RANDOM POSETS, LATTICES, AND LATTICES TERMS

Jaroslav Ježek, Václav Slavík, Praha

(Received January 6, 1998)

Abstract. Algorithms for generating random posets, random lattices and random lattice terms are given.

Keywords: lattice, poset, algorithm

MSC 2000: 06B25, 06-04

An elaboration of a conjecture concerning finite lattices often depends, in its initial phase, on the verification for a set of randomly chosen lattices.

In this paper we are going to present three algorithms: for generating a random poset, or random lattice, with a given number of elements, and for generating a random lattice term.

The algorithm for a random lattice can be also used for generating a random join semilattice: a random join semilattice with N elements is nothing else than a random lattice with $N+1$ elements, from which we remove the least element.

We suppose that a (good) random number generator is given. For a positive integer i, `rnd(i)` is a random number from $\{0,\dots,i-1\}$.

For the notation the reader is referred to either [2] or [3].

The algorithm in Section 3 is based on ideas of J.-B. Nation.

While working on this paper both authors were partially supported by the Grant Agency of the Czech Republic, Grant No 201/96/0312. The first author was also partially supported by Grant Agency of the Academy of Sciences of the Czech Republic, Grant No A1019508.

1. Posets

Denote by N the number of elements of a random poset. Let L be a two-dimensional array of size $N \times N$, which will hold the less-or-equal relation table of the poset. We initialize L by setting $L[i][j] = 0$ for $i \neq j$ and $L[i][i] = 1$ ($i, j = 0, \ldots, N-1$).

We will also need two (one-dimensional) arrays M and Q of size N.

The random poset will be given by its table L after executing the function `Work(i)` for $i = 0, \ldots, N-1$. This function calls the auxiliary function `FindMax(i)`, which finds the maximal elements of the current poset, chooses its random subset and returns the number of elements of this subset.

For a positive integer i, denote by $S(i)$ the least positive integer j such that $j^2 \geqslant i$ and $j \geqslant 2$.

```
int FindMax(i){ int k=0; int j,s,a;
for(j=0;j<i;j++){
  s=1; for(a=0;a<i;a++) if(a!=j&& L[j][a]) s=0;
  if(s) {M[k]=j;k++;} }
a=rnd(k+1); for(j=0;j<k;j++) Q[j]=0;
for(s=0;s<a;s++){j=rnd(k); if(Q[j])s--; else Q[j]=1;}
return k;}

void Work(i){ int j,l,w,s,q,u;
q=S(N-i);
if(i==0) u=0; else if(!rnd(q)) u=FindMax(i);
for(j=0;j<u;j++) if(Q[j]) L[M[j]][i]=1;
w=1; while(w){w=0;
for(j=0;j<i;j++) if(L[j][i]) for(s=0;s<i;s++)
  if(L[s][j]&&!L[s][i]){ w=1; L[s][i]=1;}}}
```

2. Lattices

The idea of generating a random lattice is similar to that of a random poset, but a little more complicated.

Again, the number of elements will be denoted by N. Instead of the less-or-equal relation, we need the join table, which will be held in a two-dimensional array J of size $N \times N$. The table is initialized by setting $J[i][i] = i$ and $J[i][j] = -1$ for $i \neq j$ (meaning that the joins are not yet defined).

The lattice is generated from below. Assume that its order ideal of k elements has been constructed. From the set of maximal elements of the order ideal we select a random subset S (if $k = N - 1$, S must be the set of all maximal elements). We then add a new element a, covering all the elements of S. (This may force some maximal elements outside S to be also covered by a.) For i, j with $i < a$, $j < a$ and $J[i][j]$ not yet defined, we set $J[i][j] = a$.

The `FindMax(i)` function is almost the same as for posets. The `Work(i)` function is different.

```
int FindMax(i){ int k=0; int j,s,a;
for(j=0;j<i;j++){
  s=1; for(a=0;a<i;a++) if(a!=j&&J[a][j]==a) s=0;
  if(s){M[k]=j; k++;}}
a=rnd(k);a++; for(j=0;j<k;j++)Q[j]=0;
for(s=0;s<a;s++){ j=rnd(k);if(Q[j])s--;else Q[j]=1;}
return k;}

void Work(i){ int j,l,w,s,q,u;
if(i==N-1){for(j=0;j<N;j++) for(l=0;l<N;l++)
  if(J[j][l]==-1) J[j][l]=N-1; return;}
q=S(N-i);
if(i==1){u=1; M[0]=0; Q[0]=1;}
else if(!rnd(q)) u=FindMax(i);
for(j=0;j<u;j++) if(Q[j]){J[M[j]][i]=i; J[i][M[j]]=i;}
w=1; while(w){w=0;
for(j=0;j<i;j++)if(J[j][i]==i)for(s=0;s<i;s++)
  if(J[s][j]==j&&J[s][i]!=i){w=1;J[s][i]=i;J[i][s]=i;}
for(j=0;j<i;j++)if(J[j][i]==i)for(l=0;l<i;l++)if(J[l][i]==i){
  s=J[j][l];if(s!=-1&&J[s][i]!=i){w=1;J[s][i]=i;J[i][s]=i;}}}
for(j=0;j<i;j++)if(J[j][i]==i)for(l=0;l<i;l++)
  if(J[l][i]==i&&J[j][l]==-1){J[j][l]=i;J[l][j]=i;}}
```

