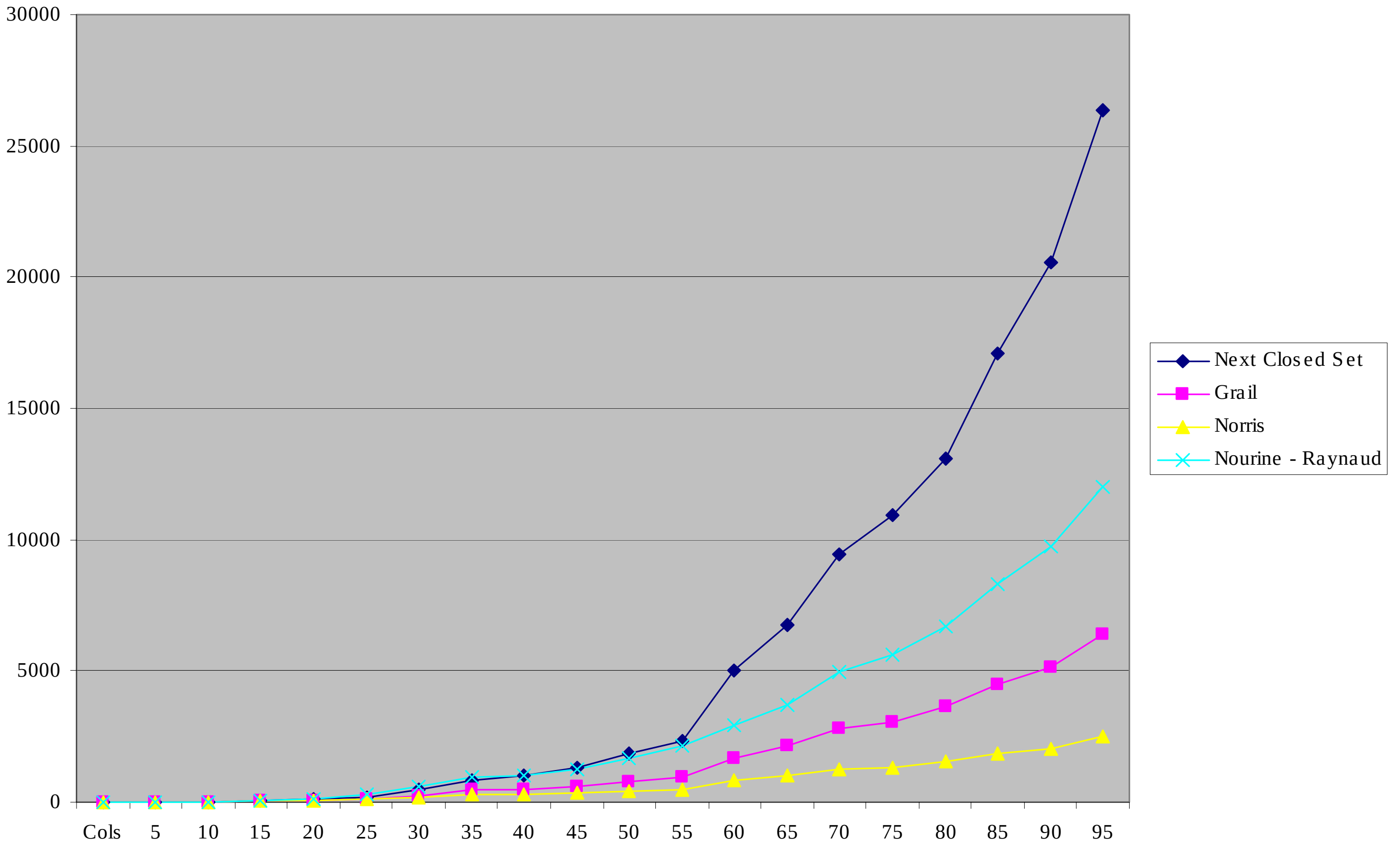
# Calculation of Concept Set |G|=20 |M|=20..100, fill factor(per column)= 0,4



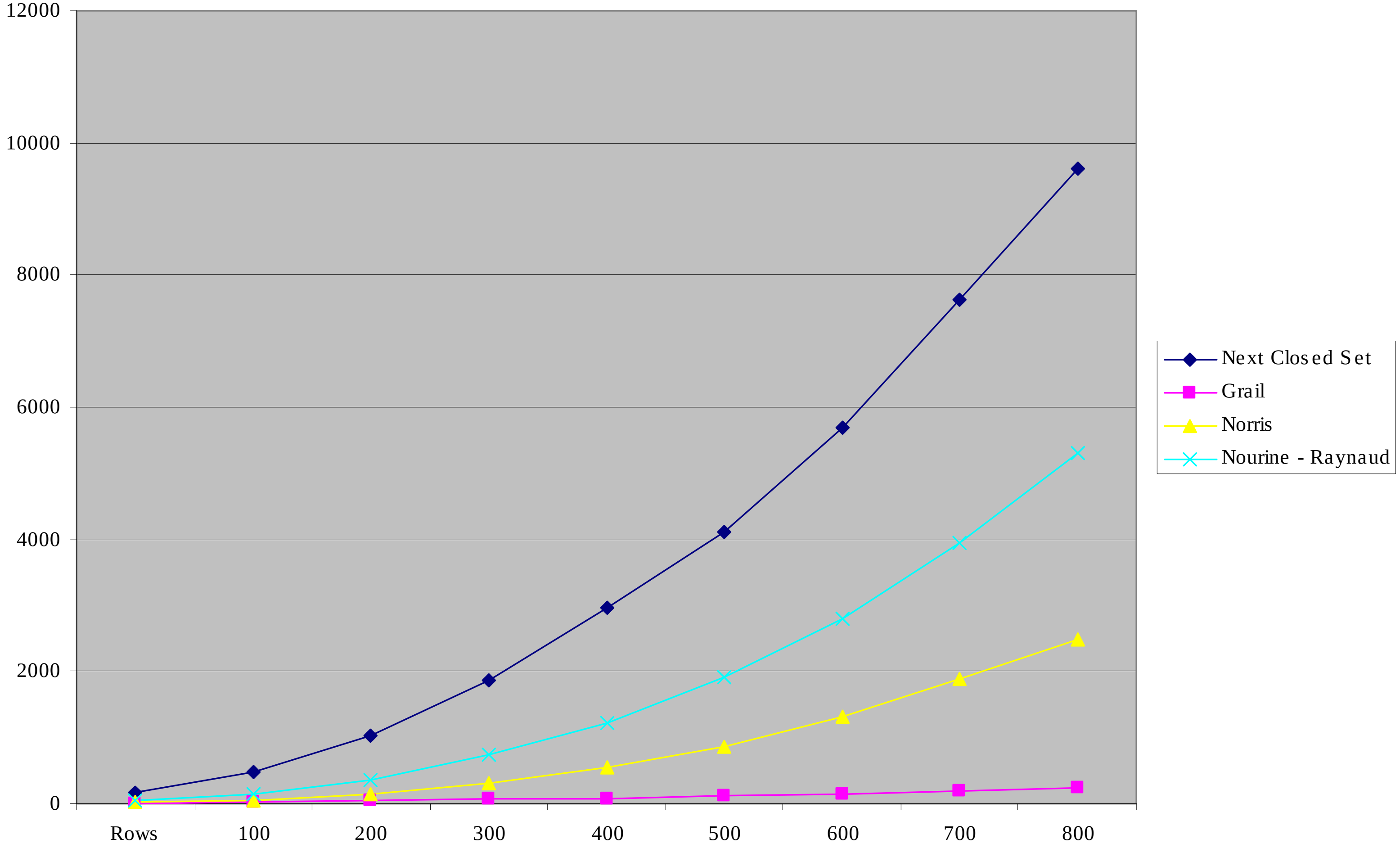
# Calculation of Concept Set |G|=20..100 |M|=20, fill factor(per row)= 0,7



