SRW computation

advanced version

Implementation of the computer-assisted proof of the paper "Generalized approach to the non-backtracking lace expansion" by Robert Fitzner and Remco van der Hofstad

Robert Fitzner
Eindhoven University of Technology

Abstract

This document is the first part of the computer-assisted proof of the non-backtracking lace expansion (NoBLE). The NoBLE was created by Remco van der Hofstad and the author to prove mean-field behavior for self-avoiding walk, lattice tree (LT), lattice animals (LA) and percolation. In this file the model-independent computations, explained in "Generalized approach to the non-backtracking lace expansion" Section 5, are computed. All reference in this file are to this paper.

We compute the Green's function of simple random walk (SRW) and other related quantities. The computation of SRW-integrals are based upon techniques of Takashi Hara and Gordon Slade explained in "The lace expansion for self-avoiding walk in five or more dimensions." from (1991). Additionally, we store the number of self-avoiding and bond-avoiding walks.

This file is the advanced version of the SRW-integral computation. We automate many steps using functions, called methods in *Mathematica*. This allows us to compute the SRW-integrals for arbitray sets of points x. We refer readers who are less familiar with programming to the basic version of the file.

Be aware: Running the file might take up to an hour as we compute a number of basic SRW-integral using a series expansion in the Subsection Computation of $I_{n,t}(x)$. During the time-consuming computations in Section 1.3 the program creates an output indicating the progress of the computation. The result of these time-consuming computations are saved into two separate notebooks that will be created on your computer. These are loaded automatically, when you run the program a second time.

25.10.2016

What to do with this file?

You use this file, together with the model-dependent file, to perform the computer-assisted part of the NoBLE. When successful, this proves the infrared bound for a given model in a given dimension.

In this advanced version, many of the functions have been automated to allow for more flexibility in the bounding of the NoBLE coefficients. Further, we have implemented the computation of the SRW-integral manually to ensure precession in dimensions closer, but still above, the upper critical dimension. This was not possible with the built-in function of *Mathematica*.

Please add the input in the next section (in particular, this fixes the dimension in which you perform the NoBLE) and then evaluate the whole notebook. Due to the manual computation of the SRW-Integrals this could take up to an hour. However, once computed, you will only need to return to this file if you want to change the dimensions or the set of specific endpoints of the diagram.

Input

The dimension in which we perform the computations

```
d = 11;
param = 4;(* for 2 we only compute lines and bubbles,
for 3 also triangles (used for SAW), for 4 also squares (used for percolation),
for 5 also pentagons (used for LTLA) *)
MaxNumberOfSteps = 20; (* Please,
choose this to be even. *)
```

We compute the SRW-bubble, SRW-triangle, SRW-square and SRW-pentagon. The SRW-pentagon is only finite for dimensions $d \geq 11$, while the square is finite for $d \geq 9$. To be able to treat percolation in $d=10,11,$ we introduce the parameter **param**.

In the basic version of the file we rely on the computation of SRW-integrals for $x = 0,\ e_1,\ 2\,e_1,\ e_1 + e_2$. This means that we considered three classes of simple diagrams: closed diagrams, diagrams that end at a neighbor of the origin and arbitray open diagrams. We bound the last class using the bounds for $x = 2\,e_1,\ e_1 + e_2$ and monotonicity arguments in x. Thus, it is sufficient to compute $L_n(x)$ only for those values. This requires the computation of $I_{n,l}(x)$ only for eight distinct values.

In this advanced version, we give the possiblity to enter the points x for which we compute the functions $L_n(x)$. We fix the choice of classes of simple diagrams: closed, ending at $x = e_1,\ 2\,e_1,\ e_1 + e_2$ and, to bound the bigger diagrams, we also include bounds on $x = e_1 + e_2 + e_3,\ 2\,e_1 + e_2,\ 3\,e_1$.

The points for which you consider the simple diagrams are entered in the following input field:

```
MainNodes = {{0}, {1}, {2}, {0, 1}, {3}, {1, 1}, {0, 0, 1}};
(* Points in usual coordinates (in the order of occurrence) x=0,e1,e1+e2,2e1*)
```

Once you have selected these inputs compile/evaluate the whole notebook. This might take up to an hour.

Having entered the *MainNodes,* the other points required to compute $L_n(x)$ are computed automatically.

For the reader of this file

This file constist of three sections:

- In Section 1, we compute the SRW integrals $I_{n,l}$, corresponding to SRW Green's functions and their n-fold convolutions and l-fold convolutions with D.
- In Section 2, we compute bounds other SRW-integrals, using the values of $I_{n,l}$.
- In Section 3, we store the number of self-avoiding and bond-avoiding walks.

Next, we highlight a feature of our implementation. Namely, the coordinates with which we identify a point $x \in Z^d$. We map the standard representation of a vector $x = (x_1, x_2, \ldots, x_d)$ to

$$x \mid --> v(x) = (N_1(x),\ N_2(x),\ N_3(x),\ N_4(x),\ \ldots.$$

where $N_j(x)$ = number of x_i with $|x| = j$. For example, we represent x as follows

$$x = e_1 \mid --> v(x) = \{1,\ 0,\ 0,\ 0,\ \ldots ..\}$$
$$x = 4\,e_1 \mid --> v(x) = \{0,\ 0,\ 0,\ 1,\ \ldots ..\}$$
$$x = -4\,e_5 \mid --> v(x) = \{0,\ 0,\ 0,\ 1,\ \ldots ..\}$$
$$x = e_1 + e_2 + 2\,e_2 + 3_4 \mid --> v(x) = \{2,\ 1,\ 1,\ 0,\ \ldots ..\}$$

This mapping is not a bijection. However, it incodes all the necessary information as the considered SRW quantities have various symmetries, e.g. $I_{n,l}(2\,e_1 + e_2) = I_{n,l}(e_3 - 2\,e_5)$. Thus, this description via $v(x)$ carries all the information required to compute the value of a SRW-integral.

We are only interested in x that are close to the origin, and we end our representation $v(x)$ after the last non-trivial entry, i.e. $v(x)=\{0,1,1,0,.....\}=\{0,1,1\}$.

1. Simple Random Walk integral $I_{n,l}(x)$

We compute the two-point function of the simple random walk,

$$I_{n,l}(x) = \int_{[-\pi,\pi]} e^{\mathrm{i}k\,x}\,\frac{\hat{D}^l(k)}{\left(1-\hat{D}(k)\right)^n}\,\frac{d^d k}{(2\pi)^d}.$$

In the advanced version, we save the computed values into the file SRWData.nb. Then, the program first checks whether we have already computed the SRW-integrals for these parameters. When this is not the case, we compute the values using the following relation:

$$I_{n,l}(x) = I_{n,(l-1)}(x) - I_{(n-1),(l-1)}(x).$$

This reduces the computation of $I_{n,l}(x)$ to the computation of $I_{0,l}(x)$ and $I_{n,0}(x)$. We split this section into four parts:

- the computation of the additional points required to compute L_n and $J_{n,l}$ for all points *MainNodes* given in the input
- the computation of $I_{0,l}(x)$
- the computation of $I_{n,0}(x)$
- use the relation above to compute $I_{n,l}(x)$ for other values of *n,l.*

1.0 Loading existing data

We save the data in the following directory:

```
$InitialDirectory
```

```
C:\Users\rfitzner\Documents
```

If you want, then you can of course change the working directory here, but in the version we distribute we leave it as it is. We next check whether you have saved the necessary data before:

```
SRWCountFile = "SRWCountData.nb";
SRWIntegralFile = "SRWIntegralsData.nb";
FileExistsQ[SRWCountFile]
FileExistsQ[SRWIntegralFile]
```

```
True
```

```
True
```

If not, then we create the file for future use by running

```
If [(! FileExistsQ[SRWCountFile]),
  Dummy = 1;
  Save[SRWCountFile, Dummy],
  Get[SRWCountFile]];
If [(! FileExistsQ[SRWIntegralFile]),
  Dummy = 1;
  Save[SRWIntegralFile, Dummy],
  Get[SRWIntegralFile]];
Clear[Dummy];
```

1.1 Computation additional necessary points

We compute $L_n(x)$ by using formula (5.17) and $J_{n,l}(x)$ in (3.30). We wrote a program to iterate all possible permutations of x in an efficient

manner. Below we also implement the method that we use to compute L_n.

