Fundamenta Informaticae 71 (2006) 309–321
IOS Press

Exploiting the Lattice of Ideals Representation of a Poset

Karel De Loof* **and Hans De Meyer**
Department of Applied Mathematics and Computer Science
Ghent University, Krijgslaan 281 S9, B-9000 Gent, Belgium
Karel DeLoof@UGent.be

Bernard De Baets
Department of Applied Mathematics, Biometrics and Process Control,
Ghent University, Coupure links 653, B-9000 Gent, Belgium

Abstract. In this paper, we demonstrate how some simple graph counting operations on the ideal lattice representation of a partially ordered set (poset) P allow for the counting of the number of linear extensions of P, for the random generation of a linear extension of P, for the calculation of the rank probabilities for every $x \in P$, and, finally, for the calculation of the mutual rank probabilities $\mathrm{Prob}(x > y)$ for every $(x, y) \in P^2$.
We show that all linear extensions can be counted and a first random linear extension can be generated in $\mathcal{O}(|\mathcal{I}(P)| \cdot w(P))$ time, while every subsequent random linear extension can be obtained in $\mathcal{O}(|P| \cdot w(P))$ time, where $|\mathcal{I}(P)|$ denotes the number of ideals of the poset P and $w(P)$ the width of the poset P. Furthermore, we show that all rank probability distributions can be computed in $\mathcal{O}(|\mathcal{I}(P)| \cdot w(P))$ time, while the computation of all mutual rank probabilities requires $\mathcal{O}(|\mathcal{I}(P)| \cdot |P| \cdot w(P))$ time, to our knowledge the fastest exact algorithms currently known.
It is well known that each of the four problems described above resides in the class of #P-complete counting problems, the counterpart of the NP-complete class for decision problems. Since recent research has indicated that the ideal lattice representation of a poset can be obtained in constant amortized time, the stated time complexity expressions also cover the time needed to construct the ideal lattice representation itself, clearly favouring the use of our approach over the standard approach consisting of the exhaustive enumeration of all linear extensions.

Keywords: poset, ideal lattice, random linear extension, number of linear extensions, rank probability distribution, mutual rank probabilities

*Address for correspondence: Department of Applied Mathematics and Computer Science, Ghent University, Krijgslaan 281 S9, B-9000 Gent, Belgium

1. Introduction

The ability to sample uniformly from the set of linear extensions $\Omega(P)$ of a given partially ordered set (poset) P has various applications. In the field of multi-criteria decision support, for instance, a linear extension corresponds to a possible ranking of the alternatives and the ability to sample uniformly allows for the estimation of so-called rank probabilities [14, 15]. Computer scientists are mainly interested in the whole set $\Omega(P)$ because of the connections with sorting theory, distributed algorithms and sequence analysis [10, 11].

Determining the number of linear extensions $|\Omega(P)|$ of a given poset is a fundamental problem in order theory. Brightwell and Winkler [1] have shown that determining $|\Omega(P)|$ is a #P-complete problem (as are the other problems addressed in this paper), indicating that the counting problem may be no easier than the generation problem. The notion of #P-completeness was introduced by Valiant [17] and denotes a class of counting problems analogous to the NP-class of decision problems. Recently, a non-polynomial algorithm for counting the linear extensions of a given poset was suggested by Peczarski [11]. This algorithm is based on the recursive partitioning of the poset in connected components, leading to a considerable gain on some classes of posets.

A lot of research has been done on the problem of generating all linear extensions. An algorithm enumerating all linear extensions in constant amortized time is due to Pruesse and Ruskey [13]. An algorithm that has linear time complexity in the size of its output is said to have Constant Amortized Time (CAT) complexity. Clearly, such an algorithm is optimal up to a constant factor. Based upon this algorithm a uniform extension generator could easily be built and (mutual) rank probabilities could be calculated as well. We will show that the algorithms suggested in this paper perform in general much better than this standard approach. However, they do not avoid the possible exponential blowup inherent to their reliance on the ideal lattice representation.

