Proceedings of the
International Conference
On Mathematics:
Trends and Developments

Egyptian Mathematical Society
Cairo 28-31 Dec.2002
Vol (V) Functional Analysis
pp111-120

PRIMALITY TESTING AND COUNTING THE NUMBER OF UNLABELED PRIME POSETS ON THIRTEEN ELEMENTS

S.M. Khamis

Mathematics Department, Faculty of Science, Ain Shams University, Cairo
Egypt

Abstract.

This paper gives a new polynomial algorithm for testing whether or not a given poset, P , is prime.

The basic idea of the given algorithm depends on finding a proper autonomous set of elements of P.

The algorithm decides that P is prime if each pair of elements of P does not belong to a proper autonomous set. Via this test, unlabeled prime posets on n elements are counted. The strategy of the suggested counting algorithm is: (1) Create an upper triangular matrix satisfying the bucket form; (2)

Apply the primality test to accept a constructed matrix if it corresponds to a prime poset; and (3)

Compute the weight of the accepted matrix of enumeration. The three steps are repeated to obtain

the sum of the weights of all upper triangular matrices with bucket form that correspond to prime posets. In constructing these matrices, we follow a colexecographic (colex.) order w.r.t. rows. As a result, the previously known numbers of such posets on $n \leq 12$ recorded in [3] 1997 are verified. A new

number on n = 13 is computed.

AMS (2000) : 05A15, 06A07, 05C30.

Key words. Counting, enumerating, partially ordered sets, finite poset, unlabeled, prime, and algorithm.

§1. Introduction.

In [6], one can find some important information about enumeration problems of different classes of posets and their comparability graphs which is summarized in the following :

Given a class P of finite posets, two enumeration problems are automatically arisen, namely how many posets in P with n elements (counting of labeled posets)? and what is the number of their isomorphism classes (counting of unlabeled posets)?

However, not all classes of posets are easy to enumerate, so answers to these problems have different forms. The calculations can be manipulated in the algebra of generating functions, if a sufficiently strong structure theorems can be obtained. The required numbers are then calculated directly or recursively from the resulting generating functions. Examples for this exact counting include: labeled and unlabeled series-parallel posets (regardless height of posets) in [18], unlabeled series-parallel posets according to height in [7], labeled and unlabeled interval graphs [10], and interval orders [6]. The author et. al. gave exact counting for unlabeled 2-dimensional posets and permutation graphs, respectively, in [2] and [12]. Recently in [14], the author derived the generating function for counting interval orders according to height.

Unfortunately, for most classes of posets, an exact counting can't be successfully found. So, in such cases, it is sufficient to find an upper bound or an asymptotic estimate of the required number. For example: asymptotic estimate for unlabeled posets, see [15] and for labeled and unlabeled N-free posets, see [1].

Although the enumeration of all posets or, equivalently, of all T_O topologies on a finite set is intrinsically difficult, some researchers used computer programs to obtain the required numbers. Up to now, there exist several algorithms which have been constructed to generate all posets (of a small number of elements) and count them. In this field, the numbers of labeled posets F_n and labeled topologies T_n for $n \leq 7$ have been found in the earliest paper [9]. In 1991, Ernr and Stege [8] extended the values of F_n up to $n \leq 14$. For unlabeled posets, Mohring in [16] recorded results for $n \leq 10$. The algorithm that was developed by Culberson and Rawlins [5] gave the numbers of all unlabeled posets with $n \leq 11$ elements. The results of this algorithm agrees with Mohring's results, [16], except for $n = 10$. In [4], C. Claunier and N. Lygerõs counted the number of unlabeled posets for $n = 12, 13$. In case of $n = 13$, they used nine Apollo workstations for six months each workstation generating on average number of about 250 orders per second. In [11],
J. Heitzig and J. Reinhold gave a new orderly algorithm to count the number of unlabeled posets on up to 14 elements. This algorithm generates about 30000 objects per second (on a DEC Alpha, 450 MHz), a single processor would still run more than 17 months for $n = 14$. So, they worked on 50 processors simultaneously to record the number. In the field of enumerating prime posets, there exist quite a few studies such as [16], [3] and [13]. In [16], one can find the numbers of unlabeled prime posets on only up to 10 elements. In [3], the numbers of types of unlabeled n-element UPO graphs and their related posets such as prime ones for $n \leq 12$ have been algorithmically counted. Furthermore, at $n = 10$ the previously known number in [16] was proved to be incorrect. While, in [13], the author applied the principle of height counting, introduced in [7], to obtain the numbers of unlabeled prime posets according to height for the same range. Up till now, there is no recorded number for the unlabeled prime posets with $n \geq 13$.