Central methods to compute the necessary points for L_n

```
(*The method that computes Ln. We create an abstract representation of the point
  and then use this representation to go through all possible permutations of x*)
ComputeNeccaryPointForLn[xNSpaceOriginal_, d_] :=
  Module[{listout, level, length, originalNPosition, Commuteded, H, HO, count, p, t, s, x},
    listout = {};
    If[Total[xNSpaceOriginal] == 0,
      listout = {{0}};,
      (*======== created shorted vector=======*)
      length = 0;
      Do[If[xNSpaceOriginal[[s]] > 0, length++;], {s, 1, Length[xNSpaceOriginal]}];
      x = Table[0, {s, 1, length + 1}];
      originalNPosition = Table[0, {s, 1, length}];
      count = 1;
      Do[If[xNSpaceOriginal[[s]] > 0, x[[count]] = xNSpaceOriginal[[s]];
          originalNPosition[[count]] = s;
          count++;], {s, 1, Length[xNSpaceOriginal]}];
      x[[length + 1]] = d - Sum[x[[s]], {s, 1, length}];
      (*======== INITIALISE VARIABLES=======*)
      (* the overlap: -1 represent not initalised parts*)
      H = Table[{-1}, {s, 1, length}];
      (* H is used to create the permutation. The entries represent the number of
        positions v_j(x) that we have not yet covered by the permutation
        (that we compute right now).*)
      Commuteded = Table[0, {s, 1, length}, {t, 1, length + 1}];
      Commuteded[[1]] = x;
      level = 1;
      (*======== Iteration=======*)
      While[ ((level > 0)),
        (*======== Iterate this level=======*)
        H[[level]] = IterateLine[Commuteded[[level]], H[[level]], x[[level]]];
        If[level < length,
          If[H[[level]] ≠ {-1},
              (*======== Integrate interation into table and continue descent=======*)
              Commuteded[[level + 1]] = x - Sum[H[[p]], {p, 1, level}];
              If[Commuteded[[level, length + 1]] ≥ 0, level++;];
              (*======== Integrate at this level ended so we go a level up=======*)
              , level--;];,
          (*reached maximal level*)
          If[H[[level]] ≠ {-1},
              (*======== reached a finite level =======*)
```

```
            If[(x[[length + 1]] − Sum[H[[p, length + 1]], {p, 1, level}] ≥ 0),
              (*======== reached a valid permutation =======*)

              HO = Table[x[[s]] − Sum[H[[t, s]], {t, 1, length}], {s, 1, length + 1}];
              listout = Union[listout, ComputeContributingPoints[x, Join[H, {HO}],
                  d, originalNPosition]];
              (*Create all y, corresponding to the permutation represented by H,
              and add it to the exists list*)
            ];,
            level−−;];
        ];
      ];
    ];
    listout
  ];

ComputeContributingPoints[vectorx_, H_, dim_, originalNPosition_] :=
  Module[{listout, value, PositionofZero, PriorCountOfIndecies, yXspace, xXspace,
    Deltas, newIndex, CountOfIndecies, NspaceRes, s, a, b, i, j, y},
   listout = {};
   (*=======Reduce the computation of +− To a minimum*)
   (*=======we compute the number of position that are not affected by +−*)
   value = Table[0, {r, 1, Length[originalNPosition]}];
   PositionofZero = Max[originalNPosition * 2 + 1];
   PriorCountOfIndecies = Table[0, {s, 1, PositionofZero}];

   yXspace = {};
   xXspace = {};
   Do[
    Do[Do[
        xXspace = Join[xXspace, {originalNPosition[[b]]}];
        yXspace = Join[yXspace, {originalNPosition[[a]]}];
        , {s, 1, H[[a, b]]}];, {b, 1, Length[H[[a]]] − 1}];

    PriorCountOfIndecies[[originalNPosition[[a]]]] += H[[a,  Length[H[[a]] ]  ]];
    , {a, 1, Length[H] − 1}];

   Do[
    PriorCountOfIndecies[[originalNPosition[[b]]]] += H[[  Length[H]  , b ]];
    , {b, 1, Length[H] − 1}];

   PriorCountOfIndecies[[PositionofZero]] = H[[  Length[H]  , Length[H] ]];
```

```
    (*==create the sequence of alternating signs=*)

    Deltas = Tuples[{-1, 1}, Length[xXspace]];
    y = yXspace;
    Do[
      CountOfIndecies = PriorCountOfIndecies;
      Do[
        y[[j]] = Deltas[[i, j]] * yXspace[[j]];
        newIndex = Abs[xXspace[[j]] + y[[j]]];
        If[newIndex > 0,
                CountOfIndecies[[newIndex]]++;
                CountOfIndecies[[PositionofZero]]++;
        ];
        , {j, 1, Length[y]}];
      NspaceRes =
        ResizeVector[Table[CountOfIndecies[[s]], {s, 1, Length[CountOfIndecies] - 1}]];
      listout = Union[listout, {NspaceRes}];
      , {i, 1, Length[Deltas]}];
    listout
  ];
```

Central methods to compute L_n

```
ComputeLNIntegral[xNSpaceOriginal_, d_, ns_] :=
  Module[{value, originalNPosition, count, length, H, Commuteded, level, HO,
    factor, contriutions, r, s, p, x},
   value = Table[0, {r, 1, Length[ns]}];
   If[Total[xNSpaceOriginal] == 0,
    value = Table[Ivalue[ns[[r]], 0, {0}], {r, 1, Length[ns]}] ;,
    (*======== created shorted vector=======*)
    length = 0;
    Do[If[xNSpaceOriginal[[s]] > 0,
        length++;], {s, 1, Length[xNSpaceOriginal]}];

    x = Table[0, {s, 1, length + 1}];
    originalNPosition = Table[0, {s, 1, length}];
    count = 1;
    Do[If[xNSpaceOriginal[[s]] > 0, x[[count]] = xNSpaceOriginal[[s]];
        originalNPosition[[count]] = s;
        count++;], {s, 1, Length[xNSpaceOriginal]}];
    x[[length + 1]] = d - Sum[x[[s]], {s, 1, length}];
    (*======== INITIALISE VARIABLES=======*)
    H = Table[{-1}, {s, 1, length}];
```

```
Commuteded = Table[0, {s, 1, length}, {t, 1, length + 1}];
Commuteded[[1]] = x;
level = 1;
(*======== Iteration=======*)
While[ ((level > 0)),
  (*======== Iterate this level=======*)
  H[[level]] = IterateLine[Commuteded[[level]], H[[level]], x[[level]]];
  If[level < length,
    If[H[[level]] ≠ {-1},
      (*======== Integrate interation into table and continue descent=======*)
      Commuteded[[level + 1]] = x - Sum[H[[p]], {p, 1, level}];
      If[Commuteded[[level, length + 1]] ≥ 0, level++;];
      (*======== Integrate at this level ended so we go a level up=======*)
      , level--;];,
    (*reached maximal level*)
    If[H[[level]] ≠ {-1},
      (*======== reached a finite level =======*)
      If[(x[[length + 1]] - Sum[H[[p, length + 1]], {p, 1, level}] ≥ 0),
        (*======== reached a valid permutation =======*)

        HO = Table[x[[s]] - Sum[H[[t, s]], {t, 1, length}], {s, 1, length + 1}];
        factor = 1/d! Product[(x[[s]]!)^2, {s, 1, length + 1}] /
            Product[(H[[s, t]]!), {t, 1, length + 1}, {s, 1, length}] *
          1 / Product[HO[[s]]!, {s, 1, length + 1}];
        (* factor = number of permutations created this H*)
        contriutions = ComputeIntegralContribution[x, Join[H, {HO}],
          d, factor, originalNPosition, ns];
        Do[
          value[[r]] += contriutions[[r]];
          , {r, 1, Length[ns]}];
        ];,
      level--;];
    ];
  ];
  value
];

ComputeIntegralContribution[vectorx_, H_, dim_, factor_, originalNPosition_, ns_] :=
```

```
Module[{value, PositionofZero, PriorCountOfIndecies, xXspace, yXspace, Deltas,
   newIndex, CountOfIndecies, NspaceRes, a, b, y, s, r, j},
 (*=======we compute the number of position that are not affected by +-*)
 value = Table[0, {r, 1, Length[ns]}];
 PositionofZero = Max[originalNPosition * 2 + 1];
 PriorCountOfIndecies = Table[0, {s, 1, PositionofZero}];
 (* Then, we create the vectors to perform the sum over all signs δ ϵ {-1,1}^d.*)
 yXspace = {};
 xXspace = {};
 Do[
   Do[
     Do[
         xXspace = Join[xXspace, {originalNPosition[[b]]}];
         yXspace = Join[yXspace, {originalNPosition[[a]]}];
         , {s, 1, H[[a, b]]}];
     , {b, 1, Length[H[[a]]] - 1}];
   PriorCountOfIndecies[[originalNPosition[[a]]]] += H[[a,   Length[H[[a]] ]    ]];
   , {a, 1, Length[H] - 1}];
 Do[
   PriorCountOfIndecies[[originalNPosition[[b]]]] += H[[  Length[H]   , b  ]];
   , {b, 1, Length[H] - 1}];

 PriorCountOfIndecies[[PositionofZero]] = H[[  Length[H]   , Length[H]  ]];
 (*==create the sequence of alternating signs=*)
 Deltas = Tuples[{-1, 1}, Length[xXspace]];
 y = yXspace;
 Do[
   CountOfIndecies = PriorCountOfIndecies;
   Do[
     y[[j]] = Deltas[[i, j]] * yXspace[[j]];
     newIndex = Abs[xXspace[[j]] + y[[j]]];
     If[newIndex > 0,
               CountOfIndecies[[newIndex]]++;
               CountOfIndecies[[PositionofZero]]++;
     ];
     , {j, 1, Length[y]}];
   NspaceRes = Table[CountOfIndecies[[s]], {s, 1, Length[CountOfIndecies] - 1}];
   Do[
     value[[r]] += 1/2^Length[y] * factor * Ivalue[ns[[r]], 0, ResizeVector[NspaceRes]];
     , {r, 1, Length[ns]}];
```

```
    , {i, 1, Length[Deltas]}];
  value
];
```