Lerche and Sørensen [9] have suggested an approximation algorithm for the generation of a random linear extension. Their approach roughly consists in selecting a random pair of incomparable elements in the poset and assigning a random mutual rank to the two elements. This procedure is repeated until no incomparable elements are left. The algorithm runs in polynomial time, but does not guarantee that every extension is generated with equal probability, neither does it guarantee any precision. However, algorithms running in polynomial time based on the Markov chain Monte Carlo method are capable of sampling almost uniformly from the set of linear extensions of a poset obeying some specified measure of preciseness. They constitute an example of a Fully Polynomial Randomized Approximation Scheme (FPRAS for short). Bubley and Dyer [3] developed such an FPRAS that is rapidly mixing, which they prove has a running time of $O(n^3 \log n \epsilon^{-1})$, where n is the number of elements in the poset and ϵ is the desired accuracy. Note that being able to sample almost uniformly from the set of linear extensions also opens the door to approximate $|\Omega(P)|$ and to calculate approximate (mutual) rank probabilities. In this paper, however, we are interested in exact results.

2. Preliminaries

A *poset* will be denoted by $(P, \leq_P)$, where P is a set and $\leq_P$ an order relation on P, *i.e.* a reflexive, antisymmetric and transitive relation whose elements $(a, b) \in \leq_P$ are written as $a \leq_P b$. If no distinction between order relations has to be made, the index P in $\leq_P$ will be omitted. Moreover, if the order

relation is clear from the context, the poset will be simply denoted by P. Two elements x and y of P are called *comparable* if $x \leq_P y$ or $x \geq_P y$; otherwise they are called *incomparable*, and we write $x||y$. A *complete order* is a poset in which every two elements are comparable. We say that y *covers* x, denoted as $x \prec_P y$, if $x <_P y$ and there exists no $z \in P$ such that $x <_P z <_P y$. A *chain* of a poset P is a subset of P in which every two elements are comparable. Dually, an *antichain* of a poset P is a subset of P in which every two elements are incomparable. The *width* of a poset P is defined to be the size of the largest antichain of P and is denoted by $w(P)$. By definition $x <_P y$ if $x \leq_P y$ and not $x \geq_P y$. The relation $\prec_P$ is called the covering relation. Furthermore, a poset can be conveniently represented by a so-called *Hasse diagram* where $x <_P y$ if and only if there is a sequence of connected lines upwards from x to y. We call the elements in the Hasse diagram *vertices* and the lines representing the covering relation *edges*. An example of a Hasse diagram is shown in Figure 1.

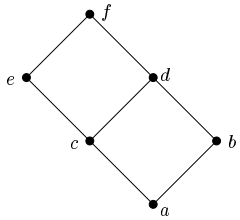

Figure 1. Hasse diagram of a poset P.

A *downset* of a poset $(P, \leq_P)$ is a subset $D \subseteq P$ such that $x \in D$, $y \in P$ and $y \leq_P x$ implies $y \in D$. Dually, an *upset* of a poset $(P, \leq_P)$ is a subset $U \subseteq P$ such that $x \in U$, $y \in P$ and $x \leq_P y$ implies $y \in U$. Downsets are also known as feasible sets, order ideals or *ideals* for short, and upsets as order filters or *filters* for short. Let us further denote the set of ideals of the poset P as $\mathcal{I}(P, \leq_P)$ or as $\mathcal{I}(P)$ for short, and the set of filters as $\mathcal{F}(P, \leq_P)$ or $\mathcal{F}(P)$.

A poset $(Q, \leq_Q)$ is called an *extension* of a poset $(P, \leq_P)$ if for all $x, y \in Q$, $x \leq_P y$ implies $x \leq_Q y$. Then $(P, \leq_P)$ is called a *reduction* of $(Q, \leq_Q)$. A *linear extension* or complete extension is an extension that is a chain. In Figure 2, all linear extensions of the poset in Figure 1 are shown.