In this paper, the author describes a new polynomial time algorithm to decide whether a given poset is prime. This test enables us to count the exact number, P_n ,of

unlabeled prime posets on n elements. Henceforth we use the following matrix-representation of a poset: Suppose P is a poset on the set $\{1,2,\ldots,n\}$. We say that P is natural if whenever $x <_p y$ then $x < y$ in their natural order. The poset P is represented by an adjacency matrix $A = [a_{ij}]$ where $a_{ij} = 1$ whenever $i <_p j$ and 0 otherwise. P is restricted to be a natural order and moreover the sets of successors of elements of P are in a bucket sort, i.e. their cardinalities are in a nondecreasing order. To obtain P_n , we use an algorithm for creating these matrices. The suggested property avoids creating too many matrices representing the same poset. Matrices created in this form are checked for primality of the corresponding posets. To each accepted matrix A a weight $1/\mathcal{R}$ is assigned where $\mathcal{R}$ is equal to the number of different matrices representing the same poset as A does. The number $\mathcal{R}$ is calculated from the automorphism group of A. By introducing these weights we could avoid storing too many matrices and comparing them for isomorphism. The required number P_n is calculated by summing up the weights of all accepted matrices.

Section 2 introduces the basic concepts and the underlying definitions. Section 3 contains the description of the primality test. The technique of counting P_n is given in Section 4. Finally the appendix contains the values of P_n for $n \leq 13$.

§2. Definitions and Basic Concepts.

The terminology, notation and concepts will follow that of [1], [6] and [16].

A *partially ordered set* (poset) P is defined by $(V,<_p)$, where V is a nonempty set of elements and $<_p$, or $<$ if P is understood, is a partial order relation on V, that is an irreflexive, antisymmetric and transitive relation. Two elements $a, b \in V$ are *comparable* ($a \sim b$) in P if $a < b$ or $b < a$, otherwise they are *incomparable* ($a \| b$). While b covers a ($a \prec b$) if $a < b$ and there is no $c \in V$ with $a < c < b$. A set of pairwise *incomparable* (resp. *comparable*) elements is called an *antichain* (resp. a *chain*).

Let $P' = (V',<')$ be a poset with vertex set $V' = \{v_1, \ldots, v_m\}$ and let $P_1 = (V_1,<_1), \ldots, P_m = (V_m,<_m)$ be posets with disjoint vertex sets. The *lexicographic product* $P'[P_1,..,P_m]$ is the poset $P = (V,<)$ where $V = V_1 \cup \ldots \cup V_m$ and $a < b$ iff either one of the following conditions is satisfied:

(i) for some $i, a, b \in V_i$ and $a <_i b$, or

(ii) $a \in V_i$ and $b \in V_j$ and $v_i <' v_j$.

It is clear that P is obtained by substituting P_i for each vertex v_i of P'. The posets $P_1, \ldots, P_m$ are called the *inner posets* and P' is the *outer poset* or the *quotient poset*. The poset P is called *decomposable* if $P = P'[P_1 , \ldots, P_m]$ where $m > 1$ and at least one of the P_i's is nontrivial, i.e. has more than one element. An *indecomposable* poset is called *prime*. Observe that if we take P' to be a chain (an antichain) then we get the ordinal sum (disjoint union) of $P_1,\ldots, P_m$.