Tools for the main methods

```
(* Tool used to iterate through the abstract represenation of possible points*)
IterateLine[x_, v_, j_] :=
  Module[{list, length, p, searches, s},
    If[v == {-1},
      list = Table[0, {s, 1, Length[x]}];
      list[[Length[list]]] = j;, (* Initiation of this level*)
      length = Length[x] - 1;
      If[Sum[v[[s]], {s, 1, length}] < j, (* can this the line once more?, j=n_s*)
        list = v;
        p = 1;
        While[(p < length ) && (x[[p]] ≤  v[[p]]), p++];
        (* find the first point where we can increase, 989==>999*)
        If[v[[p]] <  x[[p]], (* found point*)
          list[[p]]++;
          Do[list[[s]] = 0, {s, 1, p - 1}];
          list[[length + 1]] = j - Sum[list[[s]], {s, 1, length}];
          , (* all contribution we already given so we have to cancel*)
          Print["cancel unexpected increase"];
          list  = {-1};
        ];
        , (* can not increase an individual entry, so we need to iterate, e.g. 999==>0001*)
        p = 1;
        searches = True;
        While[searches, (* find the first point that are nontrivial, 009887 find 9*)
          If [(p <  length ),
              If[v[[p]] ==  0, p++;, searches = False;];,
              searches = False;
            ]; ];

        If[p == length,
          list  = {-1};, (* We can not continue this line anymore*)
          s = p + 1;
          searches = True;
          (* find the first point that we can increase, 00987 find 8*)
          While[searches,
            If[s <  length && (x[[s]] ≤  v[[s]]), s++;,
```

```
                searches = False;];
            ] ;

            If[(x[[s]] > v[[s]]), (* a check whether we found a position to increase or
                simple reached the end of the line*)
              list = v;  (* we can increase, so we do it *)
              list[[s]]++;
              Do[list[[r]] = 0, {r, 1, s - 1}];
              list[[length + 1]] = j - Sum[list[[s]], {s, 1, length}];
              ,
              list  = {-1} (* We can not increase the last position,
              so we return that this line can not be iterated*)
            ];
          ];
        ];
        list
    ];

ResizeVector[vecN_] :=
    Module[ {value, s, notFound, t},
      notFound = True;
      s = Length[vecN];
      While[notFound && (s > 0),
        If[vecN[[s]] > 0, notFound = False;, s--; ];
      ];
      If[notFound,
        value = {vecN[[1]]};,
        value = Table[vecN[[t]], {t, 1, s}];
      ];
      value
    ];
```

Computation of $J_{n,l}$

```
ComputeNeccaryPointForJnk[xNSpaceOriginal_, d_] :=
  Module[{listout, xlength, nrentries, y1, y2, y3, m, c},

   nrentries = Total[xNSpaceOriginal];
   xlength = Length[xNSpaceOriginal];
   If[xlength == 1,
    (* short vector*)
    If[nrentries == 0,
      listout = {{2 d}, {{0, 1}}};,
```

```
    listout = {{2 (d - nrentries), nrentries, nrentries},
        {{nrentries, 1}, xNSpaceOriginal, {nrentries - 1, 0, 1}}};
   ];
  ,
  (* non-short vector*)
  y1 = xNSpaceOriginal;
  y1[[2]] = xNSpaceOriginal[[2]] + 1;
  listout = {{2 (d - nrentries)}, {y1}};

  Do[
   If[xNSpaceOriginal[[c]] > 0,
     y1 = xNSpaceOriginal; y2 = xNSpaceOriginal;
     y1[[c]] = xNSpaceOriginal[[c]] - 1;
     y2[[c]] = xNSpaceOriginal[[c]] - 1;
     If[xlength > c + 1,
      y1[[c + 2]] = xNSpaceOriginal[[c + 2]] + 1;, (*at least c+2*)
      If[xlength == c,
        y1 = Join[y1, {0, 1}];,
        y1 = Join[y1, {1}]
       ];
     ];
     If[c > 2,
      y2[[c - 2]] = xNSpaceOriginal[[c - 2]] + 1;,
      If[c == 1,
        y2[[c]] = xNSpaceOriginal[[c]] (* For c=1 we recreate x,
        for c=2 we have to do nothing*)
       ];
     ];
     y1 = ResizeVector[y1];
     y2 = ResizeVector[y2];
     listout = {
       Join[listout[[1]], {xNSpaceOriginal[[c]]} , {xNSpaceOriginal[[c]]} ],
       Join[listout[[2]], {y1}, {y2}]
      };
    ];
   , {c, 1, xlength}];
 ];
 listout
];
```

Using the program

Now we use the methods implemented above to compute all points neccesary for the computation of $L_n(x)$.

```
SupplementaryNodes = {};
Do[
 SupplementaryNodes = Union[SupplementaryNodes, ComputeNeccaryPointForLn[v, d]];
 SupplementaryNodes = Union[SupplementaryNodes, ComputeNeccaryPointForJnk[v, d][[2]]];
 , {v, MainNodes}]
AllNodes = Union[SupplementaryNodes, MainNodes];

MainNodes
SupplementaryNodes
```

```
{{0}, {1}, {2}, {0, 1}, {3}, {1, 1}, {0, 0, 1}}
```

```
{{0}, {1}, {2}, {3}, {4}, {6}, {0, 1}, {0, 2}, {0, 3}, {1, 1}, {1, 2}, {2, 1}, {2, 2},
 {3, 1}, {4, 1}, {0, 0, 1}, {0, 0, 2}, {0, 1, 1}, {1, 0, 1}, {1, 1, 1}, {2, 0, 1},
 {0, 0, 0, 1}, {0, 1, 0, 1}, {1, 0, 0, 1}, {2, 0, 0, 1}, {0, 0, 0, 0, 1}, {0, 0, 0, 0, 0, 1}}
```

1.2 Computation of $I_{0,l}(x)$

Reviewing the definition it is easy to see that $(2\,d)^l\, I_{0,l}(x) = p_l(x)$, where $p_l(x)$ is the number of l −step simple random walk ending at x. We use the combinatorics explained in Section 5.1.2 to compute $p_l(x)$.

