THE 24th ANNUAL CONFERENCE

ON STATISTICS, COMPUTER SCIENCE

AND OPERATIONS RESEARCH

AN ALGORITHM TO COUNT RIME N-FREE POSETS

BY

Bayoumi I. Bayoumi , Mohamed H. El-Zahar

and Soheir M. Khamis

ORGANIZED BY

Institute of Statistical studies and Research

Cairo University

EGYPT

23 - 25 December 1989

An Algorithm To Count Prime N-Free Posets

BY

Bayoumi I. Bayoumi*, Mohamed H. El-Zahar and*

Soheir M. Khamis*

Abstract.

In this paper we present five algorithms to count the elements of the class of. So-called, prime N-free posets. Our method is based on the correspondence between super diagonal matrices and N-free posets. According to Pσlya's enumeration theorem [6] and Stanley's results [11], we obtain an efficient method to compute the number of N-free posets by using the results of our program. As a result of our algorithm, the previously known number of N-free posets with 10 elements, given in [8] 1985, Proved to be incorrect.

Keywords:

Posets, N-free posets, prime N-free posets, enumeration of N-free posets.

AMS Mathematics subject classification (1980) 05A15 and 06A10.

Department of Mathematics, faculty of science, Ain Shams University. Cairo. Egypt.

§1. Introduction.

In [5] M. Habib and R. H Mohring investigated the complexity of N-free posets with respect to some well-known combinatorial optimization problems which are polynomials solvable on series-parallel posets. Among them are the jump number problem, the isomorphism problem and the scheduling problem (minimizing the sum of weighted completion times on one machine).

This paper is devoted to the description of an algorithmic method to count the elements of the class of N-free posets. N-free posets have recently gained much attention in connection with the jump number [10, 12] since it can computed efficiently for these posets, i.e., polynomially solvable on this class. This has led to some structural insights into this class of posets. These properties, which were studied in [5] and summarized here in §2, have led to the belief that also other, generally NP-hard problems on posets should be efficiently solvable on this class. M. Habib and R. Mohring [5] have shown that both the isomorphism problem and the 1-machine scheduling problem are hard on this class although they can be solved in polynomial time by means of the decomposition tree for series-parallel posets. The main reason to this behavior of N-free posets is the fact that any given poset can be embedded into an N-free poset, and thus the complexity of arbitrary posets can be modeled within N-free posets.

We then introduce in §3 the two representations of N-free poset which are the block and the matrix representations. Apart from a possible permutation applied simultaneously to the rows and the columns, there exists a 1-1 correspondence between the super diagonal matrices of 0 and 1 entries and unlabeled prime N-free posets. This correspondence leads to construct an efficient program. This program which is described in §4 consists of five algorithms, that are used for counting the number of non-isomorphic unlabeled prime N-free posets (prime N-free posets for short) of n elements. By using Pσlya's enumeration theorem [6] and Stanley's results [11], one can easily compute the numbers of N-free posets. The appendix contains these numbers for n≤12, and the cycle index polynomials of automorphism groups of prime N-free posets.

§2. Fundamental Definitions and Basic Properties.

If a *partially ordered set (poset)* is denoted by P, then its underlying set will usually also be denoted by P, and its order relation by ≤ or $\leq_p$. By $<\cdot$ we denote *the associated* **covering** *relation*, i.e., a $<\cdot$ b if a < b and a ≤ c ≤ b implies that a = c or c = b. If a and b are *incomparable* (i.e., neither a ≤ b nor b≤a), we write $a \| b$.

Let Q be a poset. let $a_1, a_2, \ldots, a_h \in Q$ and let $P_1, P_2, \ldots, P_h$ be posets such that Q, $P_1, P_2, \ldots, P_h$ are mutually disjoint. Then by

$$P = Q^{P_1, P_2, \ldots, P_h}_{a_1, a_2, \ldots, a_h}$$

we denote the poset resulting from substituting the elements a_i of Q by the associated poset P_i (1≤i≤h). More formally:

$$a \leq_P b \leftrightarrow \exists\ i \text{ with } a, b \in P_i \text{ and } a \leq_P b,$$
$$\text{or } \exists\ i \neq j \text{ with } a \in P_i,\ b \in P_j \text{ and } a_i \leq_Q a_j.$$

The substitution is *proper* if $1 < |P_i| < |P|$ for some i. A poset is *decomposable* if it can be obtained by proper substitution. Other-Wise it is said to be *indecomposable* or *prime.* This substitution operation and the associated decomposition are well investigated in [4,9]. We recall the following facts and theorems, which will be used throughout the paper.