The operation of substitution plays an essential role in solving many combinatorial optimization problems on posets. The main idea of such divide-and-conquer techniques is to solve the problem for each of P', $P_1 , \ldots, P_m$ and combine these solutions to get a solution for $P = P'[P_1 , \ldots, P_m]$. To name a few of such optimization problems we mention :minimum covering by chains / antichains, determining the dimension and the mobious function and counting partial orders (see [16]).

This importance of the substitution and the inverse operations: decomposition of posets into prime ones mandates the search for algorithmic methods to recognize and count prime posets. This algorithmic method for recognizing a prime posets and other for counting the number of prime posets are two main problems which we tackle here.

There is another characterization of prime posets. Suppose $P = (V,<)$ is a poset and $Y \subseteq V$. Then Y is called *P-autonomous* if $\forall$ a, $b \in Y$ and $\forall\, c \in V \setminus Y$, we have: $b < c \Leftrightarrow a < c$, and $c < b \Leftrightarrow c < a$. Y is proper iff $1 < |Y| < |V|$. It is clear that if $P = P'[P_1, \ldots, P_m]$ then each P_i is P-autonomous. Thus a poset P is prime if and only if P has no proper autonomous subset. This property of prime posets is employed in our algorithm to recognize such posets

§3. Primality Testing.

This section contains the description of a polynomial time algorithm, for testing whether or not the given poset P is prime. Recall that a poset is prime iff it has no proper autonomous set. Thus the basic task of the suggested method is: for any two elements $l, k \in V$, the algorithm finds a proper autonomous set that contains l,k. If such a set exists, then the given poset is not prime. Otherwise, if each pair of elements of P does not belong to a proper autonomous set, then P is prime. This method is described in the following procedure:

Data structure needed:

(i) $A_1, \ldots, A_n$ are n lists, where A_i consists of all elements comparable to element i in P,
(ii) L is a priority queue with the natural priority of integers,
(iii) Include is a 1-dimonsional array of integers whose entries have value one if corresponding elements belong to a possible current autonomous set and zero otherwise,
(iv) Counter is an integer variable used to count the number of elements that belong to a possible autonomous set corresponding to the current pair of elements of P,
(v) Autonomous is a boolean variable having the value true if a collection of elements w.r.t. the given pair of elements of P is an autonomous set, false otherwise,
(vi) i, j, k, l are integer variables for the elements of P,
(vii) TestSet is a set of integers.

Procedure ExistAutonomousSet(*l*,k; Autonomous);

```
   Begin
1)      For i ← 1 to n Do  Include[i] ← 0;  od;
2)      Autonomous ← False;  Include[l] ← 1;   Include[k] ← 1;
3)      L ← {k};          Counter ← 2;
4)      While L ≠ ∅ Do
5)        j ← Min[L];          //Get the first element of a priority queue.
6)        DELETEMIN(L);  //Delete the first element of a priority queue.
7)        TestSet ← (A_l –A_j ) ∪ (A_j –A_l );
8)        If there exists i ∈ TestSet   s.t. 1≤ i <l ∨ l < i <k Then  return  fi;
9)        For  i ← k+1 to n Do
10)           If  i ∈ TestSet Then
11)             If Include[i] = 0 Then
12)                 Include[i] ← 1;
13)                 Counter ← Counter +1;
14)                  Insert i in L according to its value;
15)             fi;
16)           fi;
17)       od;
18)       If Counter=n-k+2  Then  return  fi;
19)     od;
20)     Autonomous ← True;
   End;
```

The primality test checks whether a given poset P is prime or not via repeating the procedure ExistAutomonousSet for pairs of distinct elements of P. All

required pairs (l,k) where $1 \le l < k \le n$ must satisfy either $l \parallel k$ or $l \prec k$ and the sets of predecessor elements of l and k whose labeled are less than l , are the same. This process is halted when a proper autonomous set including the given pair is found. If there exists no such set, the test decides that the current poset is prime.