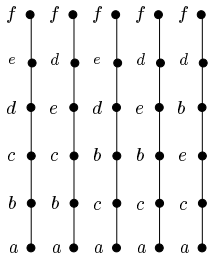

Figure 2. All linear extensions of P.

Let $(P, \leq_P)$ be a poset. If for every $x, y \in P$ $\inf\{x, y\}$ and $\sup\{x, y\}$ exist, $(P, \leq_P)$ is called a *lattice*. We can write $x \wedge y$ (read "x *meet* y") instead of $\inf\{x, y\}$ and $x \vee y$ (read "x *join* y") instead of $\sup\{x, y\}$. A *distributive lattice* is a lattice that satisfies the distributive law: $(\forall a, b, c \in P)(a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c))$. The ideals $\mathcal{I}(P)$ of a poset P equipped with set inclusion as order relation form a

distributive lattice, called the *lattice of ideals* (ideal lattice for short) and denoted by $(\mathcal{I}(P), \subseteq)$. We will implicitly assume that each edge in the Hasse diagram of the ideal lattice is labelled by the element in which the corresponding vertices differ. In Figure 3 the ideal lattice of the poset represented in Figure 1 is shown.

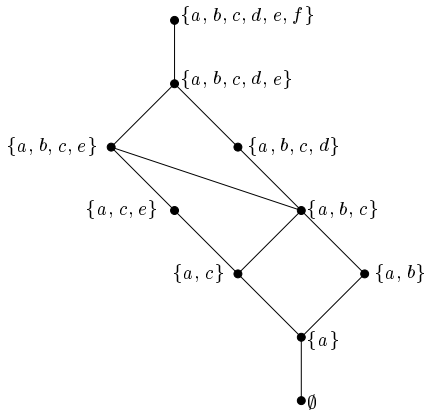

Figure 3. The ideal lattice of the poset P.

The *rank probability* $p_x(i)$ of an element $x \in P$ is defined as the fraction of linear extensions of P where x occurs at position i. The *mutual rank probability* $\mathrm{Prob}(x > y)$ of two elements $(x, y) \in P^2$ is defined as the fraction of linear extensions of P where element x is ranked higher than element y.

3. Random generation of a linear extension

The algorithm presented in this section consists of two independent parts. The first part builds up a data structure allowing for the fast generation of uniformly distributed random linear extensions, producing the number of linear extensions as a side-result. The second part consists of the generation of such a random linear extension.

3.1. Constructing the ideal lattice and counting the number of linear extensions

Since we would like to be able to generate multiple random extensions without major additional cost, it is clear that we should have at our disposal a data structure allowing for the quick selection of successive elements in the linear extension. The ideal lattice is a very adequate choice for this, since it is well known that a linear extension corresponds exactly to a path from the source (corresponding to the empty ideal) to the sink (corresponding to the poset itself). An algorithm for generating this ideal lattice in CAT is due to Habib *et al.* [6] and takes the covering relation of the poset as input. Their algorithm is based on a compact representation of the lattice, called the ideal tree [7]. Basically, a label is assigned to every ideal and an arbitrary linear extension is used to allow for the quick construction of this ideal tree. We refer to this article for an extensive technical discussion on this issue.

Their data structure representing the ideal lattice, denoted as $(\mathcal{I}(P), \subseteq)$, stores for each ideal $I \in \mathcal{I}(P)$:

- $ImSucc(I)$, the list of the immediate successors of I in $(\mathcal{I}(P), \subseteq)$, *i.e.* the ideals that can be reached from I by adding only one element of P. The first immediate successor represents the parent of I in the ideal tree.
- $Child(I)$, the list of children of I in the ideal tree, sorted in decreasing order of their labels.
- $Label(I)$, the label of I.