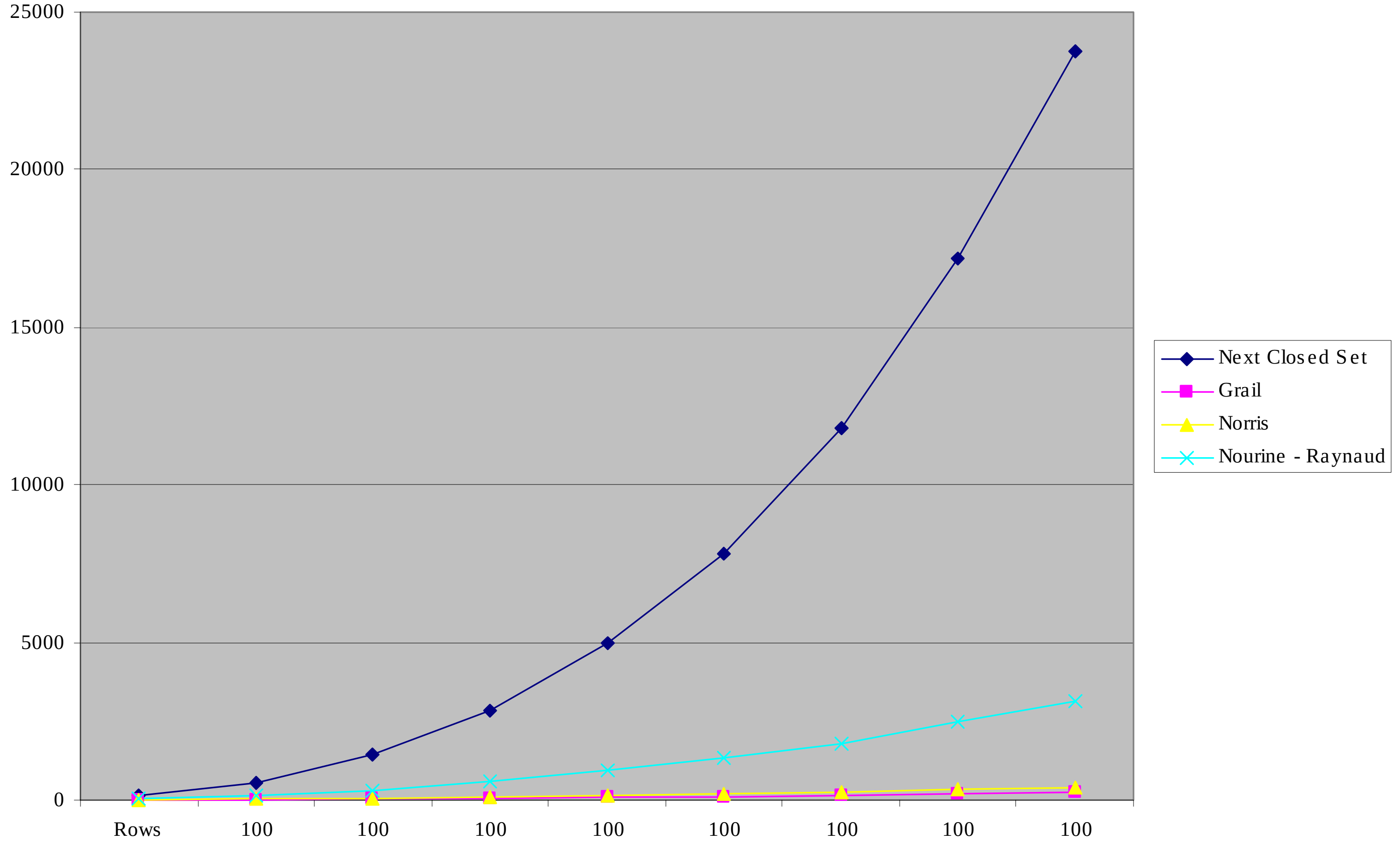
# Calculation of Concept Set |G|=20 |M|=20..100, fill factor(per column)= 0,7



# Calculation of Concept Set |G|=100..900 |M|=100, |g'|=4



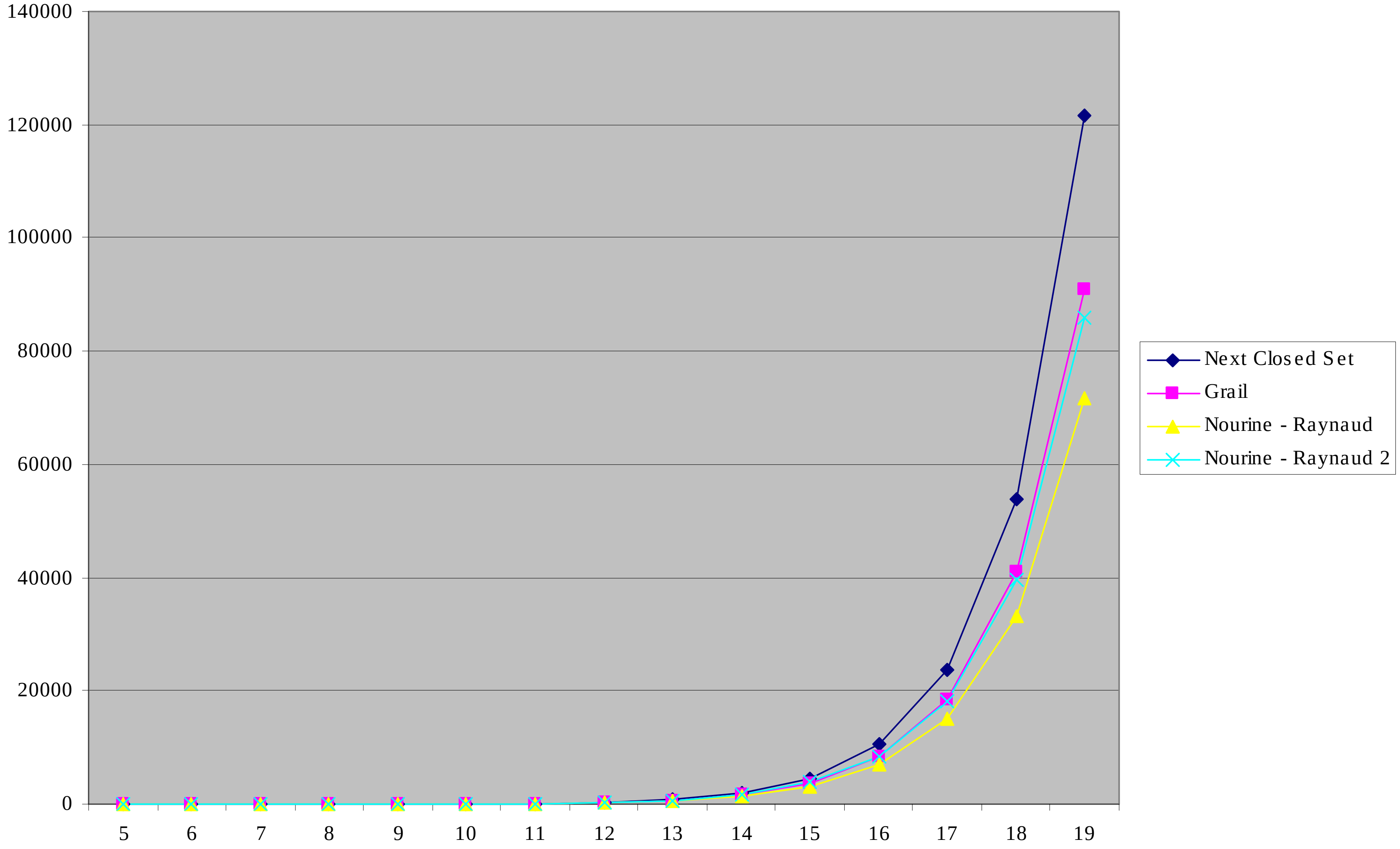
# Calculation of Concept Set |G|=100 |M|=100..900, |m'|=4