Main methods

```
(* call by e.g. NumberOfSRWTox[{1,1},7,12] for p7(e1+2e2) in dimension 12*)
NumberOfSRWTox[x_, n_, d_] := Module[
  {count = 0, minUsedDimensions, minimalNumberOfstep, i},
  minimalNumberOfstep = Sum[i * x[[i]], {i, 1, Length[x]}];
  minUsedDimensions = Sum[x[[i]], {i, 1, Length[x]}];
  (* In the following we exclude a number of trivial cases,
  and call the -hard- method only if neccesary.*)
  If[minUsedDimensions > d, count = 0;,
   If[
    minimalNumberOfstep > n, count = 0;(* can not reach x in so few steps*),
    If[ Mod[n - minimalNumberOfstep, 2] == 1, count = 0;
     (* can not reach x in even/odd number of steps*),
     If[n == minimalNumberOfstep, (* we just reach x in this many steps*)
      count = n! / Product[(i!)^x[[i]], {i, 1, Length[x]}];
      ,
      If[0 == minimalNumberOfstep,
       (* do save some implementation we bypass the computation of x=0*)
       count = 2 d * NumberOfSRWToxHard[{1}, n - 1, d, 1, 1];,
       count = NumberOfSRWToxHard[x, n, d, minimalNumberOfstep, minUsedDimensions];]
     ]]]];
  count
 ]
```

```
(* after excluding the trivial case with the former method we now turn to
 the difficult/hard cases*)
NumberOfSRWToxHard[x_, n_, d_, minNumberOfSteps_, minUsedDimensions_] :=
 Module[
  {count2 = 0, nrBuckets, i, toallocate, roughallocationtable, fineallocationtable,
   nrOfdiminitialStep, nrOfinitialSteps, level, tmptable, j},
  (* we computing p_l(x) using simple combinatorics*)
  (* In the first step we use a simple urn model *)
  (* We distribute the free back and forth movements into urn/buckets,
  Each urn repesents a dimension in which we already take 1,2,... steps.*)
  (* We begin by computing how many urns/buckets we need. *)
  nrBuckets = 0;
  For[i = 1 , i < Length[x] + 1, i++,
   If[x[[i]] > 0,
     nrBuckets++;
    ];];
  (*If we have the possibilty to use an additional dimension then we
   create a bucket for the dimension we could, but not have to use*)
  If[minUsedDimensions != d,
   nrBuckets++;
  ];
  (*Now we compute how many back/forth steps we can put into the buckets. *)
  toallocate = (n - minNumberOfSteps) / 2;

  If[nrBuckets < 200, (*Mathematica insists on an deterministic limit of
    the size of the vector to be created*)
   (* Then, we initialise the variable, which we use to save the buckets.*)
   (* the Buckets*)
   roughallocationtable = Table[0, {nrBuckets}];
   (* the alloaction within the buckets*)
   fineallocationtable = Table[{-1}, {nrBuckets}];
   (*Next we reduce x to nrBuckets buckets,
   meaning we remove all zero entries for x and add one bucket for the
    extra dimensions*)
   (* the shorten point *)
   nrOfinitialSteps = Table[0, {nrBuckets}];
   (* the orignal place in x *)
   nrOfdiminitialStep = Table[0, {nrBuckets}];
   If[minUsedDimensions != d,
    nrOfinitialSteps[[nrBuckets]] = 0;
    nrOfdiminitialStep[[nrBuckets]] = d - minUsedDimensions;
```

```
  ];

  j = 1;
  For[i = 1 , i < Length[x] + 1, i++,
   If[(x[[i]] > 0) ,
    nrOfinitialSteps[[j]] = i;
    nrOfdiminitialStep[[j]] = x[[i]];
    j++;
    If[j == nrBuckets + 1, i = Length[x] + 1];
   ](*end of if*)
  ]; (*end of for loop over x*)

  (*Initialisation of the algorithm*)
  If[( x[[nrOfinitialSteps[[1]]]] < 200), (*Mathematice needs a finite limit*)
   fineallocationtable[[1]] = Table[0, { x[[nrOfinitialSteps[[1]]]]}];
   fineallocationtable[[1, 1]] = toallocate;
   roughallocationtable[[1]] = toallocate;
  ];
  level = 1;

  (* After we have no initialized all variables we iterate through all
   combinations. *)
  While[(roughallocationtable ≠ {-1}), (* if ={-1} we have done *)
   count2 += ComputeCombinations[fineallocationtable, roughallocationtable,
     nrOfinitialSteps, nrOfdiminitialStep, n];
   tmptable = IterateAllocations[fineallocationtable, roughallocationtable,
     nrOfdiminitialStep];
   roughallocationtable = tmptable[[1]];
   fineallocationtable = tmptable[[2]];
  ];(*end of While*)
 ]; (*end of technical If*)
 count2
]
```

Distribution of free steps in buckets

We allocate the extra steps to the different dimensions, at this first rough steps we catagorise the different dimension according of how many steps are need to reach the point *x*. This means that when we consider *x = (2, 2, 1),* we consider four buckets : the *(d - 5)* independent dimension where no steps are need, the dimensions in which one step is needed, the dimension in which two steps is needed and the single dimension in which three steps are needed.

```
IterateRoughAllocation[routhallocationtable_, n_] :=
 Module[
  {array = {-1}, lengt, maxallocate, lastNonTrivial, fistsum, j},
  lengt = Length[routhallocationtable];
```

```
  If[(routhallocationtable[[lengt]] == n ) || (lengt == 1),
   array = {-1};(*termination*),
   maxallocate = Sum[routhallocationtable[[j]], {j, 1, lengt}];
   array = Table[routhallocationtable[[j]], {j, 1, lengt}];
   If[maxallocate > 0,
    lastNonTrivial = lengt - 1;
    While[(routhallocationtable[[lastNonTrivial]] == 0) && (lastNonTrivial > 1),
     lastNonTrivial--];
    (*found nontrivial*)
    array[[lastNonTrivial]]--;
    fistsum = Sum[array[[j]], {j, 1, lastNonTrivial}];
    array[[lastNonTrivial + 1]] = n - fistsum;
    For[j = lastNonTrivial + 2, j < lengt + 1, j++, array[[j]] = 0];
   ];
  ];
  array
 ]
(* Now we consider the allocation within a basket of dimensions. These
 are interchangeable
 (the same number of steps in these dimensions need to be taken) *)
InBasketIteration[basketallocation_] :=
  Module[
   {arraytmp = -1, basketlengt, lastdifferent, nextTolastdifferent, c, i, j},
   basketlengt = Length[basketallocation];
   arraytmp = Table[0, {basketlengt}];
   For[i = 1, i < basketlengt + 1, i++, arraytmp[[i]] = basketallocation[[i]];];
   lastdifferent = basketlengt;
   While[(basketallocation[[lastdifferent]] == basketallocation[[basketlengt]]) &&
     (lastdifferent > 1), lastdifferent--];
   If[(lastdifferent == 1) &&
     (basketallocation[[lastdifferent]] == basketallocation[[basketlengt]]),
    arraytmp = {-1};,
    nextTolastdifferent = lastdifferent;
    While[
     (basketallocation[[nextTolastdifferent]] - basketallocation[[basketlengt]] < 2) &&
      (nextTolastdifferent > 1), nextTolastdifferent--];
    If[(nextTolastdifferent == 1) &&
      (basketallocation[[nextTolastdifferent]] - basketallocation[[basketlengt]] < 2),
     arraytmp = {-1};(*Cancel, as we can not distribute anymore*),
     arraytmp[[nextTolastdifferent]]--;
     c = Sum[basketallocation[[j]], {j, nextTolastdifferent + 1, basketlengt}] + 1;
     For[j = nextTolastdifferent + 1, j < basketlengt + 1, j++,
      arraytmp[[j]] = Min[c, arraytmp[[nextTolastdifferent]]];
```

```
      c = Max[0, c - arraytmp[[nextTolastdifferent]]];
     ]
    ]
   ];
   arraytmp
  ];
```

Small methods for testing

```
LoopRoughAllocation[routhallocationtable_, n_] :=
 Module[
  {array = 0, i},
  array = Table[0, {Length[routhallocationtable]}];
  For[i = 1, i < Length[routhallocationtable] + 1, i++,
   array[[i]] = routhallocationtable[[i]];];
  While[array[[1]] ≠ {-1},
   Print[array];
   array = IterateRoughAllocation[array, n];
  ];
 ]
LoopInBasket[basketallocation_] :=
 Module[
  {array = 0, i},
  array = Table[0, {Length[basketallocation]}];
  For[i = 1, i < Length[basketallocation] + 1, i++,
   array[[i]] = basketallocation[[i]];];
  While[(array[[1]] != -1),
   Print[array];
   array = InBasketIteration[array];
  ];
 ]
```

Iteration through the abstract combination of steps

```
IterateAllocations[fineallocationtable_, roughallocationtable_, nrOfdiminitialStep_] :=
  Module[
   {array = 0, tryingfine, level, Finetemp, tmparray, tmprough, i, j},
   array = 1;
   (*interate through table*)
   tryingfine = True;
   level = 1;
   Finetemp = fineallocationtable; (*copy for changes*)
   While[(tryingfine) && level < Length[roughallocationtable] + 1,
    If[roughallocationtable[[level]] == 0, level++,
      tmparray = InBasketIteration[fineallocationtable[[level]]];
      If[tmparray == {-1}, (*iteration on this level failed, so we reset it*)
       tmparray = Table[0, {nrOfdiminitialStep[[level]]}];
       tmparray[[1]] = roughallocationtable[[level]];
       Finetemp[[level]] = tmparray;
       level++;,
       Finetemp[[level]] = tmparray;
       tryingfine = False;
      ];
     ];
   ];
   If[tryingfine, (*we failed to do a fine interation*)
    (*So we iterate rough and reset the fine status*)
    tmprough = IterateRoughAllocation[roughallocationtable,
      Sum[roughallocationtable[[j]], {j, 1, Length[roughallocationtable]}]];
    If[tmprough ≠ {-1},
     For[i = 1, i < Length[Finetemp] + 1, i++,
       If[tmprough[[i]] == 0,
         Finetemp[[i]] = {-1};,
         Finetemp[[i]] = Table[0, {nrOfdiminitialStep[[i]]}];
         Finetemp[[i, 1]] = tmprough[[i]];
        ];
       ];
    ];
    ,
    tmprough = roughallocationtable;
   ];
   array = {tmprough, Finetemp}
  ];
```