Remark that with every ideal $I \in \mathcal{I}(P)$ there corresponds a unique filter $F = P \setminus I$. It is well known that there exists a bijection between the ideals and the filters of a poset. We will now extend this data structure by saving for each ideal $I \in \mathcal{I}(P)$:

- $LinExtFilter(I)$, the number of linear extensions of the filter $P \setminus I \subseteq P$.
- $VisitedIdeal(I)$, a boolean flag indicating whether ideal I has already been visited by our algorithm.

Following [6], we denote by U the set of edges in the ideal lattice of P and by V the set of vertices. It should be noted that the storage required for this data structure is only $\mathcal{O}(|U| + |V|)$.

Algorithm `Build`**(Poset** P**)**
Input: The covering relation of P
Output: The ideal lattice $(\mathcal{I}(P), \subseteq)$ including the number of linear extensions for each filter
begin
  build the ideal lattice of P
  $LEcount \leftarrow$ `Assign`$(\emptyset)$
end

Algorithm `Assign`**(Ideal** I**)**
begin
  $VisitedIdeal(I) \leftarrow$ true
  $extensions \leftarrow 0$
  for each ideal I' **in** $ImSucc(I)$ **do**
    if $I' = P$ **then** $extensions \leftarrow extensions + 1$
    else
      if not $VisitedIdeal(I')$ **then** $extensions \leftarrow extensions +$ `Assign`(I')
      else $extensions \leftarrow extensions + LinExtFilter(I')$
  $LinExtFilter(I) \leftarrow extensions$
  `return`$(extensions)$
end

Note that algorithm `Build` invokes algorithm `Assign` with the empty ideal $\emptyset$ as argument, corresponding to the filter $P \setminus \emptyset = P$. The function `return()` terminates recursion and yields its argument as output. After recursion, the variable $LEcount$ contains the number of linear extensions of P. Also, to each ideal in the lattice $(\mathcal{I}(P), \subseteq)$ is associated the number of linear extensions of the complementary filter. This is illustrated in Figure 4 for the poset in Figure 1.

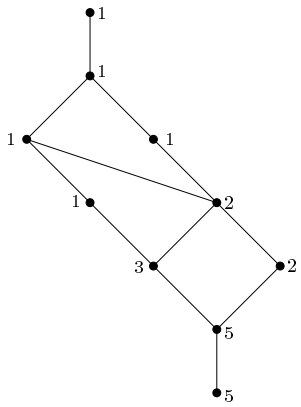

Figure 4. The ideal lattice of P after assigning to each ideal I the number of linear extensions of $P \setminus I$.

Since every edge in the ideal lattice is visited exactly once in the recursive algorithm, it is clear that the time complexity is linear in the size of the ideal lattice, therefore not changing the asymptotic complexity of the construction algorithm of Habib *et al.* [6]. It immediately follows that the time complexity of algorithm `Build` is bounded by $\mathcal{O}(|U| + |V|)$. It is clear that the maximum number of immediate successors in the ideal lattice equals $w(P)$. This observation allows us to state that $\mathcal{O}(|U| + |V|)$ is bounded by $\mathcal{O}(|\mathcal{I}(P)| \cdot w(P))$, thus yielding the time complexity stated in the abstract.

3.2. Generating random linear extensions

As noted before, a linear extension corresponds to a path in the ideal lattice from the smallest ideal, *i.e.* the empty ideal, to the greatest ideal, *i.e.* the poset itself. The data structure constructed in the previous section contains precisely the number of linear extensions in the complementary filter $F = P \setminus I$ for each ideal $I \in \mathcal{I}(P)$. From these observations it immediately follows that the probability that should be assigned to a certain choice, say $x_k \in P$ with $k \leq p$, between possible next elements $\{x_1, \ldots, x_p\}$ equals the ratio of $LinExtFilter(I \cup \{x_k\})$ to $LinExtFilter(I)$. For the poset P in Figure 1, these probabilities are shown in Figure 5.