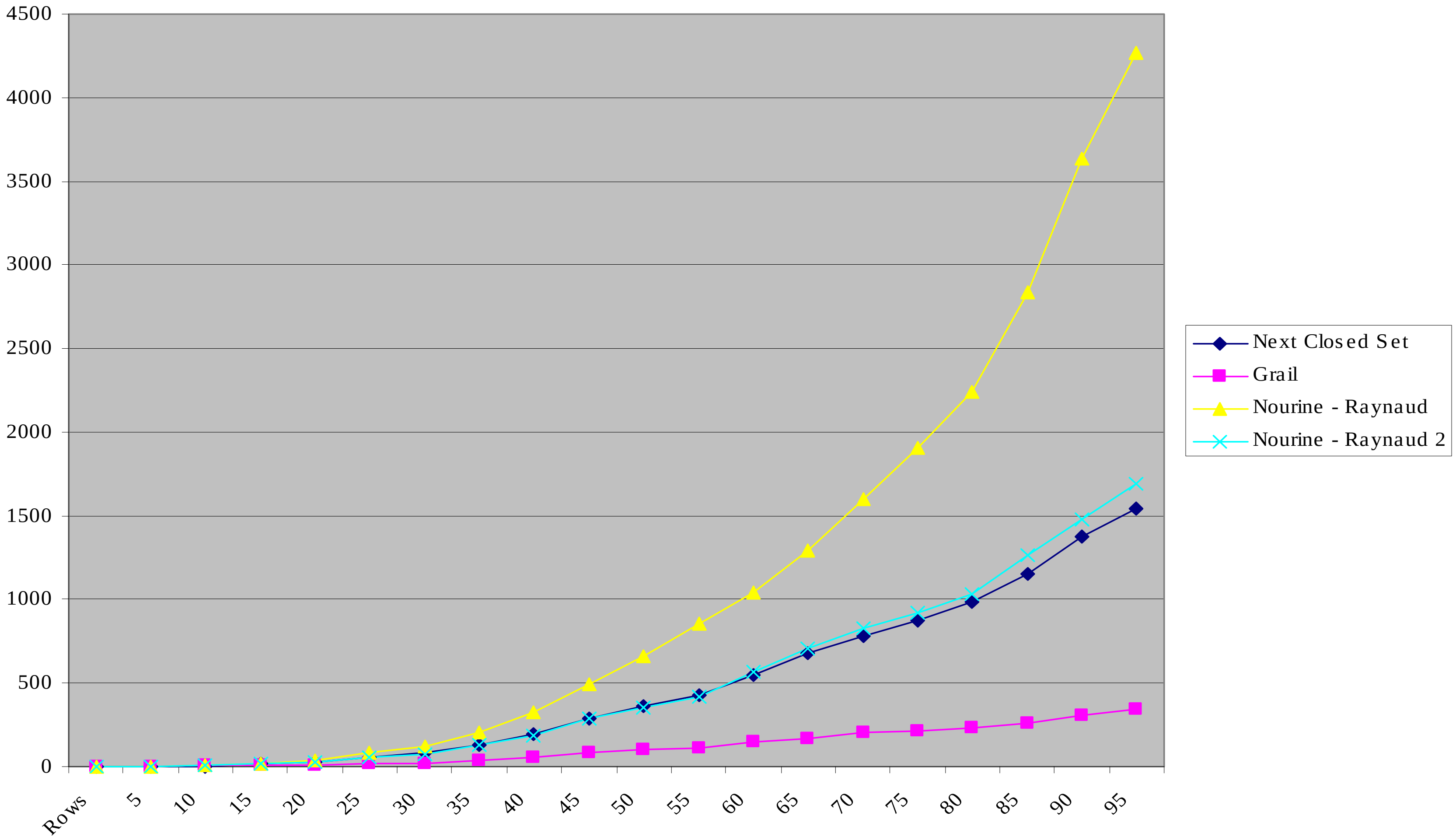
# Compared algorithms for construction of line diagram

- *Next Closed Set* (*Ganter)* - with efficient procedure for constructing line diagram, exploiting binary search, proposed by Sergey Objedkov.
- *Grail* (my)
- *Nourine-Raynaud* with procedure for construction line diagram, proposed by creators
- *Nourine-Raynaud* with procedure, based on calculation of successors intents and search of corresponding concepts in lexicografical tree

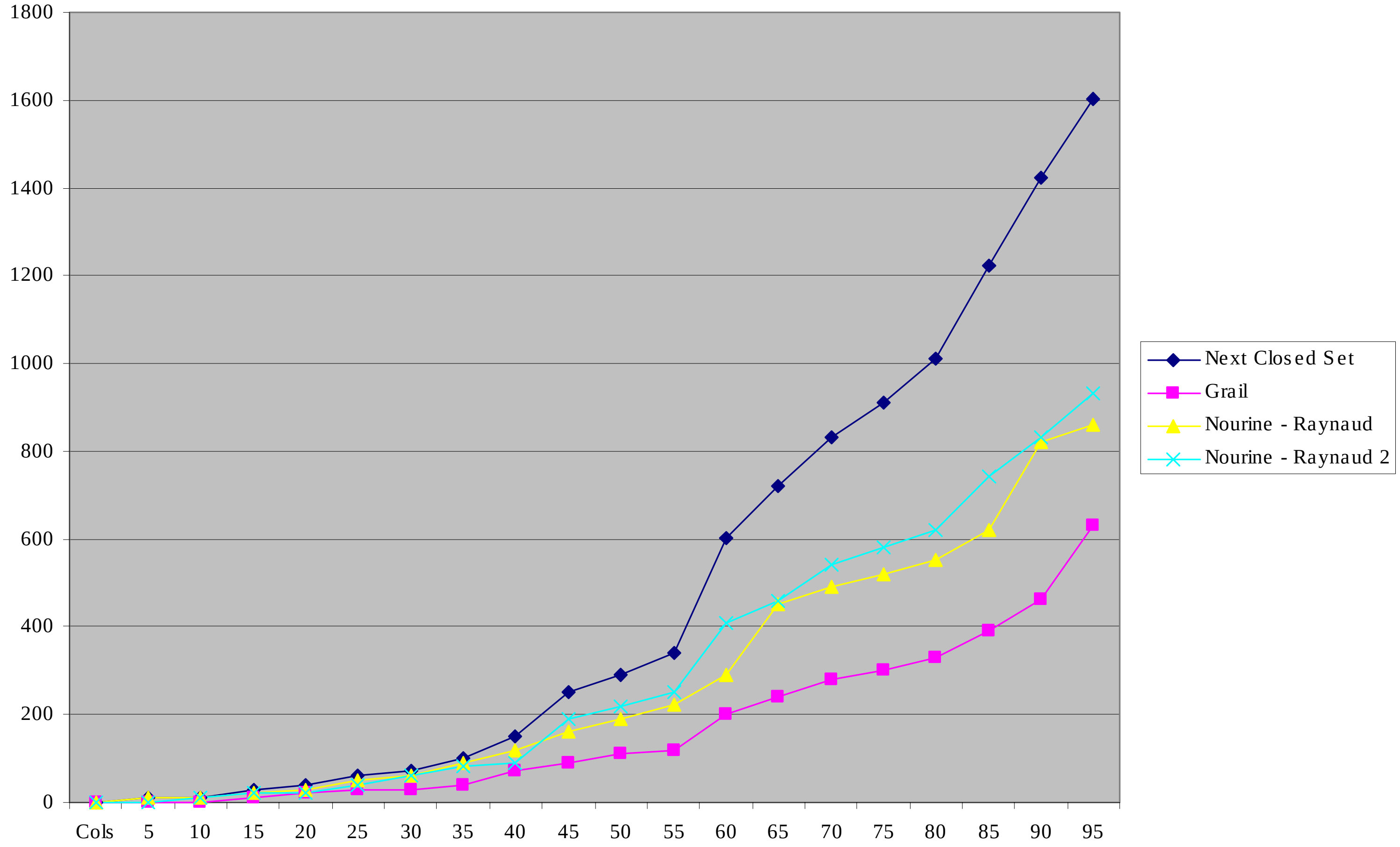
# Construction of Line Diagram (G, G, ≠)