Computation of the Number of walks belonging to the abstract allocation

(* Given the number of free back and forth step that are take in a given basket we now compute the number of SRW that have such a path.*)

```
ComputeCombinations[fineallocationtable_, roughallocationtable_, nrOfinitialSteps_,
   nrOfdiminitialStep_, n_] := Module[
```

```
    {number = 0},
    number = ComputeStepPermutations[fineallocationtable, nrOfinitialSteps,
       nrOfdiminitialStep, n] * ComputeDimensionCombinations[fineallocationtable,
       roughallocationtable, nrOfdiminitialStep];
    number
   ];

ComputeStepPermutations[fineallocationtable_, nrOfinitialSteps_,
    nrOfdiminitialStep_, n_] := Module[
    {number2 = 0, tmp1, i, j},
    tmp1 = 1;
    For[i = 1, i < Length[fineallocationtable] + 1, i++,
     If[fineallocationtable[[i]] ≠ {-1},
       For[j = 1, j < Length[fineallocationtable[[i]]] + 1, j++,
        tmp1 = tmp1 * fineallocationtable[[i, j]] ! *
            (fineallocationtable[[i, j]] + nrOfinitialSteps[[i]]) !;
       ],
       tmp1 = tmp1 * (nrOfinitialSteps[[i]] !)^nrOfdiminitialStep[[i]];
      ];
    ];

    number2 = n ! / tmp1;
    number2
   ];

ComputeDimensionCombinations[fineallocationtable_, roughallocationtable_,
    nrOfdiminitialStep_] := Module[{number3 = 0, tmp2, i, occarray, tmp3, k},
    tmp2 = 1;
    For[i = 1, i < Length[roughallocationtable] + 1, i++,
     If[roughallocationtable[[i]] ≠ 0,
       If[Length[fineallocationtable[[i]]] > 1,
         If[fineallocationtable[[i, 2]] == 0,
           tmp2 = tmp2 * nrOfdiminitialStep[[i]];,
           occarray = CountNumbers[fineallocationtable[[i]]];
           tmp3 = nrOfdiminitialStep[[i]] ! / Product[occarray[[k]] !,
               {k, 1, Length[occarray]}];
           tmp2 = tmp2 * tmp3;
          ];
        ];
      ];
    ];
    number3 = tmp2;
    number3
```

```
  ];

(* Compute the number of occurences of each number.*)
CountNumbers[list_] := Module[{number4 = {-1}, m, j},
   If[Length[list] > 1,
    m = Max[list];
    If[m < 200,
     number4 = Table[0, {m + 1}];
     For[j = 1, j < Length[list] + 1, j++,
      If[list[[j]] == 0,
        number4[[m + 1]]++;,
        number4[[list[[j]]]]++;
       ];];];
   ];
   number4
  ];
```

Using the Program

```
changes = False;
Do[
  Do[
   If[! NumericQ[SRWTo[v, n, d]],
     SRWTo[v, n, d] = NumberOfSRWTox[v, n, d];
     changes = True];
   , {n, 0, MaxNumberOfSteps}]
  , {v, AllNodes}];
If[changes,
  DeleteFile[SRWCountFile];
  Save[SRWCountFile, SRWTo];
  ,
  Print["We could use values that have been computed earlier.", d];
 ];
```

```
We could use values that have been computed earlier.11
```

1. 3 Computation of $I_{n,0}(x)$

$$I_{n,0}(x) = \frac{1}{(n-1)!}\int_0^\infty t^{n-1}\prod_{i=1}^{d}\int_{-\pi}^{\pi} e^{-(t/d)(1-\cos(k_i))}\, e^{\mathrm{i}k_i x_i}\,\frac{\mathrm{d}k}{2\pi} = \frac{1}{(n-1)!}\int_0^\infty t^{n-1}\prod_{i=1}^{d} F(t,\, d,\, |x_i|),$$

where *F(t,d,n)* is the modified Besselfunction. We use the technique of Hara and Slade presented in Appendix B of "The lace expansion for self-avoiding walk in five and more dimensions" (1991). In this approach the modified Bessel function is approximated by its truncated Taylor series $T(t, M, N)$ and its truncated asymptotic series $A(t, N, M)$, see (B.33-B.34) in Hara/Slade. Let

```
T[t_, N_, M_] := Exp[-t] (t/2)^N Sum[(t^2/4)^m / (m! (m+N)!), {m, 0, M}];
helpfun1[N_, l_] := If[l == 0, 1, 1/(4^l l!) Product[(4 N^2 - (2 s - 1)^2), {s, 1, l}]];
A[t_, N_, M_] := 1/Sqrt[2 π t] Sum[(-1)^l / (2 t)^l helpfun1[N, l], {l, 0, M}];
FApprox[M_, N_, t_] := If[t < 2000, T[t, N, Max[M, Round[2 * t + 2]]],
   A[t, N, Max[10, N]]];
ComputerPrecision = 30;
```

Using Lemma B.6 we can approximate $I_{n,0}(x)$

```
SRWintApprox[d_, n_, v_, M_, h_] :=
  Module[
   {value = 0},
   If[Length[v] < 1000, (*we need a finite limit*)
    value =
     N[
      1/(n - 1)! d^n h Sum[Exp[n m h] FApprox[M, 0, Exp[m h]]^d-Total[v]
          Product[FApprox[M, i, Exp[m h]]^v[[i]], {i, 1, Length[v]}], {m, -M, M}] +
       1/(n - 1)! d^n/(2 π)^(d/2) (h Exp[-(M + 1) (d/2 - n) h]) / (1 - Exp[-(d/2 - n) h]),
       ComputerPrecision]];
   value];
```

We then compute the values with the following parameters to ensure an accuracy of 10^{-20}:

```
Mu = 256;
hu = 20/256;
nu = 2;

changes = False;

Do[
 Do[
  If[! NumericQ[SRWIntegral[v, n, d]],
    Print[" We begin the computation of the SRW for  ", v, " with n=",
     n, " in ", d, " at ", DateString[]];
    SRWIntegral[v, n, d] = SRWintApprox[d, n, v, Mu, hu] ;
    changes = True];
  , {n, 1, param}]
 , {v, AllNodes}]

If[changes,
  DeleteFile[SRWIntegralFile];
  Save[SRWIntegralFile, SRWIntegral];
  Print["We have computed and saved the SRW-integrals in 0 at ", DateString[]];,
  Print["We could use values that have been computed earlier."];
 ];
Clear[changes];
Print["completed ", d, " in ", DateString[]];
```

```
We could use values that have been computed earlier.
completed 11 in Thu 28 Jul 2016 11:20:39
```

1.4. Computation of $I_{n,l}(x)$

Then, we compute the values of $I_{n,l}(x)$ for different l using $I_{n,l}(x) = I_{n,(l-1)}(x) - I_{(n-1),(l-1)}(x)$:

```
Print["We begin the computation of the SRW integrals at " , DateString[]];
Do[
 tabletmp = Table[0, {s, 1, MaxNumberOfSteps + 2}, {t, 1, 2 + param}];
 (*creating an empty table*)
 tabletmp[[1, 1]] = Text["l \\ n"]; (*formating top/left entry*)
 Do[tabletmp[[1, n + 2]] = n, {n, 0, param}]; (*formating first row of table*)
 Do[tabletmp[[l + 2, 1]] = l, {l, 0, MaxNumberOfSteps}];
 (*formating first coloumn of table*)
 Do[tabletmp[[2, n + 2]] = SRWIntegral[v, n, d] ;, {n, 1, param}];
 (*compute second row*)
 (*Do[tabletmp[[2,n+2]]=1 ;,{n,1,param}];(*compute second row*)*)
 Do[tabletmp[[l + 2, 2]] = SRWTo[v, l, d]/(2 d)^l;, {l, 0, MaxNumberOfSteps}];
 (*fill second coloumn*)
 Do[
  Do[tabletmp[[l + 3, n + 2]] = N[tabletmp[[l + 2, n + 2]] - tabletmp[[l + 2, n + 1]],
      ComputerPrecision], {n, 1, param}];(*fill these rest of the table using,
  I_{n,l}(x)=I_{n,(l-1)}(x)-I_{(n-1),(l-1)}(x)*)
  , {l, 0, MaxNumberOfSteps - 1}];
 SRWTwoPointFunctionTable[v] = tabletmp;
 Clear[n, l, s, t, tabletmp];
 , {v, AllNodes}]

Ivalue[n_, l_, v_] :=
  Module[ {value},
   value = SRWTwoPointFunctionTable[v][[l + 2, n + 2]];
   value
  ];
```

```
We begin the computation of the SRW integrals at Thu 28 Jul 2016 11:20:39
```

1.5. Print-Out of $I_{n,l}(x)$

If you want, then you can print our the result. This is now deactivated. If you want to reactive it, then remove the leading (* and the closing *) in the following input:

```
(*Do[
 NForm[a_]:=NumberForm[N[a] ,5];
 OutputTable=Map[NForm,SRWTwoPointFunctionTable[v],{2}];
 Print[
   Labeled[Grid[OutputTable,
      Alignment -> {{Left, Center}, Baseline,
        {{2, MaxNumberOfSteps+2}, {2, param+2}} -> {"."}}, Frame -> True,
      Dividers -> {{2 -> True, -1 -> True}, {2 -> True}},
      Spacings -> {1.5, {1.5, 1, {0.5}}}, ItemStyle -> {1 -> Bold, 1 -> Bold},
      Background -> {Automatic, Automatic,
        {{2, MaxNumberOfSteps+2}, {2, param+2}} -> GrayLevel[0.9]}],
     Style["Value of the SRW two-point function in dimension "Text[d]Text[v], Bold],
     Top] // Text]
  Clear[OutputTable];
 ,{v,AllNodes}]*)
```

2. Bounds on $L_n(x)$, $K_{n,l}(x)$, $U_{n,l}(x)$ and $T_{n,l}(x)$

Now we use the computed values of $I_{n,l}(x)$ to bound the other SRW integrals, see Section 5.2. We begin with $L_n(x)$ which we can compute explicitly, see (5.7) and (5.17).