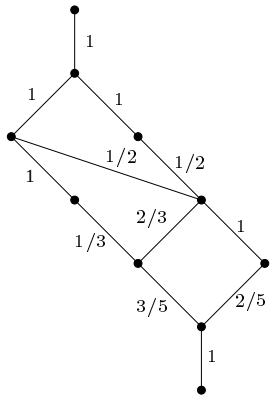

Figure 5. The ideal lattice of P with probabilities assigned to each edge.

The algorithm `Generate` generates one random linear extension of the poset P. Since all ratios $LinExtFilter(I \cup \{x_i\})$ to $LinExtFilter(I)$, for $i = 1, \ldots, p$, have to be calculated for every point in the poset P, the time complexity clearly equals $\mathcal{O}(|P| \cdot w(P))$. The problem is then completely reduced to uniform sampling in a set of p alternatives with given probabilities.

Algorithm `Generate`**(IdealLattice** $\mathcal{L}$**)**
Input: The ideal lattice data structure constructed by the algorithm `Build`
Output: A linear extension of P
begin
  $I \leftarrow \emptyset$
  $E \leftarrow$ ""
  while $I \neq P$ **do**
    $cumulAssignment \leftarrow 0$
    $rand \leftarrow$ random number between 1 and $LinExtFilter(I)$ inclusive
    for each ideal I' **in** $ImSucc(I)$ **do**
      $cumulAssignment \leftarrow cumulAssignment + LinExtFilter(I')$
      if $rand \leq cumulAssignment$ **then**
        E.`add`$(I' \setminus I)$
        $I \leftarrow I'$
        break for loop
  `return`(E)
end

It should be noted that a linear extension is represented in the algorithm as a list and the function `add()` is used to add an element to a given list. The maximal ideal P used is really the poset P itself.

Special care should be taken when implementing the algorithm `Assign`. Since the number of linear extensions of a poset can be huge, a slightly modified version that only saves approximate values for $LinExt(I)$ could be suggested. Instead of saving integer values, a transformation into real values could be introduced when a certain threshold is exceeded. Only the exponent and some digits of the mantissa would then be saved. Since the numbers of extensions added to each other in general have the same order of magnitude, the approximation error will be negligible. Remark that in this case only an approximate result for the number of linear extensions will be obtained and that a slight bias on the distribution will be induced. However, since in the context of the random generation of linear extensions we are interested in ratios, for the above-mentioned reason, we can almost ignore this.

4. Computation of the rank probability distributions

We observe that counting the number of linear extensions of a poset P with a specific element $x \in P$ on position i ($i = 1, 2, ..., |P|$) amounts to the problem of counting the number of paths in the ideal lattice representation containing the edges labelled x at height i, a problem that is easily solved by careful counting. The rank probability $p_x(i)$ then equals the fraction of the total number of paths, *i.e.* the number of linear extensions of P, containing one of the edges labelled x at height i. Note that height in this context is the path length starting from the unique source of the ideal lattice.

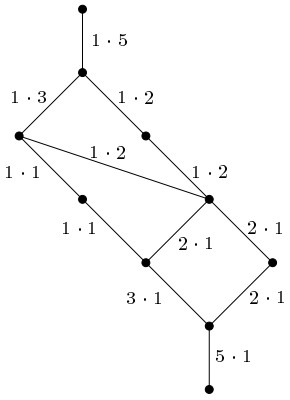

Figure 6. The ideal lattice of P with both path numbers assigned.

In order to calculate for each edge the number of paths containing that edge, we suggest the following strategy: a first pass consists of the Algorithm `Assign` described in the previous section and a second pass of a breadth-first counterpart `AssignBF` counting the number of paths of the ideal lattice bottom-up instead of top-down. Essentially, this means we will extend the data structure representing the ideal lattice with the following item:

- $LinExtIdeal(I)$, the number of linear extensions of the ideal $I \subseteq P$.