Obviously, the complexity status of the primality test depends on the status of the procedure ExistAutonomousSet multiplying by the number of used pairs. In ExistAutomonousSet, the cycle from step(4) to step(19) does not repeat more than $n - k$ times, where $1 \le l < k \le n$. Each cycle has at most n steps of O(1). Therefore the complexity of the procedure is at most $O(n^2)$. Hence the total running time of the primality test in the worst case is $O(n^4)$. We remark that a code of the procedure runs more faster than it is expected. This is because the number of used pairs is less than $\frac{1}{2}\, n\,(n-1)$ and also we don't need all of them if a poset is not prime. Moreover, if a poset is prime the while-Do loop (steps4-19) is not completed. So, this leads to count the numbers of unlabeled n-element prime posets on $n \le 13$ as shown in the following.

§4. Counting Prime Posets.

In this section, we describe an algorithm for counting unlabeled prime posets depending on the new primality test (see §3). The algorithm is based on the technique that is introduced in [3]. That technique counted unlabeled posets via constructing all corresponding upper triangular matrices in a certain order with some special properties which are described as follows:

Let $P = (V,<)$ be a natural poset, i.e., if $v_i < v_j$ then $i < j$. A natural poset P on n elements can be represented with an upper triangular square matrix $A(P)$ of order $(n+1)$. The submatrix of $A(P)$, obtained by omitting the 1st row and the last column is the adjacency matrix with the lower triangular part is zeroes .The last column is the rows sums, R_i, and the first row is the columns sums, C_i.

Apart from isomorphism, it is clear that there exists a one-to-many relation between unlabeled posets and the above defined matrices. To improve this relation, only the matrices whose rows appear in sorted form according to R_i are considered, i.e. bucket sort, (see Fig.1). This subclass of matrices is sufficiently enough to count the numbers of specified posets.

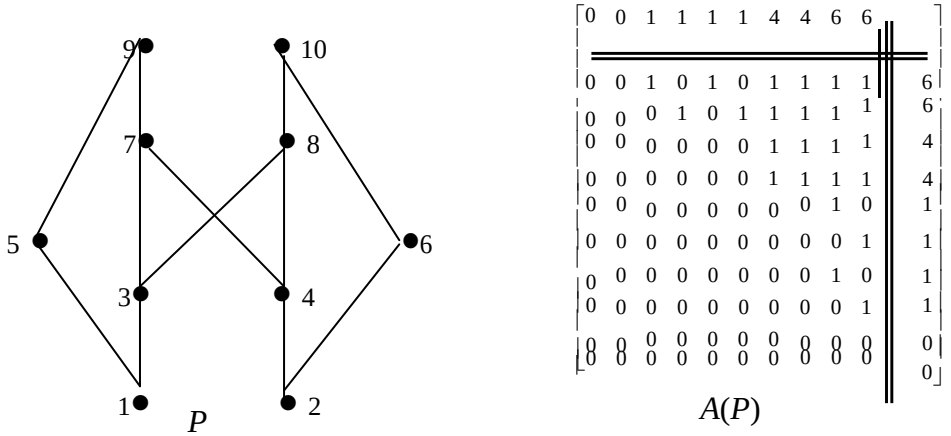

Fig. 1. The prime poset P and its corresponding matrix $A(P)$.