Each decomposable poset P is obtained by a sequence:

$$P_1 = Q_1,\ P_2 = P_1{}^{Q_2}_{a_2},\ \dots,\ P = P_{m-1}{}^{Q_m}_{a_m}$$

of elementary substitutions in which each Q_j is prime. The Q_j is unique (up to isomorphism and rearrangement), and are called the *factors* of P.

A subset B of P is called *autonomous* if $a < b_o$ ($a > b_o$) for some $b_o \in B$ and $a \in P \backslash B$ implies that $a < b$ ($a > b$) for all $b \in B$. A poset is a decomposable iff it has a non-trivial autonomous set B ($1 < |B| < |P|$). Then $P = Q_a^{P/B}$ where P/B denotes the subposet of P induced by B, and where Q is obtained by replacing B by just one vertex a. If $\Pi = \{B_1, B_2, \dots, B_m\}$ is a partition of P into autonomous sets. Then we denote by P/Π the poset obtained from P by replacing the set B_i by just one representative vertex $a_i \in B_i$ P/Π is called the quotient of P modulo Π.

Decomposition Theorem. For each decomposable poset P, one of the following three cases applies.

(1) $P = Q_{a_1,a_2,\dots,a_h}^{P_1,P_2,\dots,P_h}$, where Q is an antichain. Then P is said to be obtained by parallel composition of $P_1, P_2, \dots, P_h$.

(2) $P = Q_{a_1,a_2,\dots,a_h}^{P_1,P_2,\dots,P_h}$, where Q is a chain , (i.e., linear order).

(3) $P = Q_{a_1,a_2,\dots,a_h}^{P_1,P_2,\dots,P_h}$ where Q is a prime poset. Then P is said to be of the prime type (the irreducible poset w.r.t. parallel and series compositions) and Q is called the associated prime quotient poset.

The class of N-free posets was introduced by Grillet in [3]. He defined them as *the* class of posets that satisfy *CAC property* (Chain-Antichain-Complete, i.e., each maximal chain meets each maximal antichain). Grillet also shewed that a poset has CAC property if and only if it does not contain a subposet on four elements a, b, c, d with $a <_p b$, $c <\cdot b$. $c <_p d$ and $a \| c$, $a \| d$, $b \| d$.

Leclerc and Monjardet improved in [7] this characterization by proving that a poset P has the CAC property iff it does not contain as a subposet, the poset on four elements a, b, c, d, with $a <\cdot b$ and $c <\cdot b$, $c <\cdot d$, and $a \| c$, $a \| d$, $b \| d$, (Fig.1). This poset looks like the letter 'N'. Rival in [11] called them *N-free* posets (A poset P has the CAC property iff P has no 'N' in its diagram as an induced subgraph)

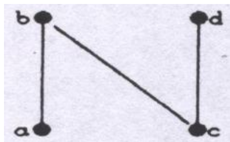


Fig.1

Unfortunately, being an N-free poset is not a hereditary property of posets (To obtain such an example just add a vertex on the covering edge cb in the poset of Fig.1). Even worse (and awkward with respect to CAC property), there are N-free posets such that the deletion of any maximal chain violates the property of being N-free. An example is given in Fig.2. However, if all sub-posets of an N-free poset are N-free. then it is necessarily series-parallel.

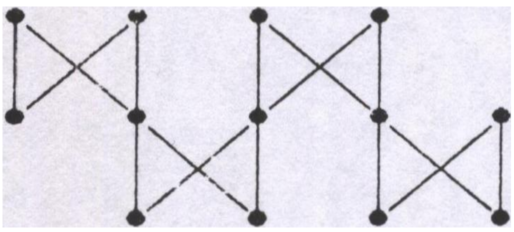


Fig .2 An N-free poset which is not hereditary under the removal of maximal chains.

There exist many different characterizations of N-free posets, among those let us recall the most important ones.

Theorem. For a poset P, the following statements are equivalent.

(i) P has the CAC property;

(ii) P is N-free;

(iii) For all x, y ∈ P, ImdPred(x) ∩ ImdPred(y) = ∅ or ImdPred(x) = ImdPred(y), where *ImdPred(x) denotes the set of lower covers of x in P.*

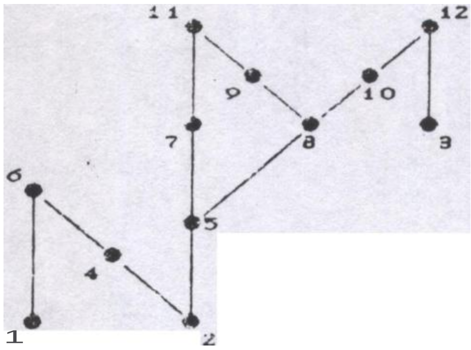
§3. Two Representations of N-Free Posets.