The idea of generating a random lattice term (which should be given in its canonical form) in n variables $x_1, \ldots, x_n$ is the following. We first generate a random lattice with a set of n generators $g_1, \ldots, g_n$. Then we seek for an element g standing as far from the generators as possible, and obtain a term $t(x_1, \ldots, x_n)$ with $g = t(g_1, \ldots, g_n)$ as a result.

The previously described algorithm for producing a random lattice cannot be used for this purpose, since it does not allow any control over the generators of the lattice. However, one can see easily that it is sufficient for the present purpose to generate a random bounded (in the sense of, e.g., [1] and [2]) lattice instead of a random general lattice. As is well known (and proved in A. Day [1]), finite bounded lattices are precisely those lattices that can be obtained from the one-element lattice in finitely many steps by doubling the intervals. So, it is easy to generate an infinite random sequence of finite bounded lattices $L_0, L_1, \ldots$ of increasing sizes: L_0 is the one-element lattice, and L_{i+1} is obtained from L_i by doubling its random interval.

One can set $g_1 = \ldots = g_n = 0$ in L_0, and if the lattice L_i is generated by n elements, again denoted by $g_1, \ldots, g_n$, one can restrict the random selection of an interval in L_i in such a way that the lattice L_{i+1}, resulting by doubling this interval, is again n-generated, and its n generators $g_1, \ldots, g_n$ can be obtained from those of L_i, taking only one appropriate element each time when a generator has been doubled. We will not give the details of the algorithm here, since it is rather technically complicated but the idea is simple.

Since the cardinalities satisfy $|L_i| < |L_{i+1}| \leqslant 2|L_i|$ for all i, one can find in the sequence a random bounded lattice L with $N \leqslant |L| < 2N$ for any given N. Let J and M be two two-dimensional arrays of sizes $2N \times 2N$, holding the join and meet tables of such a random bounded lattice. We will suppose, for example, that $n = 3$ (the number of generators of the lattice.) The three generators of L will be denoted by g_1, g_2, g_3 (so that $0 \leqslant g_1, g_2, g_3 < 2N$ with respect to encoding lattice elements by nonnegative integers). The function `ProduceTerm()`, listed below, produces a random term in three variables x, y, z based on this lattice. The function `wr(i)` is auxiliary; it serves to print the term. We also need four auxiliary arrays A, B, C, D of sizes $2N$.

```
void wr(i){
if(i==0) printf(``x'');
else if(i==1) printf(``y'');
else if(i==2) printf(``z'');
else{if(B[i]>2)printf(``('');  wr(B[i]);  if(B[i]>2) printf(``)'');
```

```
    if(D[i]==1) printf(``.''); else printf(``+'');
    if(C[i]>2)printf(``('');wr(C[i]);if(C[i]>2)printf(``)'');}}

void ProduceTerm(){  int i,j,k,l,c,d,u,m,p;
A[0]=g1; A[1]=g2; A[2]=g3; p=1;
k=3; while(p){m=k; p=0;
  for(i=0;i<m;i++) for(j=0;j<m;j++){
    u=J[A[i]][A[j]];
    c=0; for(l=0;l<k;l++) if(u==A[l]) c=1;
    if(!c){p=1; A[k]=u; B[k]=i; C[k]=j; D[k]=2; k++;}}
  for(i=0;i<m;i++) for(j=0;j<m;j++){
    u=M[A[i]][A[j]];
    d=0; for(l=0;l<k;l++) if(u==A[l]) d=1;
    if(!d){p=1; A[k]=u; B[k]=i; C[k]=j; D[k]=1; k++;}}}
wr(k-1);}
```

The random term obtained in this way is given in its canonical form.

References

[1] *A. Day*: Splitting lattices generate all lattices. Algebra Universalis *7* (1977), 163–170.
[2] *R. Freese, J. Ježek, J. B. Nation*: Free Lattices. Mathematical Surveys and Monographs 42, American Mathematical Society, Providence, RI, 1995.
[3] *G. Grätzer*: General Lattice Theory. Academic Press, New York, 1978.

Authors' addresses: *Jaroslav Ježek*, MFF UK, Sokolovská 83, 186 75 Praha 8, Czech Republic, e-mail: `jezek@karlin.mff.cuni.cz`; *Václav Slavík*, Czech Agricultural University, Kamýcká 129, 165 21 Praha 6, Czech Republic, e-mail: `slavik@tf.czu.cz`.