The strategy of the suggested method is: (1) Create an upper triangular matrix satisfying the bucket form; (2) Apply the primality test for accepting a constructed matrix if it corresponds to a prime poset; and (3) Compute the weight of an accepted matrix of enumeration. The three steps are repeated to obtain the sum of the weights of all upper triangular matrices with bucket form that correspond to prime posets. To

construct all these matrices, we follow a *colexecographic* (*colex.*) order w.r.t. rows of matrices. This ordering is defined as follows:

Let A and B be two distinct matrices of order n. Then A is before B in colex. order w.r.t. rows of matrices, if for some k the last $n - k$ rows of A and B are the same and the kth row of A precedes the kth row of B in *lexicographic* (*lex.*) order. Thus, The colex list of matrices begins with the zero matrix which represents the antichain of n elements. While it ends with the matrix whose elements in the upper main part (without the first row and the last column) are ones. Clearly, this last matrix corresponds to chain of n elements.

Now the steps of the algorithm are illustrated in the procedure CountAcceptedMatrix which is designed recursively below. The procedure uses the following data structures. Also, it requires some elementary procedures whose basic ideas given in [3]. So, some briefly explanation of them are stated after the procedure.

Data structure needed:

a) n is an integer variable for the number of elements of a poset,
b) A is a 2-dimonsional array of integers which represents a natural poset defined as above,
c) Pivot_Row and Pivot_Column are integer variables for determining the current location in A at which 0-entry will be changed to 1-entry,
d) Last-Matrix is a boolean variable for deciding whether or not the current array is the last one,
e) MatrixAccepted is a boolean variable for showing that A is in non increasing order w.r.t. columns sums,
f) Prime is a boolean variable for deciding whether or not the current array corresponds to a prime poset,
g) P_n is a real variable for counting the number of unlabeled prime posets on n elements.

```
Procedure    CountAcceptedMatrix(n,A;Last-Matrix,Pn);
  Begin
1)   GetNextLexicographicRow (A,Pivot_Row,Pivot_Column,Last-Matrix);
2)    If  not (Last-Matrix)  Then
3)     If (Pivot_Column <> n) Then
4)        Adjustable_Transitivity(Pivot_Row,Pivot_Column,A);
5)        UpdateMatrix(Pivot_Row,A);
6)     fi ;
7)     TestMatrixAccepted(A;MatrixAccepted);
8)      If  MatrixAccepted Then
9)        CheckPrime(A;Prime);     // given in §3.
10)         If Prime Then
11)            GetWeight(A;Pn);
12)         fi;
13)       fi;
14)     CountAccepted Martix(n,A;Last-Matrix,Pn);
15)    fi;
  End;
```

Elementary Procedures:

(1) GetNextLexicographicRow(*A*,Pivot_Row, Pivot_Column,Last- Matrix):

It is used to find the smallest index, Pivot_Row, of a row having at least one zero entry at the right of the main diagonal of A. When Pivot_Row = n , there is no next matrix in specified colex order and Last-Matrix returns with value true. Otherwise the procedure gets the successor of this row in lex. order. First the procedure finds the position, Pivot_Column, of the most right zero entry in the row. Then put A[Pivot_Row, Pivot_Column] ← 1

and A[Pivot_Row,j] $\leftarrow$ 0 for Pivot_Column < j $\leq n$. In this case, the outputs of the procedure are: (1) Pivot_Row and Pivot_Column having the values of the location of the changed zero element, and (2) Last-Matrix having the value false since A is not the last matrix.

(2) Adjustable_Transitivity(Pivot_Row,Pivot_Column,*A*):

The procedure modifies the entries A[Pivot_Row,j] where j varies from Pivot_Column+1 to n in order to satisfy the transitivity constraint. This is done as follows:

If A[Pivot_Row,j]=1 and A[j,i]=1 then put A[Pivot_Row,i]$\leftarrow$1, where Pivot_Row < j < i and Pivot_Column < i $\leq n$. The entry A[Pivot_Row,n+1] is updated by summing the entries of the pivot row.

(3) UpdateMatrix(Pivot_Row,*A*):

In this procedure, the rows of A above the Pivot_Row are updated without destroying the bucket sort w.r.t. rows sums. For 1 < i < Pivot_Row, the following assignments are done:

(1)A[i,n+1]$\leftarrow$$A$[Pivot_Row,$n$+1];

(2)A[i,j]$\leftarrow$0 for j = i+1, …, n-A[Pivot_Row,n+1]; and

(3)A[i,j] $\leftarrow$1 for j = n+1-A[Pivot_Row,n+1], …, n.