In this section, we will present some selected representations of N-free posets that are the block and the matrix representations. Property (iii) of the above theorem states that the set of immediate predecessors (lower covers) of vertices form a partition of P. This is a necessary and sufficient condition for a *dag* (directed acyclic graph) to be a line graph of N-free poset. According to this property we describe the *block representation of N-free poset* as follows [2].

Let P be a finite N-free poset. A *block* of P means a maximal complete bipartite graph in the directed covering graph of P. More precisely a block of P has the form (A.B) where A, B ⊆ P are such that A is the set of all upper covers (in P) of every y ∈ B and B is the set of all lower covers of every x ∈ A. By convention, (Min P,∅) and (∅,Max P) are also blocks where Min P and Max P are respectively the minimal and maximal elements of P.

Let $(A_1,B_1),...,(A_k,B_k)$ be all the blocks of P. Note that for any two elements x, y ∈ P property (iii) of the above theorem must be true. Thus the A_i's form a partition of P and so do the B_i's.

we shall always assume that the blocks of P are ordered such that for any x ∈ P if $x \in A_i$ and $x \in B_j$ then i < j . We get the block representation of P by filling a 2 × k array with the A_i's in the first row and the B_i's in the second row in the above order. This is illustrated in fig . 3.



P

A_i	1,2,3	4,5	6	7,8	9.10	11	12	∅
B_i	∅	2	1.4	5	8	7,9	3,10	6,11,12

Fi.g .3 The poest P and its block representation.

Clearly every N-free poset has a unique block representation apart from a possible permutation of the columns in the array. It is very difficult to use the block representation of an arbitrary N-free poset on a computer because of the use of the SET facility, which needs more running time. Therefore, we chose to deal with the dual representation, that is, with the matrix representation of P, [2].

Let P be an N-free poset with n elements and k blocks (A_1, B_1), …, (A_k, B_k) ordered as before. Define a $k \times k$ matrix $M(P) = [m_{ij}]$ where $m_{ij} = |A_i \cap B_j|$. The prescribed order of the blocks implies that $m_{ij} = 0$ whenever $i \geq j$, that is, M[P] is a super diagonal matrix. Again M(P) is unique up to a possible permutation ∂ applied simultaneously to the rows and the columns. The following matrix is an illustration of M(P), where P is that of Fig.3.

$$\begin{vmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Fig. 4 The matrix representation of the poset that given in Fig. 3.

This correspondence relation between the matrices and N-free posets leads to the construction of the described below algorithms for counting N-free posets.

§4. <u>Counting Prime N-Free Posets.</u>

Here we develop a program to count special type of N-free posets of n elements. This program is constructed to get the list of all compositions of n in inverse lexicographic (lex) order. For each composition with k parts, $1 < k < n$, the program creates all associative matrices in inverse colexicographic (colex) order, that satisfy the characterization of matrix representation. For programming simplicity the authors use lower triangle matrices which are transpose of super diagonal matrices after excluding the first column and the last row. The first matrix of the list is constructed by putting the k parts at the main diagonal of a $k \times k$ matrix. The next matrix in the inverse colex order is obtained by replacing pivot row, m, with the next composition of the part number m. Then modify the above m-1 rows by putting the values of m-1 parts on the main diagonal and leave the other k-m rows without change.

Then the algorithm tests whether this matrix represents a poset having one component or not. If not the matrix will be excluded and the next one will be created.

The next step is to check the uniqueness of the matrix under consideration (The matrix is a unique iff its main diagonal has no zeros). In case the matrix is not unique the test of isomorphism must be applied. The suggested algorithm is used to get all possible permutations that can be applied simultaneously on the rows and the columns of the matrix. In the same time the algorithm counts the number of permutation that lead to accepted matrices (TP) and the number of permutations that lead to the same matrix (EP). The first set of permutations is the elements of a permutation group whose cardinality is TP and the second set is the elements of the above group under which the object left invariant.

According to Burnside's lemma [6], *"The number of equivalent classes* is *equal to the average takes over the group, of the number of elements that are left* invariant *by a group element* " we obtain the number of distinct posets = EP/TP.

The result of this program is the number of connected N-free posets. Applying Stanley's results, [11], we easily computed the total number of N-free posets.

Unfortunately, the running time of this version increases very rapidly and it can't be executed for $n \geq 13$, (the running time at n = 12 on PC/XT with 8 MHZ 8088-1 processor takes more than 200 hours). So, the need of a modification to reduce this running **was** essential. Since the test of isomorphism needs running time O (the factorial of the order of the matrix).