$L_n(x)$

```
Do[
   tmp = ComputeLNIntegral[v, d, Table[n, {n, 0, param}]];
   Do[
    L[n, v] = tmp[[n + 1]];
    , {n, 0, param}];
   Clear[tmp];
   , {v, MainNodes}];
```

$K_{n,l}(x)$

Then we bound $K_{n,l}(x)$ as described in (3.36), and then improve the bound for $x = e_1,\ e_1 + e_2,\ 2\,e_1$ using comment ??.

```
Do[
   Do[
     Do[
      K[n, l, v] = Abs[Ivalue[n, 2 l, {0}]^(1/2) L[n, v]^(1/2)];
      , {v, MainNodes}];
     K[n, l, {1}] = Min[Abs[Ivalue[n, 2 l, {0}]^(1/2) L[n, {1}]^(1/2)],
       Abs[Ivalue[n, l + 1, {0}]]];
     , {l, 0, MaxNumberOfSteps / 2}];
   , {n, 0, param}];
```

```
Do[
  Do[
    Do[
      K[n, l, v] = Abs[(Ivalue[n, MaxNumberOfSteps - 2, {0}])^1/2 (L[n, v])^1/2];
      , {v, {{1}, {2}}}];
     , {l, Round[MaxNumberOfSteps / 2] + 1, MaxNumberOfSteps - 4}];
   , {n, 0, param}];

Do[
 Do[
  K[n, 0, v] = Abs[K[n - 1, 0, v] + (Ivalue[n - 1, 2, {0}])^1/2 (L[n - 1, v])^1/2 +
       Ivalue[n, 4, {0}]^1/2 (L[n, v])^1/2];
  , {n, 1, param}], {v, {{1}, {0, 1}, {2}}}]
```

For $x = 0$ we bound use the bound given in (5.14):

```
Do[
  Do[
   K[n, 2 l, {0}] = Ivalue[n, 2 l, {0}];
   , {l, 0, MaxNumberOfSteps / 2}];
  Do[
   K[n, 2 l + 1, {0}] = Abs[√(Ivalue[n, 2 l, {0}] Ivalue[n, 2 l + 2, {0}])];
   , {l, 0, Round[MaxNumberOfSteps / 2] - 1}];
  , {n, 0, param}];
```

$$T_{n,l}(x)$$

We bound $T_{n,l}(x)$ as defined in (3.37) and bounded in (5.11):

```
Do[Do[Do[
    T[n, l, v] = K[n, l + 1, v] + Min[4/d K[n, l, v], 2/d K[n + 1, l, v]];
    , {n, 1, param - 1}];
   T[param, l, v] = K[param, l + 1, v] + 4/d K[param, l, v];
   , {l, 0, MaxNumberOfSteps}], {v, MainNodes}];
```

$$V_{n,l}$$

We compute the bounds on $V_{n,l}$ defined in (3.37) and (5.8), and computed in (5.25):

```
Do[Do[
   V[n, l] =
     1/(2 d)^2 (Ivalue[n, l, {0}] - 2 Ivalue[n, l, {0, 1}] + (d - 1)/d Ivalue[n, l, {0, 2}] +
       1/(2 d) Ivalue[n, l, {0}] + 1/(2 d) Ivalue[n, l, {0, 0, 0, 1}]);
   , {n, 0, param}]
  , {l, 0, MaxNumberOfSteps}];
```

$U_{n,l}(x)$

Then we compute $U_{n,l}(x)$ as given in (3.38) and bounded in (5.9) and (5.12)-(5.13):

```
Do[Do[Do[
    U[n, l, v] = V[n, 2 l]^1/2 L[n, v]^1/2;
    , {v, {{0, 1}, {2}, {3}, {1, 1}, {0, 0, 1}}}];
   If[Mod[l, 2] == 0, U[n, l, {0}] = 1/(2 d) (Ivalue[n, l, {0}] - Ivalue[n, l, {0, 1}]),
    U[n, l, {0}] = 1/d K[n, l, {0}]];
   , {n, 0, param}], {l, 0, 8}];
Do[Do[
   U[n, l, {1}] = Min[V[n, 2 l]^1/2 L[n, {1}]^1/2, U[n, l + 1, {0}]];
   , {l, 0, 5}];
  U[n, 6, {1}] = V[n, 12]^1/2 L[n, {1}]^1/2;
  , {n, 0, param}];
```

Weighted SRW-diagram

We begin with the bound for the initial point, derived in Section 2.3.3, see (2.30).

```
IM[n_, l_, v_] :=
  Module[{result = 10 000, vectors, c},
   If[2 * (n + 3) < d,
    result = Ivalue[n + 2, l + 1, v] - 1/d Ivalue[n + 3, l, v];
    vectors = ComputeNeccaryPointForJnk[v, d];
    Do[
     result = result + 1/(2 d^2) vectors[[1, c]] Ivalue[n + 3, l, vectors[[2, c]] ];
     , {c, 1, Length[vectors[[1]]]}];
    ,
    If[2 * (n + 2) < d,
     result = Ivalue[n + 2, l + 1, v] + 4/d Ivalue[n + 2, l, {0}];];
   ];
   result
  ];
```