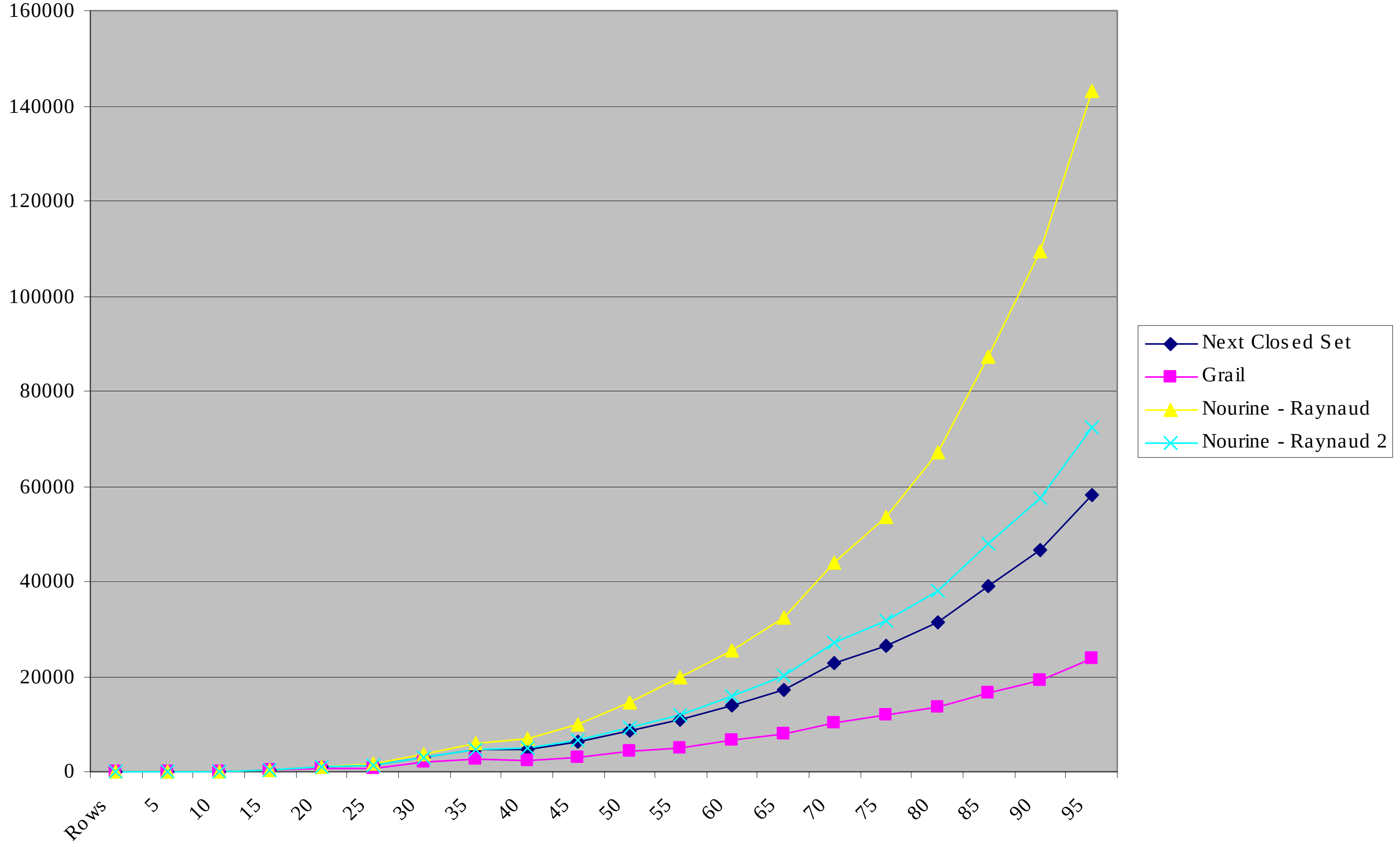
# Construction of Line Diagram |G|=20..100 |M|=20, fill factor(per row)= 0,4



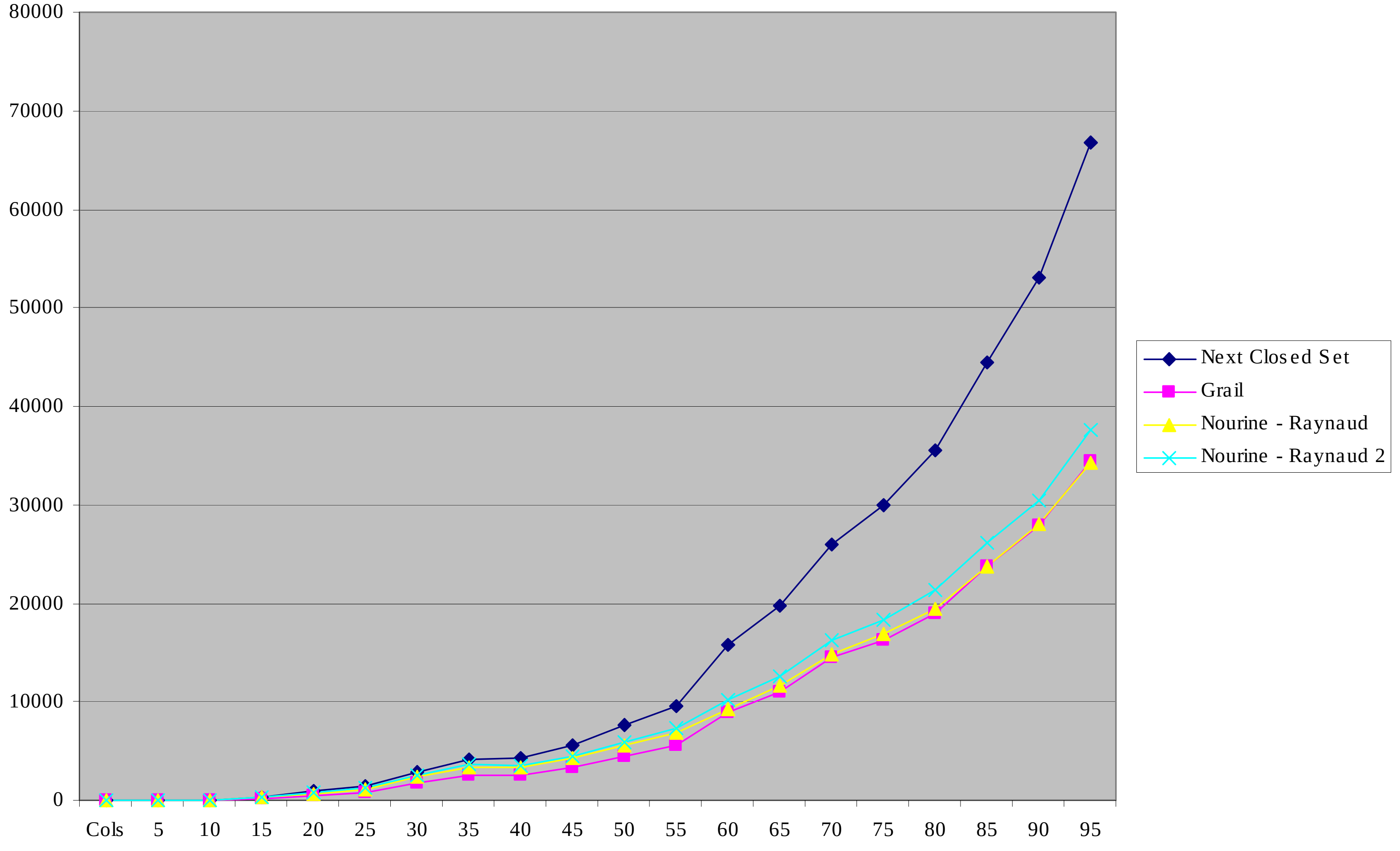
# Construction of Line Diagram |G|=20 |M|=20..100, fill factor(per column)= 0,4