The number of paths containing an edge is then simply the product of the numbers $LinExtFilter$ and $LinExtIdeal$ attached to the two ideals connected by that edge as shown in Figure 6. The result of the application of Algorithm `ComputeRankProbs` on the poset in Figure 1 is shown in Figure 7.

Algorithm `ComputeRankProbs`**(Poset** P**)**
Input: The covering relation of P
Output: The rank probability distribution of each element of P
begin
  build the ideal lattice of P
  $LEcount \leftarrow$ `Assign`$(\emptyset)$
  `AssignBF`(P)
  for each element E **in** P **do**
    for each height $Height$ **in** $0, ..., |P|$ **do**
      $RankProb(E, Height) \leftarrow 0$
  `ComputeRankProb`$(\emptyset, 1)$
  `return`$(RankProb)$
end

Algorithm `ComputeRankProb`**(Ideal** I**, Integer** $Height$**)**
begin
  $VisitedIdeal(I) \leftarrow true$
  for each ideal I' **in** $ImSucc(I)$ **do**

$$RankProb(I' \backslash I, Height) \leftarrow RankProb(I' \backslash I, Height) + \frac{LinExtIdeal(I) \cdot LinExtFilter(I')}{LEcount}$$

if $I' \neq P$ **and not** $VisitedIdeal(I')$ **then**
`ComputeRankProb`(I', $Height + 1$)
end

$x \setminus i$	1	2	3	4	5	6
a	1	0	0	0	0	0
b	0	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{5}$	0	0
c	0	$\frac{3}{5}$	$\frac{2}{5}$	0	0	0
d	0	0	0	$\frac{2}{5}$	$\frac{3}{5}$	0
e	0	0	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	0
f	0	0	0	0	0	1

Figure 7. The rank probability distributions $p_x(i)$ for the poset in Figure 1

The above discussion implies that all rank probability distributions can be obtained by this two-pass technique in time linear in the number of edges in the ideal lattice, justifying the complexity bound of $\mathcal{O}(|\mathcal{I}(P)| \cdot w(P))$ stated in the abstract. This is clearly a major gain in time complexity compared to the approach suggested in [2] consisting of exhaustive enumeration of all linear extensions.

5. Computation of the mutual rank probabilities

Applying the technique sketched in the previous section provides us with the number of paths containing each edge in the ideal lattice. It is immediately clear that if we sum all numbers associated to edges labelled y lying under an edge labelled x, we obtain the number of paths, *i.e.* the number of linear extensions, for which it holds that $x > y$. Finally, we only have to divide this number by the total number of paths in order to obtain the mutual rank probability $\mathrm{Prob}(x > y)$. Note that this total number can be derived immediately from the counting information, similarly as in Algorithm `Assign`.

The problem thus reduces to the calculation of the sum of all numbers associated to edges labelled y lying under an edge labelled x. Of course, we insist on obtaining all mutual rank probabilities $\mathrm{Prob}(x > y)$ for all $x, y \in P$ in a single pass, which will constitute a major gain in time complexity compared to the standard approach of enumerating all linear extensions and calculating for every couple $(x, y) \in P$ the number of linear extensions obeying the condition $x > y$.

We will propose an algorithm that gradually builds up a two-dimensional table *MRP* initialized with zeroes, which will, after execution of the algorithm, contain all mutual rank probabilities $\mathrm{Prob}(x > y)$. The main idea is to traverse the ideal lattice recursively in a depth-first manner, using an array of boolean flags $HasVisited$ (one flag for each element in P) indicating which elements have already been visited by the traversal.