3. Number of Self-avoiding and Bond-avoiding walks

We improve our bounds on simple diagrams by explicitely extracting short contributions. For this we have computed the number of self-avoiding walks (SAW) and bond-avoiding walks (BAW) with a specific number of steps. As we use this for all model we save/store these numbers here in the model-independent part of the implementation. These value are computed using a simple JAVA program.

```
ComputedPoints = Join[Table[{i}, {i, 0, 10}], Table[{i, 1}, {i, 0, 8}],
   Table[{i, 0, 1}, {i, 0, 7}], Table[{i, 0, 0, 1}, {i, 0, 6}],
   Table[{i, 0, 0, 0, 1}, {i, 0, 5}], Table[{i, 0, 0, 0, 0, 1}, {i, 0, 4}],
   Table[{i, 0, 0, 0, 0, 0, 1}, {i, 0, 3}], Table[{i, 0, 0, 0, 0, 0, 0, 1}, {i, 0, 2}],
   Table[{i, 0, 0, 0, 0, 0, 0, 0, 1}, {i, 0, 1}], Table[{i, 2}, {i, 0, 6}],
   Table[{i, 0, 2}, {i, 0, 4}], Table[{i, 0, 0, 2}, {i, 0, 2}], Table[{i, 3}, {i, 0, 4}],
   Table[{i, 0, 3}, {i, 0, 1}], Table[{i, 4}, {i, 0, 2}], Table[{i, 5}, {i, 0, 0}],
   Table[{i, 1, 1}, {i, 0, 5}]];

ComputedSteps = 10;
Do[Do[
   nrSAW[n, d, v] = 0;
    nrBAW[n, d, v] = 0;, {n, 0, ComputedSteps}], {v, ComputedPoints}];


(*n=1,SAW*)
nrSAW[1, d, {1}] = If[(d ≥ 1), 1, 0];
(*n=1,BAW*)
nrBAW[1, d, {1}] = If[(d ≥ 1), 1, 0];
(*n=2,SAW*)
```

```
nrSAW[2, d, {0, 1}] = If[(d ≥ 1), 1, 0];
nrSAW[2, d, {2}] = If[(d ≥ 2), 2, 0];

(*n=2,BAW*)
nrSAW[2, d, {0, 1}] = If[(d ≥ 1), 1, 0];
nrBAW[2, d, {2}] = If[(d ≥ 2), 2, 0];

(*n=3,SAW*)
nrSAW[3, d, {1}] = If[(d ≥ 1), 2 (d - 1), 0];
nrSAW[3, d, {0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrSAW[3, d, {1, 1}] = If[(d ≥ 2), 3, 0];
nrSAW[3, d, {3}] = If[(d ≥ 3), 6, 0];
(*n=3,BAW*)
nrBAW[3, d, {1}] = If[(d ≥ 1), 2 (d - 1), 0];
nrBAW[3, d, {0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrBAW[3, d, {1, 1}] = If[(d ≥ 2), 3, 0];
nrBAW[3, d, {3}] = If[(d ≥ 3), 6, 0];

(*n=4,SAW*)
nrSAW[4, d, {2}] = If[(d ≥ 2), 4 + 12 (d - 2), 0];
nrSAW[4, d, {0, 1}] = If[(d ≥ 1), 6 (d - 1), 0];
nrSAW[4, d, {0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrSAW[4, d, {1, 0, 1}] = If[(d ≥ 2), 4, 0];
nrSAW[4, d, {0, 2}] = If[(d ≥ 2), 6, 0];
nrSAW[4, d, {2, 1}] = If[(d ≥ 3), 12, 0];
nrSAW[4, d, {4}] = If[(d ≥ 4), 24, 0];
(*n=4,BAW*)
nrBAW[4, d, {2}] = If[(d ≥ 2), 4 + 12 (d - 2), 0];
nrBAW[4, d, {0, 1}] = If[(d ≥ 1), 6 (d - 1), 0];
nrBAW[4, d, {0}] = If[(d ≥ 0), 4 d (d - 1), 0];
nrBAW[4, d, {0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrBAW[4, d, {1, 0, 1}] = If[(d ≥ 2), 4, 0];
nrBAW[4, d, {0, 2}] = If[(d ≥ 2), 6, 0];
nrBAW[4, d, {2, 1}] = If[(d ≥ 3), 12, 0];
nrBAW[4, d, {4}] = If[(d ≥ 4), 24, 0];

(*n=5,SAW*)
nrSAW[5, d, {3}] = If[(d ≥ 3), 54 + 72 (d - 3), 0];
nrSAW[5, d, {1, 1}] = If[(d ≥ 2), 11 + 36 (d - 2), 0];
nrSAW[5, d, {1}] = If[(d ≥ 1), 6 (d - 1) + 16 (d - 1) (d - 2), 0];
nrSAW[5, d, {0, 0, 1}] = If[(d ≥ 1), 12 (d - 1), 0];
nrSAW[5, d, {0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrSAW[5, d, {1, 0, 0, 1}] = If[(d ≥ 2), 5, 0];
```

```
nrSAW[5, d, {0, 1, 1}] = If[(d ≥ 2), 10, 0];
nrSAW[5, d, {2, 0, 1}] = If[(d ≥ 3), 20, 0];
nrSAW[5, d, {1, 2}] = If[(d ≥ 3), 30, 0];
nrSAW[5, d, {3, 1}] = If[(d ≥ 4), 60, 0];
nrSAW[5, d, {5}] = If[(d ≥ 5), 120, 0];
(*n=5,BAW*)
nrBAW[5, d, {3}] = If[(d ≥ 3), 54 + 72 (d - 3), 0];
nrBAW[5, d, {1, 1}] = If[(d ≥ 2), 11 + 36 (d - 2), 0];
nrBAW[5, d, {1}] = If[(d ≥ 1), 14 (d - 1) + 24 (d - 1) (d - 2), 0];
nrBAW[5, d, {0, 0, 1}] = If[(d ≥ 1), 12 (d - 1), 0];
nrBAW[5, d, {0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrBAW[5, d, {1, 0, 0, 1}] = If[(d ≥ 2), 5, 0];
nrBAW[5, d, {0, 1, 1}] = If[(d ≥ 2), 10, 0];
nrBAW[5, d, {2, 0, 1}] = If[(d ≥ 3), 20, 0];
nrBAW[5, d, {1, 2}] = If[(d ≥ 3), 30, 0];
nrBAW[5, d, {3, 1}] = If[(d ≥ 4), 60, 0];
nrBAW[5, d, {5}] = If[(d ≥ 5), 120, 0];
(*n=6,SAW*)
nrSAW[6, d, {4}] = If[(d ≥ 4), 576 + 480 (d - 4), 0];
nrSAW[6, d, {2, 1}] = If[(d ≥ 3), 168 + 240 (d - 3), 0];
nrSAW[6, d, {2}] = If[(d ≥ 2), 16 + 168 (d - 2) + 144 (d - 2) (d - 3), 0];
nrSAW[6, d, {1, 0, 1}] = If[(d ≥ 2), 26 + 80 (d - 2), 0];
nrSAW[6, d, {0, 2}] = If[(d ≥ 2), 24 + 120 (d - 2), 0];
nrSAW[6, d, {0, 1}] = If[(d ≥ 1), 20 (d - 1) + 72 (d - 1) (d - 2), 0];
nrSAW[6, d, {0, 0, 0, 1}] = If[(d ≥ 1), 20 (d - 1), 0];
nrSAW[6, d, {0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrSAW[6, d, {1, 0, 0, 0, 1}] = If[(d ≥ 2), 6, 0];
nrSAW[6, d, {0, 1, 0, 1}] = If[(d ≥ 2), 15, 0];
nrSAW[6, d, {2, 0, 0, 1}] = If[(d ≥ 3), 30, 0];
nrSAW[6, d, {0, 0, 2}] = If[(d ≥ 2), 20, 0];
nrSAW[6, d, {1, 1, 1}] = If[(d ≥ 3), 60, 0];
nrSAW[6, d, {3, 0, 1}] = If[(d ≥ 4), 120, 0];
nrSAW[6, d, {0, 3}] = If[(d ≥ 3), 90, 0];
nrSAW[6, d, {2, 2}] = If[(d ≥ 4), 180, 0];
nrSAW[6, d, {4, 1}] = If[(d ≥ 5), 360, 0];
nrSAW[6, d, {6}] = If[(d ≥ 6), 720, 0];
(*n=6,BAW*)
nrBAW[6, d, {4}] = If[(d ≥ 4), 576 + 480 (d - 4), 0];
nrBAW[6, d, {2, 1}] = If[(d ≥ 3), 168 + 240 (d - 3), 0];
nrBAW[6, d, {2}] = If[(d ≥ 2), 36 + 232 (d - 2) + 168 (d - 2) (d - 3), 0];
nrBAW[6, d, {1, 0, 1}] = If[(d ≥ 2), 26 + 80 (d - 2), 0];
nrBAW[6, d, {0, 2}] = If[(d ≥ 2), 24 + 120 (d - 2), 0];
nrBAW[6, d, {0, 1}] = If[(d ≥ 1), 28 (d - 1) + 84 (d - 1) (d - 2), 0];
```

```
nrBAW[6, d, {0, 0, 0, 1}] = If[(d ≥ 1), 20 (d - 1), 0];
nrBAW[6, d, {0}] = If[(d ≥ 0), 12 d (d - 1) + 32 d (d - 1) (d - 2), 0];
nrBAW[6, d, {0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrBAW[6, d, {1, 0, 0, 0, 1}] = If[(d ≥ 2), 6, 0];
nrBAW[6, d, {0, 1, 0, 1}] = If[(d ≥ 2), 15, 0];
nrBAW[6, d, {2, 0, 0, 1}] = If[(d ≥ 3), 30, 0];
nrBAW[6, d, {0, 0, 2}] = If[(d ≥ 2), 20, 0];
nrBAW[6, d, {1, 1, 1}] = If[(d ≥ 3), 60, 0];
nrBAW[6, d, {3, 0, 1}] = If[(d ≥ 4), 120, 0];
nrBAW[6, d, {0, 3}] = If[(d ≥ 3), 90, 0];