Then, the algorithm was

modified to deal only with matrices that represent prime N-free posets. This modification leads to the following improvements :-

(1) The number of compositions of n that used is reduced, (i.e., it is much more less than 2^{n-1}). All parts of a composition is not exceed its position, n = $\sum_{i=1}^{k} R_i$, where $R_i \le i, R_{k-1} \neq 1$ and $R_k < k$.

(2) All matrices have 0 and 1 entries only. Since if there exists $m_{ij} \geq 2$ this means that this poset has an autonomous set of at least 2-elements.

(3) The number of matrices that will be tested for isomorphism is much more less than before.

The result of these improvements reduced the running time to 18 hours only and we hope to get the number of prime N-free posets with n > 12.

Note that : henceforth we will use the following declared type.

```
Type
      matrix       :array [ 1..integer_no, 1..integer_no] of integer
      vector       :array [ 1 ..integer_no] of integer
      vector-set   :array [ 1 ..integer_no] of set of 1..integer-no
```

A. Assistant Algorithms.

Function *Sum* (RowNo,CoLNo : integer, A : matrix): integer

$$\{ \sum_{i=1}^{COLNO} A_{ROWNO,i} \}$$

```
begin
   Add ← 0
   for i ←1 to CoLNo do
          Add ←Add + A [RowNo,i ]
   Sum ←Add
end.
```

Function *Pivot_Row(n* : integer, Part : vector) :integer

```
   {Search for the first row, i, at which Part[i] ≠i }
begin
   i ← 1
   repeat
       i ← i + 1
   until(Part[i] ≠i )or (i = n)
   Pivot-Row ←i
end.
```

```
procedure      Adjacent_sets_of_Elements ( A : matrix,
                                            var Adj: vector_set)
 begin
   ElementNo  ←  1
   for i ←——      2 to n do
    for j  ←  1 to  i  do
        if  A[i,j]≠0
          then
              ElementNo ←   Element + l;     B[i,j] ←   ElementNo
   for i  ←   2 to n do
     for j  ←   1 to i do
        if  B[i,j]≠ 0
          then begin
                for k ← i+1 to n do
                   if  B[k,i+l] ≠  0
                     then begin
                          X[B[i,j]  ←   X[B[i,j]] ∪ {B[k,i+l]}
                          Y[B[K,I+l] ←   Y[B[k,i+l]]∪ {B[i,j]}
                       { X : array of upper adjacent sets,
                        Y  :array of lower adjacent sets}.
                        end
              end
    for i ←———1 to  n-2  do
     for j  ←    i+1 to n-1 do
      if j ∈ X[i]
       then begin
             for  k ←    j+l  to  n do
               if  k ∈ X[j]
                then
                  X[i] ←   X[i] ∪ {k};     Y[k] ←   Y[k] ∪ {i}
           end
   for i  ←   1  to  n  do

           Adj [i] ←   X[i]∪ Y[i]
           { Adj  :array of  total  adjacent  sets}
 end.

Procedure      CycLe_Type_Of_Mapping (π : vector, A :matrix,
                                       var Cycle_type :vector)
        {Get the cycle type of automorphism mapping
              which is(j_1,j_2,...,j_p) s.t. P = ∑_{i=1}^{P} i j_i}
begin
  for i ←—— 1 to   n  do
    for j  ←   1 to  i do
      if  B[i,j]  ≠ ∅
        then begin
             L[B[i,j],l]  ←   i;      L[ B[i,j],2] ←  j
             L[B[i.j],3] ←   B[i.j]
            end
 k  ←   0
 repeat
   k ← k+1
   if  L[k,3] ≠ 0
```

```
    then begin
        i ← π [ L[k,1] +1] - 1;  j ← π [L[k,2]
        if(L[ k,1] = i)  and  ( L[k,2]= j)
          then begin
                 Cycle_type[1]  ← Cycle_type [1] + 1
                 (Cycle_type [i] is the cycle of length i)
                 L[k,3]         ←   0
               end
          else begin
                 Ln ←   1 {Ln is the length of cycle}
                 first_Element    ←   L [k,3]
                 Closed_Cycle     ←   false
                 repeat
                   r ←   k-1
                   repeat
                      r ←   r+1
                   until  (i = L[r,1]) and  (j  = L [r,2])
                   Next_Element ←   L [r,3];     L[r,3]← 0
                   if first_Element  = Next_Element
                     then begin
                            Closed_Cycle  ←  true
                            L[k,3]        ←  0
                            Cycle_type[Lnl ←  Cycle_type[Ln]+1
                          end
                     else begin
                            Ln  ←           Ln + 1
                             i  ←  π  [L [r,1]+1] -1
                             j ←    π  [L[r,2]
                          end
                 until  Closed_Cycle
               end
        end
  until k = P
end.
```