Algorithm `ComputeMutualProbs`**(Poset** P**)**
Input: The covering relation of P
Output: The mutual rank probability distribution

begin
  build the ideal lattice of P
  $LEcount \leftarrow$ `Assign`($\emptyset$)
  `AssignBF`(P)
  for each element E **in** P **do**
    for each element F **in** P **do**
      MRP$(E, F) \leftarrow 0$
  for each element $Flag$ **in** P **do**
    $HasVisited(Flag) \leftarrow$ false
  `ComputeMutualProbsDFS`($\emptyset$)
  `return`(*MRP*)
end

Algorithm `ComputeMutualProbsDFS`**(Ideal** I**)**
begin
  $VisitedIdeal(I) \leftarrow$ true
  for each ideal I' **in** $ImSucc(I)$ **do**
    for each element E **in** P **do**
      if $HasVisited(E)$ **then** *MRP*$(I' \setminus I, E) \leftarrow$ *MRP*$(I' \setminus I, E)$
        $+\dfrac{LinExtIdeal(I) \cdot LinExtFilter(I')}{LEcount}$
    if $I' \neq P$ **and not** $VisitedIdeal(I')$ **then**
      $HasVisited(I' \setminus I) \leftarrow$ true
      `ComputeMutualProbsDFS`(I')
      $HasVisited(I' \setminus I) \leftarrow$ false
end

After the depth-first traversal the table *MRP*(x, y) will precisely contain the mutual rank probabilities $\mathrm{Prob}(x > y)$. The result of the application of this algorithm on the poset of Figure 1 is shown in Figure 8. It is easy to see that the required time complexity is bounded by $\mathcal{O}(|\mathcal{I}(P)| \cdot |P| \cdot w(P))$ since for every edge of the ideal lattice a loop from 1 to $|P|$ will be executed.

$x \setminus y$	a	b	c	d	e	f
a	0	0	0	0	0	0
b	1	0	$\frac{3}{5}$	0	$\frac{1}{5}$	0
c	1	$\frac{2}{5}$	0	0	0	0
d	1	1	1	0	$\frac{3}{5}$	0
e	1	$\frac{4}{5}$	1	$\frac{2}{5}$	0	0
f	1	1	1	1	1	0

Figure 8. The mutual rank probabilities $\mathrm{Prob}(x > y)$ for the poset in Figure 1

6. On the relationship between the number of ideals and the number of linear extensions of a poset

For an arbitrary poset, the number of ideals $|\mathcal{I}(P)|$ is in general much lower than the number of linear extensions $|\Omega(P)|$. Table 1 gives some evidence for this statement, justifying the use of the suggested algorithms on the ideal lattice over the exhaustive enumeration of all linear extensions of P as suggested in [2]. The poset elements have been generated by choosing uniformly at random n points (x_1, x_2) out of a two-dimensional real grid of size 20 by 20, equipped with the usual product ordering.

$\lvert P\rvert$	$\lvert\Omega(P)\rvert$	$\lvert\mathcal{I}(P)\rvert$
5	6	9
10	$5.40 \cdot 10^2$	33
15	$2.39 \cdot 10^5$	148
20	$1.13 \cdot 10^9$	518
25	$1.07 \cdot 10^{12}$	953
30	$7.83 \cdot 10^{14}$	1406
35	$5.57 \cdot 10^{17}$	2637

Table 1. The number of ideals compared to the number of linear extensions in a poset.

Not surprisingly, since the problems under consideration are #P-complete, $|\mathcal{I}(P)|$ could still be exponential in the size of the poset. Indeed, consider the extreme case of an antichain poset. It can easily be shown that the number of ideals in this case is $|\Omega(P)| = 2^n$ for a poset P consisting of n elements. Therefore, the Algorithm `Build` could have a time complexity of $\mathcal{O}(n \cdot 2^n)$. However, to date known algorithms for an arbitrary poset have $\mathcal{O}(e(P))$ or $\mathcal{O}(|P| \cdot e(P))$ time complexity, where $e(P)$ is the number of linear extensions of P, which could be $n!$. The presented algorithms succeed in replacing the factor $n!$ with a factor $n \cdot 2^n$ in the worst case. However, there is a drawback in the sense that all suggested algorithms can require exponential memory complexity. Nevertheless, in practice there is a large class of thin posets with a connected Hasse diagram that have a reasonable size of the ideal lattice as for example shown in 1, for which the algorithms imply a major gain in time.