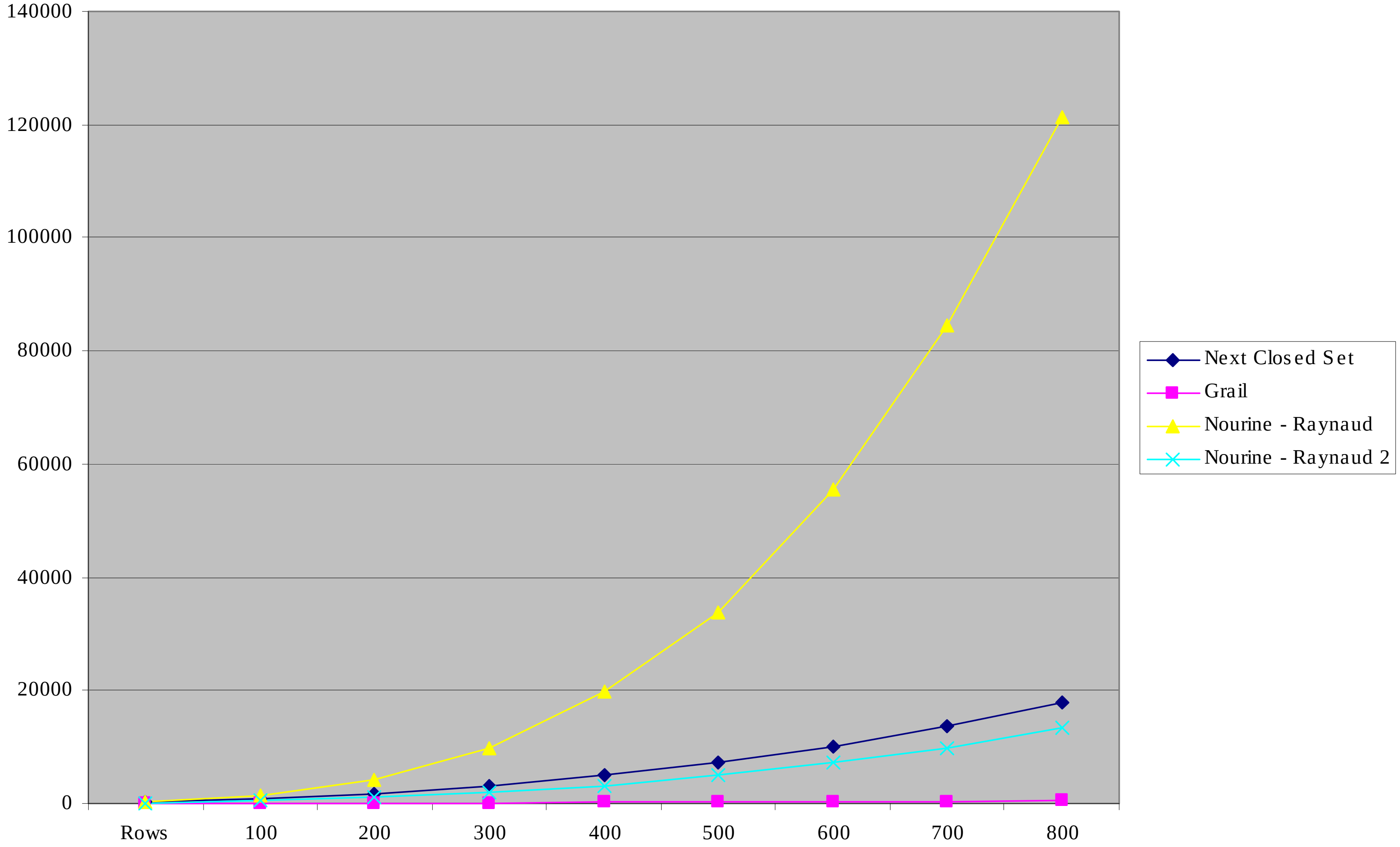
# Construction of Line Diagram |G|=20..100 |M|=20, fill factor(per row)= 0,7



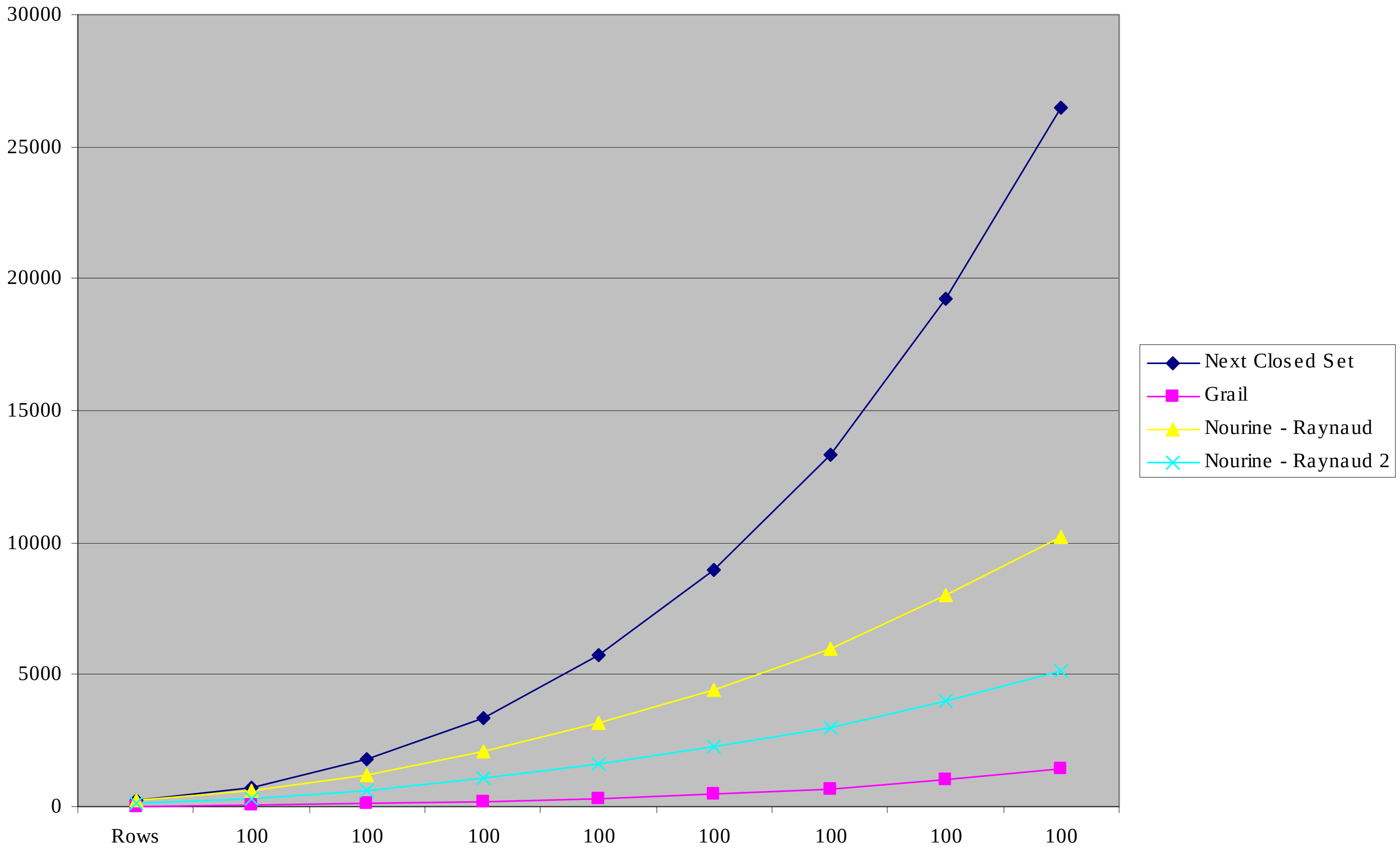
# Construction of Line Diagram |G|=20 |M|=20..100, fill factor(per column)= 0,7