B. Basic Algorithms.

Algorithm(1).

```
Procedure Get_first_Comosition  (P  : integer , var  n        :integer,
                                                var  Part     :vector)
         {Get the first composition of P in the inverse lex
         order list that satisfy the following condition:
```

R_i= i for $1 \leq i \leq n-1$, $R_{n-1} \neq 1$ and $2 \leq R_n \leq n-1$ }

```
 begin
   Part[1] ←   1;    n ←    1;    Temparory ←   P-1
   repeat
       n  ←n + 1; Part [n] ←  n
       Temparory  ←    Temparory  -  n
   until  Temarory <  n+1
   if Temparory  = 0
     then
       Part[n] ←   Part[n] - 2;    n ←   n + 1;   Part[n] ←   2
```

```
    else begin
           if Temparorv = 1
             then
                 Part [n-1] ← Part[n-1] -1;   Part[n] ← 2
              else
                 n ← n+1;   Part [n] ← Temparory
         end
  end.

Procedure Get_Next_Composition (P : integer , var n :integer,
                                          var Part : vector)
   {Get the next composition of P in the inverse lex
    order that satisfy the following condition :
    1 ≤ R_i ≤ i for 1 ≤ i ≤ n-1, R_{n-1} ≠1 and 2 ≤ R_n ≤n-1}
begin
   i ← 0
   repeat
        i ← i+1
   until Part[i] ≠1
   X ← Part[i];   Part[i] ← 1
   if i = n
     then begin
             Part[n] ← 2;   n ← n+1
             Part [n] ← 2;   X ← X-2
          end
     else begin
             if Part[i+1]+1 > i+1
               then begin
                       j ← i;   Done ← false
                       repeat
                         j ← j+1
                         if Part [j]+1 > j
                            then
                              X ← X + 1;   Part [j] ← 1
                            else
                              Part[j] ←Part[j]+1;   Done ←True
                       until Done
                       if' j > n
                         then begin
                                Part [j] ← 2;
                                X ← X-1;   n ← n+1
                              end
                    and
               else Part[i+l] ← Part[i+l] +1
          end
   X ← X-1
   if X > 2
     then begin
             j ← 1
             repeat
                j ← j + 1;   X ← X + Part[j]
                if X < j
                  then
```

```
                    Part[j]  ← j;                    X← 0
                 else
                        Part[j]← j;                  X ← X − j
             until  X= 0
          end
    else  Part[2]  ←    1
  if Part [n-1]  - 1
    then begin
            i ←   n-1
            repeat
                 i ←   i-1
            until (Part[i]≠1) or (i = 0)
             if i ≠ 0
               then begin
                     Part[i]   ←  Part[i]  - 1
                     Part[n-1]←  Part[n-1] + 1
                  end
              else Get_Next_Composition
        end
  if Part[n]  =n
    then Get_Next_Composition
end.
```

Algorithm (2).

```
Procedure Create_first_Matrix (n:   integer,    Part :vector,
                                        var  A  :matrix)
 begin
   for i←    1 to  n do
    for j←    1 to Part[i] do
                     A[i,i+l-j]← 1
   for i←    1 to n do
    for j ←   i to n do
             A[n+l,i]  ←   A [n+l,i]+ A [j,i]
 end.
```

```
Procedure   Create_Next_Matrix (n , m  : integer,  Part  :vector,
                                           var   A :matrix)
```

{Create next matrix that represents N-free poset. First, we change the pivot row , m . Then modify the upper m-1 rows and leave the other n-m rows without change}.

```
begin
    i ← 1
    repeat
        i← i+l
    until A[m,i]≠   0
    A[m,i]  ←   0;    X ←   A[m,1] + 1;    A [m,1]  ←    0
    if X > i-2
      then begin
             repeat
               j  ←   i-1
               repeat
```

```
                j ←  j+1
             until A [m,j]≠ 0
             A[m,j) ←  0;      X ←  X+1
           until X ≤  j-1
           i←j
         end
      for j ←1 to X do
             A[m,i-j] ←1
      for i ←2 to m-1 do
             begin if Part [i]≠i

             then begin
                    for j ←   1 to Part[i] do
                              A[i,i+1-j] ←  1
                     for j ←   i - Part[i] downto 1 do
                                        A[i.j] ← 0
                   end
       end
   for i ←—— 1 to m do
       begin
           A[n+1,i] ←  0
           for j ←  i to n do
                       A[n+1,i ]←    A[n+1,i]  + A [j,i]
       end
 end.
```