Then each entry A[1,j] is updated by computing the sum of entries of column j, where j = 2, …, n.

(4) TestMatrixAccepted(*A*;MatrixAccepted):

It is a simple test for accepting those matrices that are in the form of non increasing order w.r.t. columns sums. The procedure checks whether or not there is an i, 1< i $\leq n$ satisfying (A[1,i-1]>A[1,i]) and (A[i,n+1]=A[i+1,n+1]). If such an i exists, then the current matrix is excluded. In this case, the identifier MatrixAccepted returns with the value false.

The four procedures, altogether, construct in $O(n^2)$ an upper triangular matrix in the bucket form which represents a poset. Then in step(9), the current accepted matrix is checked for primality of the corresponding poset as shown in §3. Matrices accepted in step (9) represent labeled prime posets. While, our purpose is to enumerate the number of non-isomorphic (unlabeled) prime posets. So, the number, R, of all the matrices corresponding to the same poset that is represented by the current matrix must be computed to get the weight of this matrix. This is done via applying the procedure GetWeight (step(10)) which is designed as follows. First, we introduce some terminologies.

Again, let P be a poset having n elements. The elements of P can be divided into m distinct parts, $m \leq n$. Each part includes elements of P having the same number of predecessors and successors. Clearly, these elements will appear as consecutive rows and the corresponding columns in $A(P)$. For $1 \leq i \leq m$, assume that r_i is the number of elements in the ith part of P. Therefore, the numbers r_i form a partition of n into m parts, i.e.

$$n = \sum_{i=1}^{m} r_i, \quad \text{where} \quad 1 \leq r_i \leq n \quad \text{and} \quad m \leq n \ .$$

Also, the partition leads to divide the matrix into m vertical regions where the first region is the $r_1 \times r_1$ submatrix and the second is $(r_1 + r_2) \times r_2$ submatrix and so on until the last one which is submatrix of order $n \times r_m$. For example, Fig. 2 illustrate the partition of the matrix that is shown in Fig.1.

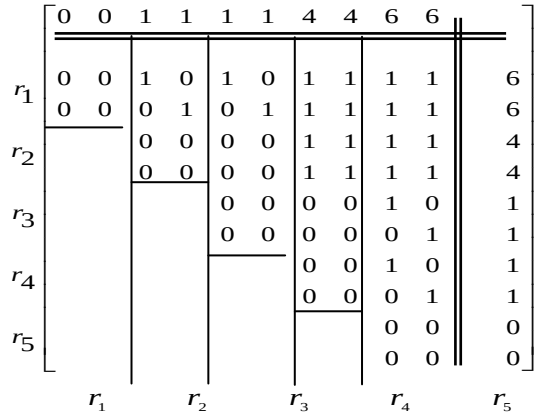

Fig. 2.

Let Γ denote the set of all permutations σ on $[n]$ such that σ permutes only elements in the same part of a poset. For $\sigma \in \Gamma$ denote by $\sigma(A)$ the matrix obtained from A by applying σ simultaneously on its columns and rows. In addition σ can be written as

$$\sigma = \alpha_{q_1} \alpha_{q_2} \ldots \alpha_{q_m},$$

where α_{q_i} is q_ith permutation in the list of all permutations on $[r_i]$ that are generated in lex. order for $1 \le i \le m$, and $0 \le q_i < r_i!$. It applies simultaneously on the columns of the region i in A and the corresponding rows. Obviously the number, K, of all permutations of Γ is given by $r_1! \, r_2! \ldots r_m!$. In fact, some permutations $\in \Gamma$ satisfy $\sigma(A) = A$. Let L be the number of such permutations. L is calculated by generating all possible permutations $\sigma \in \Gamma$ satisfying $\sigma(A) = A$. Then R which equals $|\{\sigma(A) : \sigma \in \Gamma\}|$ is given by $\mathcal{K}/\ell$ and hence the weight of A is assigned $1/\mathcal{R}$. Finally, the following recursive algorithm is devoted to calculate the weight of A via creating and counting all possible permutations $\sigma \in \Gamma$ such that $\sigma(A) = A$.