# Construction of Line Diagram |G|=100..900 |M|=100, |g'|=4



# Calculation of Concept Set |G|=100 |M|=100..900, |m'|=4



# Algorithm for finding base of implications, which holds in context

- Also was developed algorithm for finding bases of implications, which holds in context
- It is based on classical in FCA notion of pseudointents (Duquenne-Guiguies) and based on top – down approach, like algorithms for calculating set of concept/building line diagram
- Drawback – order, which is good for building concept lattice, isn’t so good for finding base of implications – algorithms  has a poor performance compared with a NextClosedSet.

# Concept Explorer

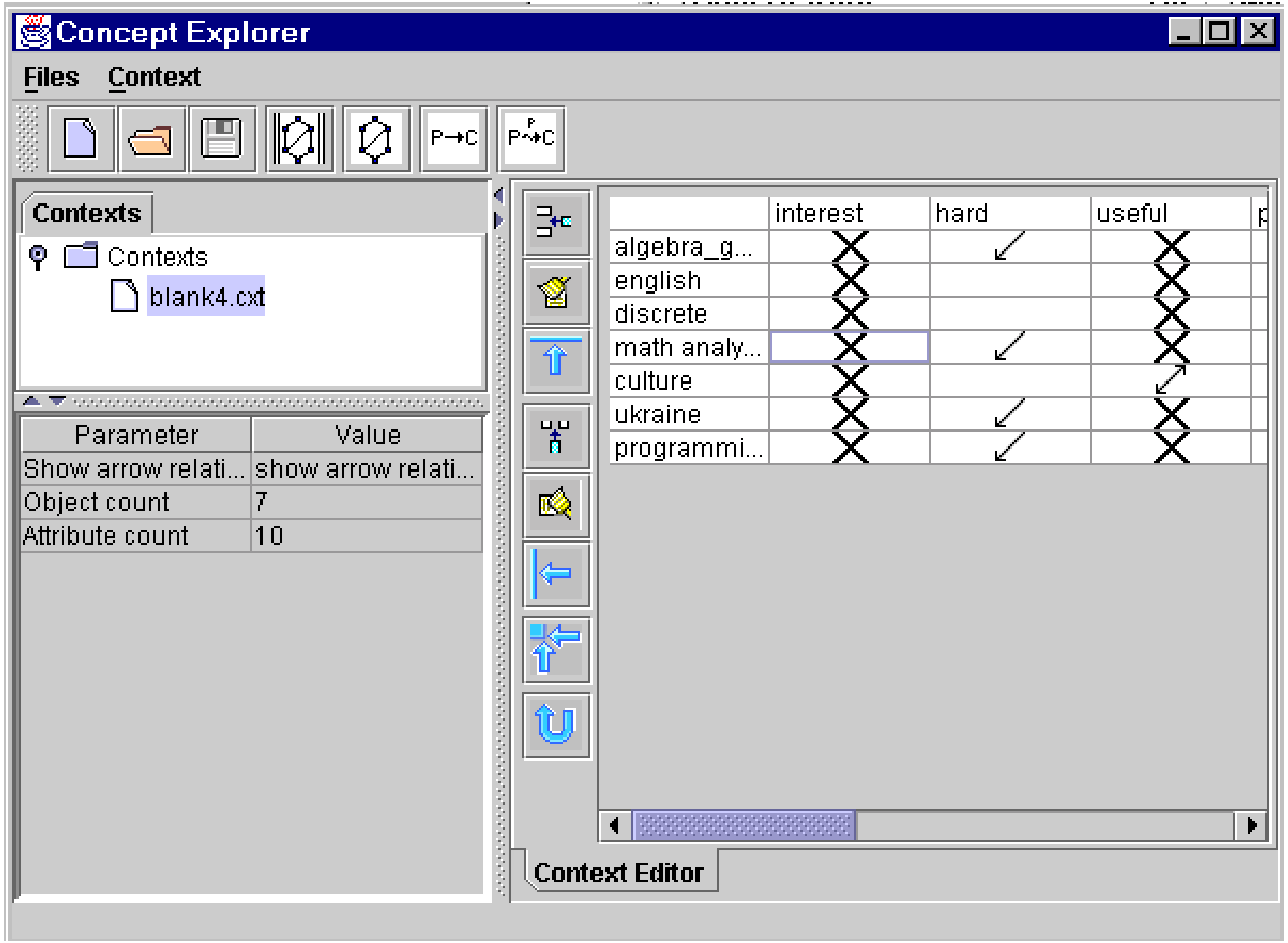
During work on project I developed system "Concept Explorer".

It is written on Java™ language.

Now it consists from two parts:

- GUI front end
- Library for performing experiments with algorithms

# Concept Explorer



# Concept Explorer

It supports following functionality:

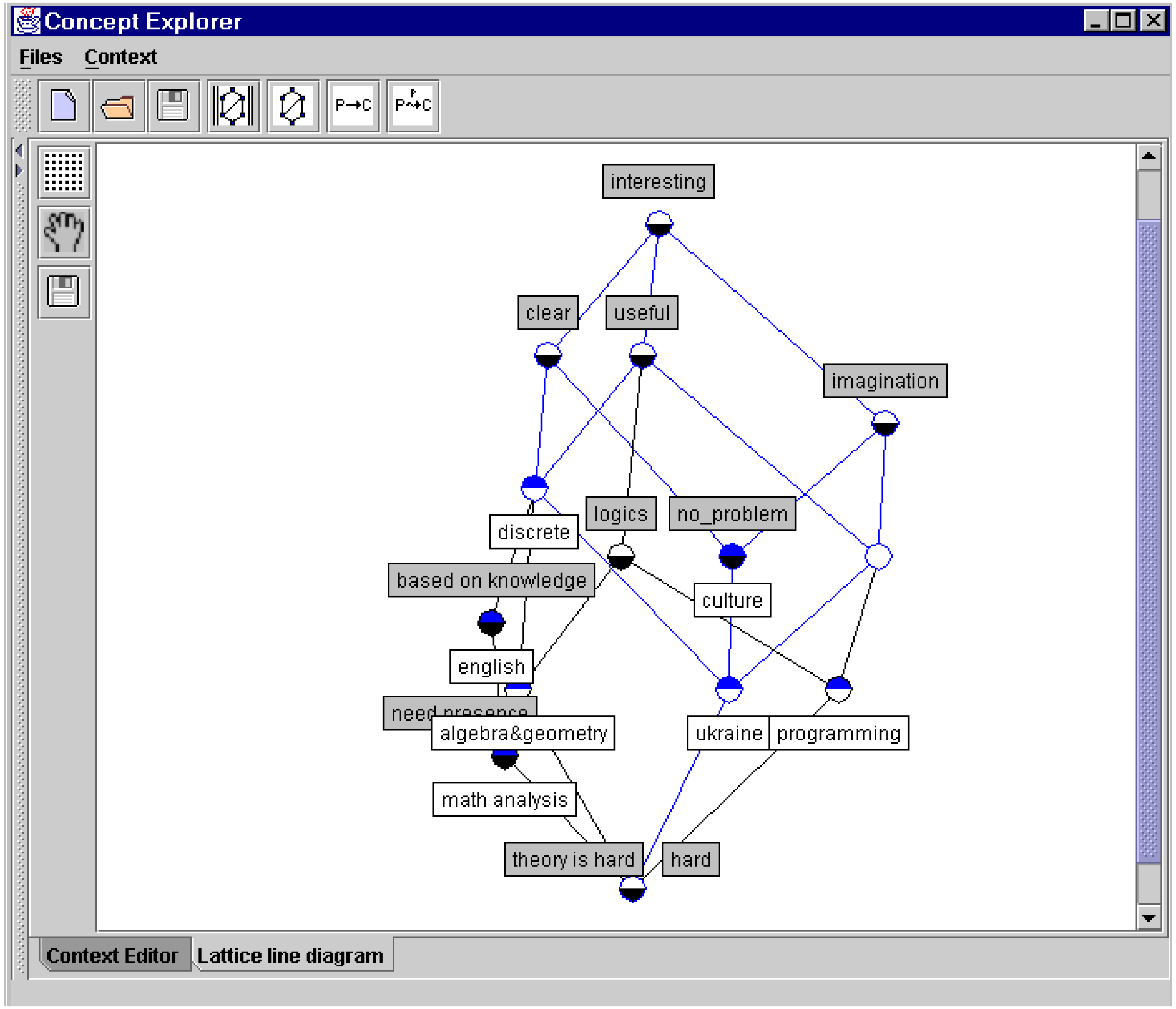
Context processing:

- Context editing
- Calculation of arrow relations
- Reduction and purifying of context

FCA operations

- Defining concepts count
- Calculating set of all concepts
- Construction of line diagrams
- Finding base of implications, which holds in context