Algorithm (3)

```
Procedure  Test_Connectedness   (n : integer, A :matrix,
                                 var  Poset_Connected :boolean)
     {Search for this matrix that represents a poset
     with one component or more. In the first case we
     return with Poset_Connected is true}

     Procedure Get_Path (D : integer)
          var
                i , j :integer
       begin
         for i ←2  to  D do
            if A[D,i]≠ 0
               then begin
                      B ←B U { i-1}
                      if i≠2 then Get_Path(i-1)
                    end
       end.
begin
  L ←0 {L : the number of components}
  for k ←2 to  n do
    begin
      if A[n,k]≠ 0
        then begin
               B ←  {k-1}  {set of adjacent elements of k}
                if k ≠2 then Get_Path  ( k-1)
             if  L = 0
```

```
                    then begin
                          L ←1; C[1]  ← B
                          {C is an array of sets that represent
                             disjoint componenets of a poset}
                        end
                    else begin
                          j  ←   0
                          repeat
                               j  ←  j  + 1
                          until ( B ∩ C[j]≠ ∅) or (j= L+1)
                          if j =L+1
                            then    L ←   L+1;     C[L] ←   B
                            else begin
                                    q ← j ;     C[q]←C[q) ∪B
                                  repeat
                                       j ← j  + 1
                                  until (B ∩C[j]≠ ∅) or(j=L+1)
                                  if j≠L+1
                                    then begin
                                           V       ← j– 1
                                            C[q] ←  C[q] ∪C[j]
                                              for i ← j+1 to L do
                                               if (B ∩ C[i])≠∅
                                                 then
                                                  C[q]  ←C[q] ∪ C[i]
                                                 else begin
                                                        V    ← V + 1
                                                        C[V] ←  C[i]
                                                      end
                                           L ← V
                                         end
      end
    if L = 1
      then  Poset_Connected ←  true
            {A matrix represents a connected prime N-free poset}.
      else  Poset_Connected ←   false
            {A matrix represents a disconnected prime N-free poset}.
end.
```

Algorithm(4)

```
Procedure Check _Prime (n: integer, A: matrix,
                        var Poset_Prime : boolean)

    {Decide whether this matrix represents prime poset or not.
      In the first case we return with Poset_Prime is true}.
begin
  Adjacent_Sets_Of_ELements(A, Adj)
  Poset_Prime ←   false
  for i ←    1 to n do
      if Adj [i] = 0 then return
  for L ←    1 to n-1 do
   for k ←    L+1 to  n do
```

```
begin
  A1←{L} ; A2 ← {L..K}
  {A1,A2 : two sets are used for testing if their exist an
         autonomous set}
  repeat
    Untested_Elements ←A1/A2;        A1 ←  A2
     i ← L-1
    while A1 ∩{1..i} ≠ Ø do
      begin
       i ←  either i+1 if (i+1) ∈ A1 or min(A1) > i+1
       while Untested_Elements ≠ Ø do
        begin
             j ←  either i+l if (i+1 ) ∈ Untested_Elements
                    or min(Untested_Elements) > i+1
             R ←  (Adj[i]/Adj[j] ∪ (Adj[j]/Adj[i])
             if R ∩({1..k}/{L,k}) = ∅
               then go to 100
             A2 ←  A2 ∪ R
            if A2 = {1..n}
               then  go to 100
             Untested_Elements ←  Untested_Elements/{j}
        end
      end
  until A1 = A2
 return
100:end
 Poset_Prime ←  true
end.
```

Algorithm (5)

```
Procedure      Test_Isomorphism (n, Pos : integer, A : matrix,
                                 var Equ, Total :integer)
   {Apply all possible permutations. that create by the Johnson-
    Trotte algorithm, simultaneously on the rows and the columns of
    A to get the number of matrices which are isomorphic with A}
begin
   for i ←—— 1 to n+1 do
                 π⁻¹[i] ←i;  π[i] ←—— i;  d[i] ← -1
   A←   {Pos+1...n};        Last._Perm ←  false
   Total ←  1;               Equ      ←  1
           {Equ is the number of' automorphic posets
            and Total is the number of' isomorphic posets}
   Cycle_type_Of_Mapping (π, A, C)
   while not Last_Perm do
      begin
        if A ≠ Ø
          then begin
                 m← max{i, i ∈ A};    j← π⁻¹[m]
                 Move ← false
                if (A[m-1,π[j+d[m]]=0) and (m > π [j+d[m]])
                 then begin
                        π[j]  ← π[j+d[m]];        π[j+d[m]]←m
```

```
                        π⁻¹[m] ←  j + d[m];      π⁻¹[π[j]] ← j
                         Move  ← true
                      end
                  else begin
                       if m < π[j+d[m]]
                        then
                           d[m] ← - 1;      A ←   A/{m}
                        else if (d[m]=-1) and (Move)
                             then begin
                                  A     ←  A U {m+1...n}
                                  Total ←  Total. + 1
                                  for i ←  1 to n do
                                   for j ←  1 to ido
                                    if A[i,j]≠A[π[i+1]-1,π[j]]
                                      then go to 200
                                  Equ.  ←  Equ + 1
                                  Cycle_type_Of_Mapping (π ,A,C)
                                end
                end
            else  Last_Perm ←  true
200: end
end.
```