Data Structure needed:

a) m is an integer variable for counting the number of parts that are partitioned the elements of P via using the corresponding matrix A (as defined above),

b) i is an integer variable for finding the submatrix corresponding to the current part,

c) R is a 1-dimentional array of integers whose entries R[i] are for determining the number of elements of P in the ith part via using the corresponding matrix A,

d) q is a 1-dimentional array of integers whose entries q[i] are for counting the number of already created permutation on [R[i]],

e) Per is a 1-dimentional array of integers whose entries Per[i] are for storing the current permutation α_{q_i},

f) $\ell, \mathcal{K}$ are integer variables for computing the number of identical copies of A and the number of all permutations that can applied on A respectively.

```
Procedure GetWeight(A;Pn);
  Begin
1)  R[1] ← 1;   m←1;
2)  For i←2 to n Do        //Partition A into m vertical regions
3)     If (A[i,n+1]=A[i+1,n+1]) AND (A[1,i-1]=A[1,i])
4)           Then   R[m] ← R[m]+1
5)           Else       m ← m+1; R[m] ← 1;
6)     fi;
```

```
 7)  od;
 8)  If  m=n   // There is no isomorphic copy to the corresponding poset of A
 9)      Then  Pn ←Pn+1; return;
10)  fi;
11)  i ←1;    q[1] ← 0;
12)  get the first permutation Per[i] in the list of all permutation on[R[i]] that are
      generated in lex order;
13)  Repeat
14)    If the region i of A is the same after applying Per[1] Per[2] …Per[i] on A
15)      Then If  i≠m
16)             Then   i ← i+1;  // Forward step
17)                    q[i] ← 0;
18)        get te first permutation Per[i] on [R[i]] in lex order;
19)                    GOTO 30
20)               Else   ℓ ← ℓ +1;
21)             fi;
22)    fi;
23)    If  q[i] ≠ R[i]!-1  Then
24)     get the successor  permutation of Per[i] on [R[i]] in lex order;
25)          q[i] ← q[i] +1
26)      Else Repeat
27)           i ← i-1;            // Backward   step
28)           Until  (q[i]≠R[i]!-1) OR (i=0);
29)        fi;
30)     Until (i=0);
                m
31)   K ← ∏ (R[i]!);
               i=1
32) Pn ← Pn + ℓ / K ;
End;
```

The advantage of this method is that it does not need any comparisons with previous or next created matrices. So, we need only one matrix in memory at any time, i.e. $O((n+1)^2)$ space. Clearly, the running time of this procedure

$$O(n^2 \prod_{i=1}^{m} (r_i!))$$

depends on the largest value of r_i. If the values of r_i's are quite small (with respect to n), the algorithm can be considered a bounded one. Unfortunately, the whole counting algorithm depends on the number of constructed matrices which is exponential so the problem of counting unlabeled prime is still NP-hard as in the general case.

Appendix.

According to the well-known fact that "*almost all partially ordered sets are prime*", [17]. Likewise total posets, the numbers of prime posets P_n will be rapidly increasing as n increases. Therefore, it is expected that the running time increases more rapidly like the case of counting unlabeled posets on 14 elements, [11]. For this reason we executed the suggested algorithm up to 13 elements only. In the case of calculating P_{13} the author used a single processor (800 MHZ) for about 5 months to generate in average about 1700 unlabeled prime posets per second. Table (1) contains the values of P_n for $n \leq 13$. The result of paper agreed with that introduced in [3]. Moreover, it strengthen that P_{10} in [16] is incorrect (In [16], P_{10} is less than the actual number with 35 posets).