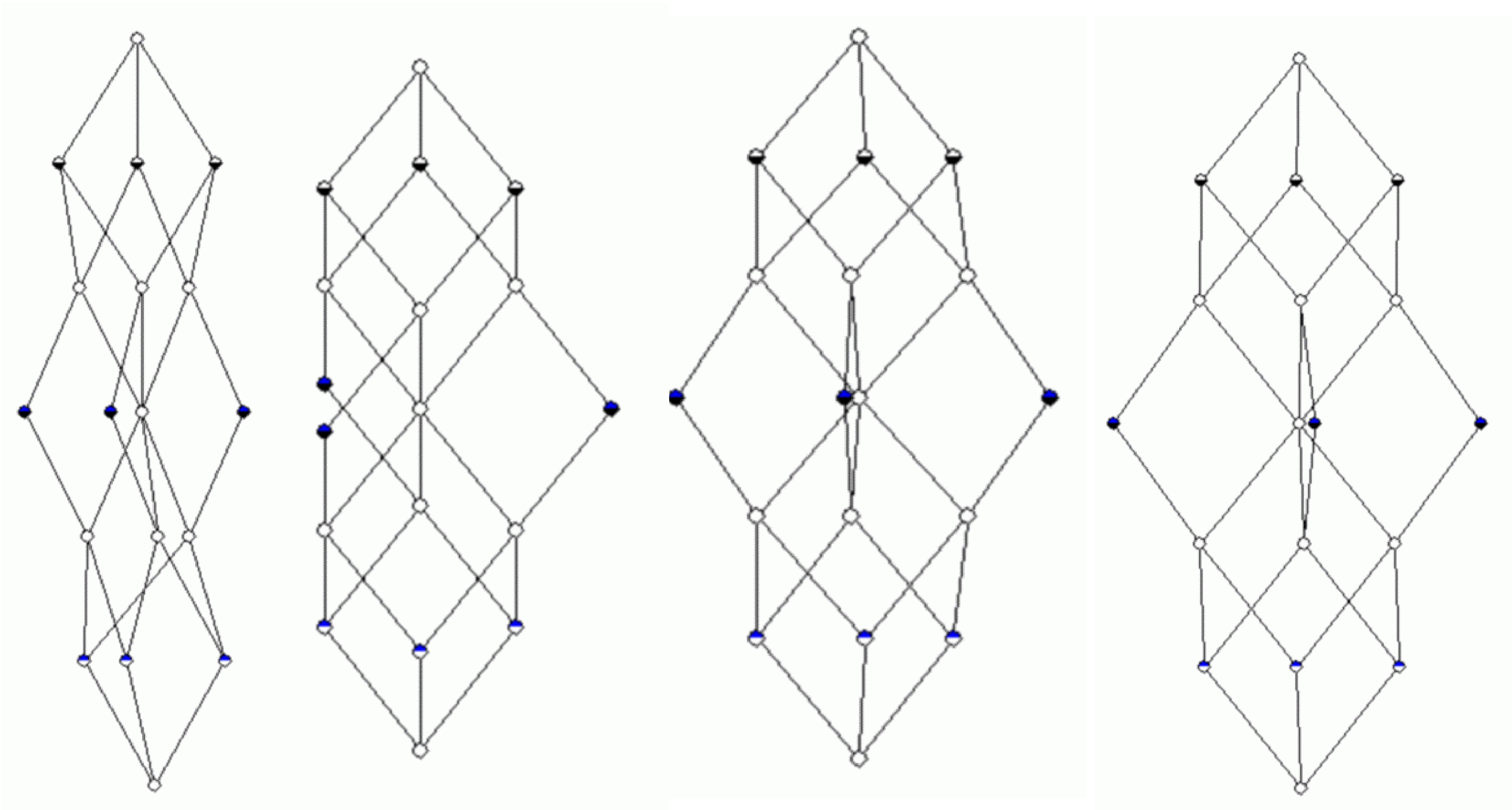
# Concept Explorer



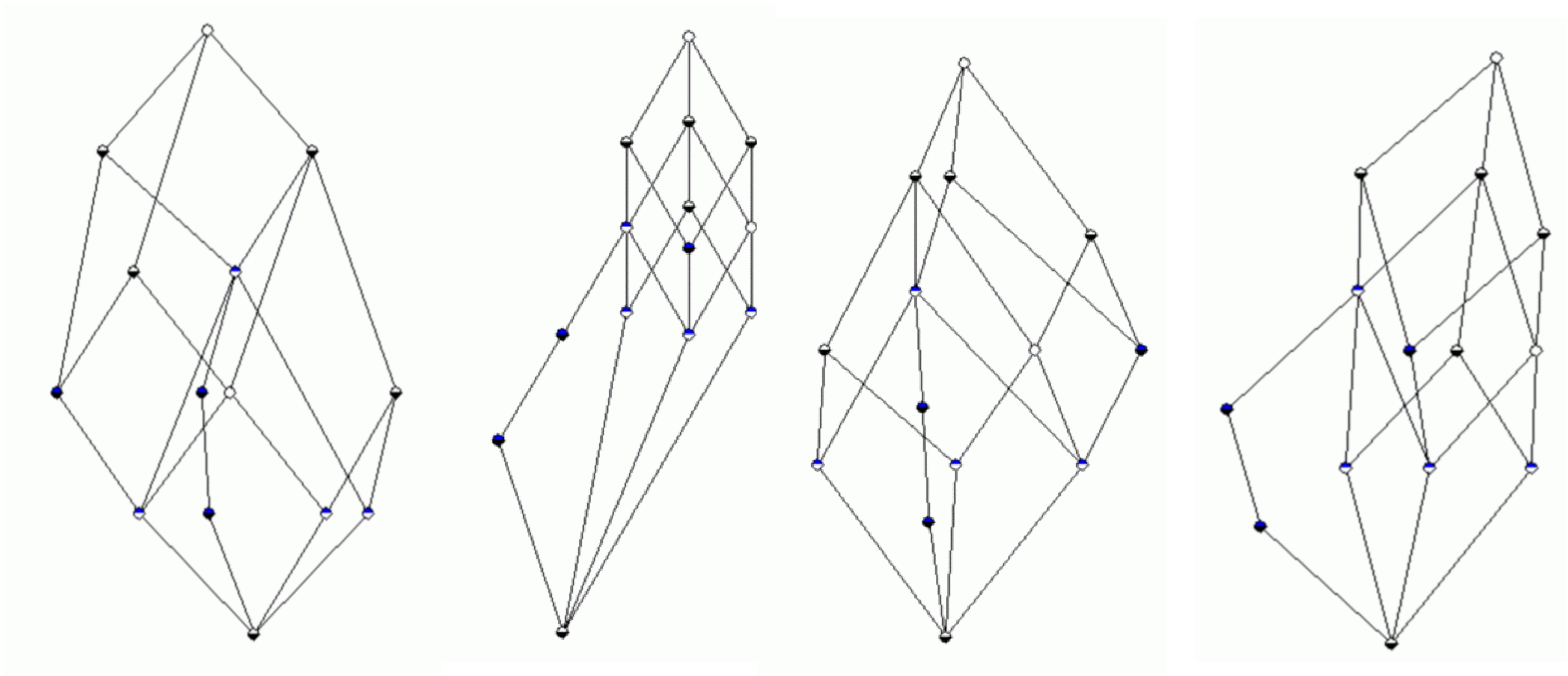
# Concept Explorer

- Visualization of line diagrams(several layout methods and modes of visualization)
  - Layout, minimizing number of edge intersection
  - Chain decomposition layout
  - Two different schemes of force-directed layout
- Mining bases of association rules

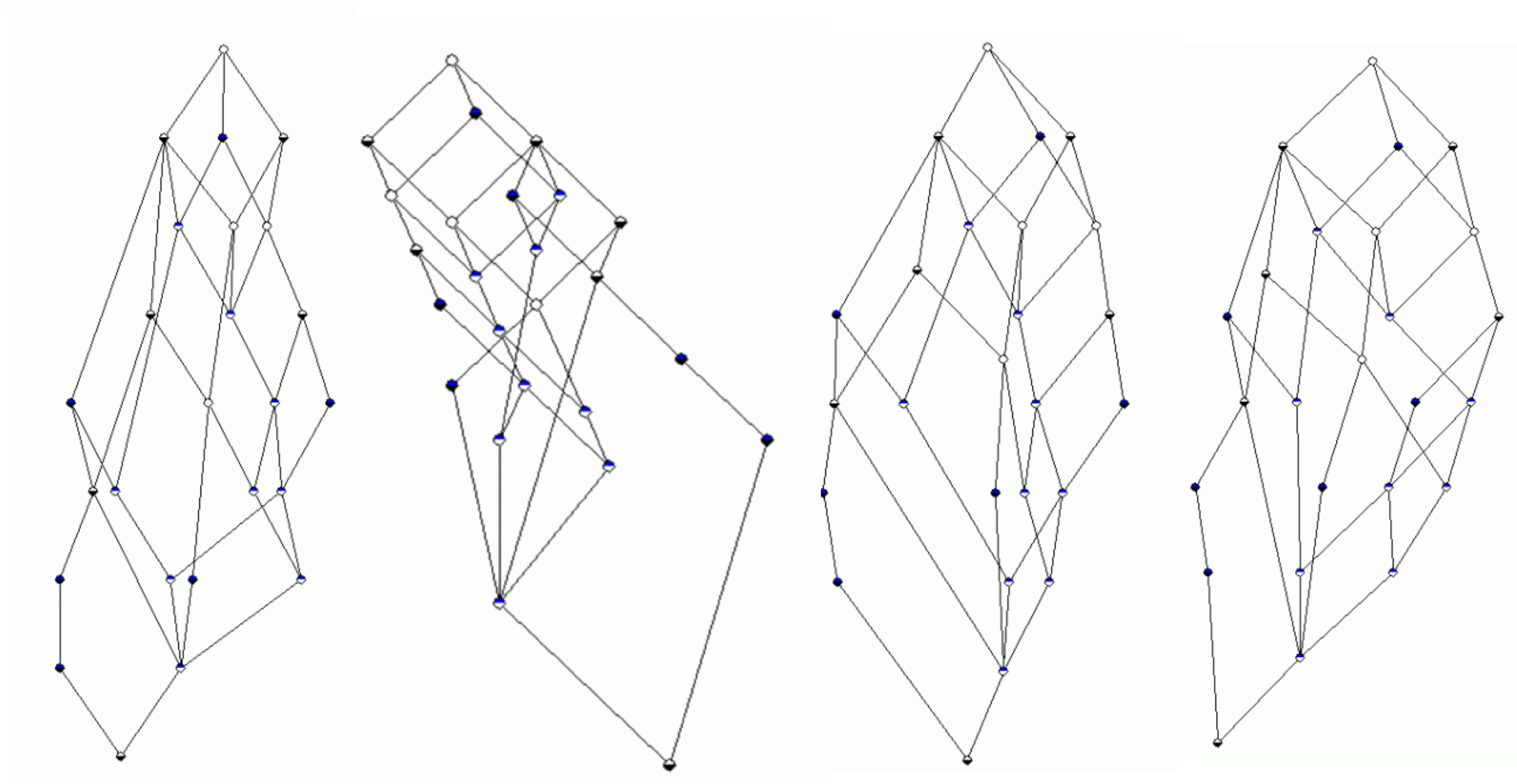
# Supported methods of layout of concept lattices



# Supported methods of layout of concept lattices (2)



# Supported methods of layout of concept lattices (3)



# Concept Explorer: future development

- Support for multi-valued context
- Integration of tools for data preprocessing
- Support for nested line diagrams
- Integration of tools of other logic – algebraic methods of data analysis (JSM-method, Rough set theory)

# Areas of current interest

- Development of algorithms for FCA using BDD – based presentation of concept lattice
- Performing analysis of data, gathered in Ukrainian Cancer Register

# Algorithm(for calculating Hasse diagram)

```
C = (G, G′)
if (G, G′) ≠(M′, M)
       C = C∪ (M′, M)
     FindPredecessors((G, G′), ∅, ∅)

FindPredecessors((A′, A″), Prohibited)
   Desc = (∪{gI | g ∈A′})\A″
    if Desc = ∅
       Connect((A′, A″), (M′, M))
    else
       WorkSet = Desc
     For each m ∈ WorkSet
               Intent = M ∩ (∩{gI |g ∈ A′ & m ∈ gI })
                   Extent={g | g ∈ A′ & m ∈ gI}
                   Outer = ∪{gI | g ∈A′ & m ∉ gI }
                        if (Intent ∩ Outer )\A″ = ∅
                            WorkSet = (WorkSet \ Intent) ∩ Outer
                            if (Intent ∩ Prohibited)\A″ = ∅
                                     if intent=M
                                         Connect((A′, A″), (M′, M))
                                     else
                                         C = C∪ (extent, intent)
                                         FindPredecessors((extent,            intent),
              Prohibited)
                   Prohibited=Prohibited ∪ {m}
```