Main Program.

step 0 Find P and MaxNo
{P : number of points
MaxNo: maximum number of parts of composition of P}
Set $P_P \leftarrow 0$
{P_P : number of prime N-free posets of P points}.

step 1 *Get_First_Composition* (P, N, Part)
{N: number of parts of composition of P.
Part : the array of n elements whose elements are n parts of composition of P}

step 2 if N = MaxNo then *go* to *step 16*

step 3 *Create_First_Hatrix* (N, Part, A)
{A : an n+1 × n matrix that represents a labeled N-free poset}.

step 4 if A[1,1]≠ A[N+1,1]
then *Check_Prime* (P, N. A, Poset_Prime)
else *go* to *step 6*

step 5 if Poset_Prime then $P_P \leftarrow P_P + 1$

step 6 M ← *Pivot_Row* (N, Part)
{M : the pivot row at which Part[M]≠M}.

step 7 if ((M=N) and (A[N,1]≠ 0)) or ((A[N+1,M]≠1) and (A[M,M]≠1)
then *Create_Next_Matrix* (N, M, Part, A), go to *step 9*

```
step 8   Set M ←M-1
         repeat
             M ←   M + 1
            while Sum(M, Part[M], A) = Part[M]  do M ←M + 1
         until (A[N+1,M]≠1)  or   (A[M,M]=0)  or
             ((Sum(M, Part[M]-1,A)≠Part[M]-1) and Part[M]≠1))
         Create_Next_Matrix   (N, M, Part, A)

step 9  if A[1,1] = A[N+1,1],
                   {this poset has a unique minimal element}
           then M ←2 go to step 14

step 10 if A{N,1] = 0.
                   {this poset has no isolated point}
          then Test_Connectedness(N . A. Post_Connected).
          else M ←N,    go to step 14

step 11 if Poset_Connected
            then Check_Prime(N , A,  Poset Prime)
            else M ←   Pivot_Row ( N, Part), go to step 14

step 12  if not (Poset_Prime)
           then M ←   Pivot_Row (N, Part), go to step 14

step 13  i ←1
         repeat   i ←i + 1        until  (A[i,i ] = 0)  or  (i = N)
         if i = N
            then P_p ←   P_p + 1
            else begin
                     Test_Isomrphism  (N,  i , A, Equ, Total)
                     P_p ←  P_p +  Equ/Total
                 end

step 14  if Sum(N, Part[Ml-1, A)≠Part[Nl - 1
           then go to step 7

step 15 Get_Next_Composition  (P, N, Part),
        go to step 2

step 16 stop.
```

Finally, the number of N-free posets, f_n, can be obtained as follows:

(1) Executing the program one can count the number of prime N-free posets, P_n and determine the cycle index polynomial of automorphism groups (Tables I and II in the appendix).

(2) Applying Pólya's enumeration theorem [6] with knowing P_n

and cycle index polynomials compute the numbers of N-free posets, i_n, that produced by substitution composition.

(3) According to Stanley's results [11], the numbers of Connected, v_n, and total N-free posets can be easily determined by using the computed numbers i_n.

Note that for details see [1], the authors computed the numbers of 2-dimensional posets via counting the numbers of prime 2-dimensional posets by using the same method.

At the end of the paper, we must record that MOhring's results given in [8] are incorrect *for* n = 10 not only in the case of 2-dimensional posets (see [1]) but also of N-free posets.

Appendix.

P_n : number of Prime N-free posets of n elements.

i_n : number of irreducible N-free posets w. r. t. series and parallel compositions of n elements.

v_n : number of connected N-free posets of n elements.

u_n : number of disconnected N-free posets of n elements.

f_n : number of **total** N-free posets of n elements.