Table (1)

n	1	2	3	4	5	6	7	8	9	10	11	12	13
P_n	1	0	0	1	4	28	234	2585	36326	646405	14528011	412212506	14768822778

References

[1] B. I. Bayoumi, M. H. El-Zahar and S. M. Khamis, (1989), “Asymptotic Enumeration of N-free Partial Orders", Order 6, No. 3, p. 219-232.

[2] B.I.Bayoumi, M.H.El-Zahar and S.M.Khamis, (1994), “Counting 2-Dimensional Posets", Disc Math.,Vol 131,pp. 29-37.

[3] B. I. Bayoumi , M. H. El-Zahar and S. M. Khamis,(1997), "Algorithmic Counting of Types of UPO Graphs and Posets”, Congressus Numerantium 127, Canada, pp.117-122.

[4] C. Chaunier and N. Lygerõs, (1992), “TheNumber of Orders with Thirteen Element”,Order 9, pp. 203-204.

[5] J. C. Culberson and G. J. E. Rawlins, (1991), "New Results from An Algorithm for Counting Posets", Order 7, p.361-374.

[6] M. H. El-Zahar, (1989), "Enumeration of Ordered Sets", in Algorithms and Order (ed. I.Rival), NATO Adv. Sci. Inst. Ser.C: Math. Phys. Sci., p. 327-352, Kluwer Academic Publishers.

[7] Mohamed H. El-Zahar and Soheir M. Khamis,(2000), “Enumeration of Series-Parallel Posets According To Heights”, Journal of the Egyptain Mathematical Society,Vol. 8(1), pp.1-7.

[8] M. Ernw and K. Stege, (1991), "Counting Finite Posets and Topologies," Order 8, p. 247-265.

[9] J. W. Evans, F. Harary, and M. S. Lynn, (1967), "On the Computer Enumeration of Finite Topologies", Commun. ACM 10, p. 295-297.

[10] P. Hanlon, (1982), "Counting interval graphs", Trans. Amer. Math. Soc., 272, p. 383-426

[11] J. Heitzig and J. Reinhold, (2000), “The number of Unlabeled Orders on Fourteen Elements, Order17, pp. 333-341.

[12] S. M. Khamis, (1991), " On Enumerative and algorithmic aspects of partially ordered sets and graphs", Thesis of Ph.D., Dept. of Math. Faculty of science, Ain Shams University, Cairo, Egypt (unpublished).

[13] S. M. Khamis,(2000) “On Numerical Counting of Prime, UPO, and the General Type of Posets According To Heights ", Congressus Numerantium 146, Canada, pp.157-171.

[14] S. M. Khamis,(2002) " Height Counting of Unlabeled Interval and N-Free Posets ", Discearte Math, (submitted).

[15] D. J. Kleitman and B. L. Rothschild, (1975), "Asymptotic Enumeration of Partial Orders", Trans. Am. Math. Soc. 20, p. 205-220. R. H. Mohring, (1984), " Almost all comparability graphs are UPO", Discrete Math. 50, p.63-70.

[16] R. H. Mohring, (1985), "Algorithmic Aspects of Comparability Graphs and Interval Graphs", in Graphs and Order: The Role of Graphs in the Theory of Ordered Sets and Applications (ed.I. Rival), volume 147, p. 41-102. NATO Adv. Study Inst. Ser.C: Math. Phys. Sci., Vol. 147.

[17] R. H. Mohring, (1985), "Computationally Tractable Classes of Order Sets”, I. Rival (ed.) Algorithms and Orders, Kluwer cademic Pulishers, p.105-193.

[18] R. P. Stanley, (1974), "Enumeration of Posets Generated by Disjoint Unions and Ordinal Sums", Proc. Am. Math. Soc.45(2), p. 295-299.