Fast algorithms enumerating all ideals of a poset are known. However, the problem of finding a so-called loopless algorithm generating all ideals in CAT is still an open question. The fastest currently known algorithm enumerating all ideals of a poset is due to Squire [16] and generates all ideals in $\mathcal{O}(\log n)$ amortized time. For certain classes of posets, such as series-parallel posets or forest posets, better algorithms are available ([8], [12]). Since the number of ideals of a poset can be exponential in the size of the poset, it is not surprising that the problem of counting all ideals of a poset is also #P-complete [5]. The problem of determining $|\mathcal{I}(P)|$ even resides in the subclass #RHΠ_1 of #P, which can be characterized as being the hardest subclass of #P to approximate.

7. Conclusion

We have established simple algorithms for generating a random linear extension and to derive (mutual) rank probabilities of a poset. The first algorithm consists of two essential parts: the computation of probabilities associated to the edges in the ideal lattice of the given poset and the generation of a linear extension. The first part is the most time consuming one but should only be executed once, independent of the number of linear extensions of the same poset required. The second algorithm provides us immediately with all (mutual) rank probabilities based on the ideal lattice representation and similar techniques as in the first algorithm, thereby improving the standard approach of exhaustive enumeration of all linear extensions.

Acknowledgements

The authors wish to thank the anonymous referee for the many helpful comments and suggestions on an earlier version of this manuscript.

References

[1] G. Brightwell and P. Winkler. Counting linear extensions. *Order*, 8(3):225–242, 1991.

[2] R. Brüggemann, D. Lerche, and P.B. Sørensen. First attempts to relate structures of Hasse diagrams with mutual probabilities. Technical Report 479, The 5th workshop held at the National Environmental Research Institute (NERI), Roskilde, Denmark, 2003.

[3] R. Bubley and M. Dyer. Faster random generation of linear extensions. *Discrete Mathematics*, 20:81–88, 1999.

[4] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, 2003.

[5] M. Dyer, L.A. Goldberg, C. Greenhill, and M. Jerrum. On the relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2003.

[6] M. Habib, R. Medina, L. Nourine, and G. Steiner. Efficient algorithms on distributive lattices. *Discrete Applied Mathematics*, 110:169–187, 2001.

[7] M. Habib and L. Nourine. Tree structure for distributive lattices and its applications. *Theoretical Computer Science*, 165:391–405, 1996.

[8] Y. Koda and F. Ruskey. A gray code for the ideals of a forest poset. *Journal of Algorithms*, 15(2):324–340, 1993.

[9] D. Lerche and P.B. Sørensen. Evaluation of the ranking probabilities for partial orders based on random linear extensions. *Chemosphere*, 53:981–992, 2004.

[10] H. Mannila and C. Meek. Global partial orders from sequential data. Proceedings of the Sixth ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining, Boston, MA, USA, 161–168, 2000.

[11] M. Peczarski. New results in minimum-comparison sorting. *Algorithmica*, 40:133–145, 2004.

[12] G. Pruesse and F. Ruskey. Gray codes from antimatroids. *Order*, 10:239–252, 1993.

[13] G. Pruesse and F. Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23:373–386, 1994.

[14] U. Simon, R. Brüggeman, and S. Pudenz. Aspects of decision support in water management - example Berlin and Potsdam (Germany) I - spatially differentiated evaluation. *Water Research*, 38:1809–1816, 2004.

[15] U. Simon, R. Brüggeman, and S. Pudenz. Aspects of decision support in water management - example Berlin and Potsdam (Germany) II - improvement of management strategies. *Water Research*, 38:4085–4092, 2004.

[16] M.B. Squire. Enumerating the ideals of a poset, North Carolina State University, 1995. http://citeseer.ist.psu.edu/465417.html.

[17] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.