Table I.

n	P_n	i_n	v_n	u_n	f_n
1	0	1	1	1	1
2	0	0	1	1	2
3	0	0	3	2	5
4	0	0	9	6	15
5	1	1	31	19	49
6	0	10	115	75	180
7	7	72	474	313	715
8	15	456	2097	1440	3081
9	73	2791	9967	7041	14217
10	304	16965	50315	36555	69905
11	1456	104241	268442	199725	363926
12	7185	652650	1505463	1144109	1996922

$Z(\Gamma(P))$: the cycle index polynomial of automrphism group, $\Gamma(P)$, of prime N-free poset, P.

ζ_i^m : m cycles of length i.

Table II

n	$\lvert\Gamma(P)\rvert \times Z(\Gamma(P))$	P_n
5	ζ_1^5	1
7	ζ_1^7	5
	$\zeta_1^7 + \zeta_1 \ \zeta_2^3$	2
	ζ_1^8	12
8	$\zeta_1^8 + \zeta_1^2 \ \zeta_2^3$	2
	$\zeta_1^8 + 2\ \zeta_1^2 \ \zeta_2^3 + \zeta_2^4$	1
9	ζ_1^9	65
	$\zeta_1^9 + \zeta_1^3 \ \zeta_2^3$	8
	ζ_1^{10}	274
	$\zeta_1^{10} + \zeta_2^5$	1
	$\zeta_1^{10} + \zeta_1^2 \ \zeta_2^4$	3
10	$\zeta_1^{10} + \zeta_1^4 \ \zeta_2^3$	22
	$\zeta_1^{10} + \zeta_1^2 \ \zeta_2^4 + \zeta_1^4 \ \zeta_2^3 + \zeta_2^5$	2
	$\zeta_1^{10} + 3\ \zeta_1^4 \ \zeta_2^3 + 2\ \zeta_1 \ \zeta_3^3$	2
	ζ_1^{11}	1334
	$\zeta_1^{11} + \zeta_1 \ \zeta_2^5$	11
	$\zeta_1^{11} + \zeta_1^3 \ \zeta_2^4$	13
	$\zeta_1^{11} + \zeta_1^5 \ \zeta_2^3$	90
11	$\zeta_1^{11} + 3\ \zeta_1^5 \ \zeta_2^3 + 2\ \zeta_1^2 \ \zeta_3^3$	2
	$\zeta_1^{11} + \zeta_1^5 \ \zeta_2^3 + \zeta_1^3 \ \zeta_2^4 + \zeta_1 \ \zeta_2^5$	4
	$\zeta_1^{11} + \zeta_1^3 \ \zeta_2^4 + 3\ \zeta_1^5 \ \zeta_2^3 + 3\ \zeta_1 \ \zeta_2^5 + 2\ \zeta_1^2 \ \zeta_3^3 + 2\ \zeta_2 \ \zeta_3 \ \zeta_6$	2

References.

[1] B. I. Bayoumi, M. H. El-Zahar and S. M. Khamis (1988), "Enumeration of 2-dimensional posets via counting prime 2-dimensional posets", The 23 th Annual Conference (Computer Science & its Applications) in Statistics, Computer Science and information and Operation Research, 50-70.

[2] B. I. Bayoumi, M. H. El-Zahar and S. M. Khamis, "Asymptotic enumeration of N-free partial orders", Order j., to appear.

[3] P. A. Grillet (1969). "Maximal chains and antichains". Fund. Math. 65, 157-167.

[4] M. Habib and M. C. Maurer (1979). "On the X-join decomposition for undirected graphs", Discrete Appl. Math. 1, 201-207.

[5] M. Habib and R. Mohring (1987), "On some complexity properties of N-free posets and posets with bounded decomposition diameter", Discrete Math. 63, 157-182.

[6] F. Harary and E. Palmer (1973), " Graphical enumeration", New York and London: Academic Press.

[7] B. Leclerc and B. Monjardet (1973). "Orders CAC", Fund. Math. 11-22.

[8] R. .Mehring (1985),"Algorithmic aspects of comparability graphs and interval graphs" ,in Graphs and order , Reidel ,Dordrecht, 41-101.

[9] R. H. Mohring and F. J. Radermacher (1984), "Substitution decomposition for discrete structures and connections with combinatorial optimization", Ann. Discrete Math. 19, 257-356.

[10] I. Rival (1983), "Optimal linear extensions by interchanging chains", Proc. Amer. Math. Soc. 89, 387-394.

[11] R. P. Stanley (1974), "Enumeration of posets generated by disjoint unions and ordinal sums". Proc. Amer. Math. Soc. 45 (2), 295-299.

[12] M. M. Syslo (1985), "A graph theoretic approach to the jump number problem" in I.Rival, ed. Graphs and order, (Reidel, Dordrecht), 185-215.