nrBAW[6, d, {2, 2}] = If[(d ≥ 4), 180, 0];
nrBAW[6, d, {4, 1}] = If[(d ≥ 5), 360, 0];
nrBAW[6, d, {6}] = If[(d ≥ 6), 720, 0];

(*n=7,SAW*)
nrSAW[7, d, {5}] = If[(d ≥ 5), 6000 + 3600 (d - 5), 0];
nrSAW[7, d, {3, 1}] = If[(d ≥ 4), 2040 + 1800 (d - 4), 0];
nrSAW[7, d, {3}] = If[(d ≥ 3), 708 + 2520 (d - 3) + 1224 (d - 3) (d - 4), 0];
nrSAW[7, d, {2, 0, 1}] = If[(d ≥ 3), 430 + 600 (d - 3), 0];
nrSAW[7, d, {1, 2}] = If[(d ≥ 3), 540 + 900 (d - 3), 0];
nrSAW[7, d, {1, 1}] = If[(d ≥ 2), 45 + 636 (d - 2) + 612 (d - 2) (d - 3), 0];
nrSAW[7, d, {1, 0, 0, 1}] = If[(d ≥ 2), 52 + 150 (d - 2), 0];
nrSAW[7, d, {0, 1, 1}] = If[(d ≥ 2), 55 + 300 (d - 2), 0];
nrSAW[7, d, {1}] = If[(d ≥ 1), 28 (d - 1) + 248 (d - 1) (d - 2) + 216 (d - 1) (d - 2) (d - 3), 0];
nrSAW[7, d, {0, 0, 1}] = If[(d ≥ 1), 54 (d - 1) + 204 (d - 1) (d - 2), 0];
nrSAW[7, d, {0, 0, 0, 0, 1}] = If[(d ≥ 1), 30 (d - 1), 0];
nrSAW[7, d, {0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrSAW[7, d, {1, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 7, 0];
nrSAW[7, d, {0, 1, 0, 0, 1}] = If[(d ≥ 2), 21, 0];
nrSAW[7, d, {2, 0, 0, 0, 1}] = If[(d ≥ 3), 42, 0];
nrSAW[7, d, {0, 0, 1, 1}] = If[(d ≥ 2), 35, 0];
nrSAW[7, d, {1, 1, 0, 1}] = If[(d ≥ 3), 105, 0];
nrSAW[7, d, {3, 0, 0, 1}] = If[(d ≥ 4), 210, 0];
nrSAW[7, d, {1, 0, 2}] = If[(d ≥ 3), 140, 0];
nrSAW[7, d, {0, 2, 1}] = If[(d ≥ 3), 210, 0];
nrSAW[7, d, {2, 1, 1}] = If[(d ≥ 4), 420, 0];
nrSAW[7, d, {4, 0, 1}] = If[(d ≥ 5), 840, 0];
nrSAW[7, d, {1, 3}] = If[(d ≥ 4), 630, 0];
nrSAW[7, d, {3, 2}] = If[(d ≥ 5), 1260, 0];
nrSAW[7, d, {5, 1}] = If[(d ≥ 6), 2520, 0];
nrSAW[7, d, {7}] = If[(d ≥ 7), 5040, 0];
(*n=7,BAW*)
nrBAW[7, d, {5}] = If[(d ≥ 5), 6000 + 3600 (d - 5), 0];
```

```
nrBAW[7, d, {3, 1}] = If[(d ≥ 4), 2040 + 1800 (d - 4), 0];
nrBAW[7, d, {3}] = If[(d ≥ 3), 1020 + 2952 (d - 3) + 1320 (d - 3) (d - 4), 0];
nrBAW[7, d, {2, 0, 1}] = If[(d ≥ 3), 430 + 600 (d - 3), 0];
nrBAW[7, d, {1, 2}] = If[(d ≥ 3), 540 + 900 (d - 3), 0];
nrBAW[7, d, {1, 1}] = If[(d ≥ 2), 77 + 756 (d - 2) + 660 (d - 2) (d - 3), 0];
nrBAW[7, d, {1, 0, 0, 1}] = If[(d ≥ 2), 52 + 150 (d - 2), 0];
nrBAW[7, d, {0, 1, 1}] = If[(d ≥ 2), 55 + 300 (d - 2), 0];
nrBAW[7, d, {1}] = If[(d ≥ 1), 70 (d - 1) + 468 (d - 1) (d - 2) + 312 (d - 1) (d - 2) (d - 3), 0];
nrBAW[7, d, {0, 0, 1}] = If[(d ≥ 1), 62 (d - 1) + 220 (d - 1) (d - 2), 0];
nrBAW[7, d, {0, 0, 0, 0, 1}] = If[(d ≥ 1), 30 (d - 1), 0];
nrBAW[7, d, {0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrBAW[7, d, {1, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 7, 0];
nrBAW[7, d, {0, 1, 0, 0, 1}] = If[(d ≥ 2), 21, 0];
nrBAW[7, d, {2, 0, 0, 0, 1}] = If[(d ≥ 3), 42, 0];
nrBAW[7, d, {0, 0, 1, 1}] = If[(d ≥ 2), 35, 0];
nrBAW[7, d, {1, 1, 0, 1}] = If[(d ≥ 3), 105, 0];
nrBAW[7, d, {3, 0, 0, 1}] = If[(d ≥ 4), 210, 0];
nrBAW[7, d, {1, 0, 2}] = If[(d ≥ 3), 140, 0];
nrBAW[7, d, {0, 2, 1}] = If[(d ≥ 3), 210, 0];
nrBAW[7, d, {2, 1, 1}] = If[(d ≥ 4), 420, 0];
nrBAW[7, d, {4, 0, 1}] = If[(d ≥ 5), 840, 0];
nrBAW[7, d, {1, 3}] = If[(d ≥ 4), 630, 0];
nrBAW[7, d, {3, 2}] = If[(d ≥ 5), 1260, 0];
nrBAW[7, d, {5, 1}] = If[(d ≥ 6), 2520, 0];
nrBAW[7, d, {7}] = If[(d ≥ 7), 5040, 0];


(*n=8,SAW*)
nrSAW[8, d, {6}] = If[(d ≥ 6), 64 800 + 30 240 (d - 6), 0];
nrSAW[8, d, {4, 1}] = If[(d ≥ 5), 24 000 + 15 120 (d - 5), 0];
nrSAW[8, d, {4}] = If[(d ≥ 4), 16 296 + 32 976 (d - 4) + 11 040 (d - 4) (d - 5), 0];
nrSAW[8, d, {3, 0, 1}] = If[(d ≥ 4), 5760 + 5040 (d - 4), 0];
nrSAW[8, d, {2, 2}] = If[(d ≥ 4), 7800 + 7560 (d - 4), 0];
nrSAW[8, d, {2, 1}] = If[(d ≥ 3), 2586 + 10 488 (d - 3) + 5520 (d - 3) (d - 4), 0];
nrSAW[8, d, {2, 0, 0, 1}] = If[(d ≥ 3), 936 + 1260 (d - 3), 0];
nrSAW[8, d, {1, 1, 1}] = If[(d ≥ 3), 1480 + 2520 (d - 3), 0];
nrSAW[8, d, {0, 3}] = If[(d ≥ 3), 1800 + 3780 (d - 3), 0];
nrSAW[8, d, {2}] =
  If[(d ≥ 2), 76 + 2468 (d - 2) + 6192 (d - 2) (d - 3) + 2480 (d - 2) (d - 3) (d - 4), 0];
nrSAW[8, d, {1, 0, 1}] = If[(d ≥ 2), 118 + 1916 (d - 2) + 1840 (d - 2) (d - 3), 0];
nrSAW[8, d, {0, 2}] = If[(d ≥ 2), 112 + 2244 (d - 2) + 2760 (d - 2) (d - 3), 0];
nrSAW[8, d, {1, 0, 0, 0, 1}] = If[(d ≥ 2), 92 + 252 (d - 2), 0];
nrSAW[8, d, {0, 1, 0, 1}] = If[(d ≥ 2), 118 + 630 (d - 2), 0];
```

```
nrSAW[8, d, {0, 0, 2}] = If[(d ≥ 2), 120 + 840 (d - 2), 0];
nrSAW[8, d, {0, 1}] = If[(d ≥ 1), 92 (d - 1) + 1216 (d - 1) (d - 2) + 1240 (d - 1) (d - 2) (d - 3),
   0];
nrSAW[8, d, {0, 0, 0, 1}] = If[(d ≥ 1), 126 (d - 1) + 460 (d - 1) (d - 2), 0];
nrSAW[8, d, {0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 42 (d - 1), 0];
nrSAW[8, d, {0, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrSAW[8, d, {1, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 8, 0];
nrSAW[8, d, {0, 1, 0, 0, 0, 1}] = If[(d ≥ 2), 28, 0];
nrSAW[8, d, {2, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 56, 0];
nrSAW[8, d, {0, 0, 1, 0, 1}] = If[(d ≥ 2), 56, 0];
nrSAW[8, d, {1, 1, 0, 0, 1}] = If[(d ≥ 3), 168, 0];
nrSAW[8, d, {3, 0, 0, 0, 1}] = If[(d ≥ 4), 336, 0];
nrSAW[8, d, {0, 0, 0, 2}] = If[(d ≥ 2), 70, 0];
nrSAW[8, d, {1, 0, 1, 1}] = If[(d ≥ 3), 280, 0];
nrSAW[8, d, {0, 2, 0, 1}] = If[(d ≥ 3), 420, 0];
nrSAW[8, d, {2, 1, 0, 1}] = If[(d ≥ 4), 840, 0];
nrSAW[8, d, {4, 0, 0, 1}] = If[(d ≥ 5), 1680, 0];
nrSAW[8, d, {0, 1, 2}] = If[(d ≥ 3), 560, 0];
nrSAW[8, d, {2, 0, 2}] = If[(d ≥ 4), 1120, 0];
nrSAW[8, d, {1, 2, 1}] = If[(d ≥ 4), 1680, 0];
nrSAW[8, d, {3, 1, 1}] = If[(d ≥ 5), 3360, 0];
nrSAW[8, d, {5, 0, 1}] = If[(d ≥ 6), 6720, 0];
nrSAW[8, d, {0, 4}] = If[(d ≥ 4), 2520, 0];
nrSAW[8, d, {2, 3}] = If[(d ≥ 5), 5040, 0];
nrSAW[8, d, {4, 2}] = If[(d ≥ 6), 10 080, 0];
nrSAW[8, d, {6, 1}] = If[(d ≥ 7), 20 160, 0];
nrSAW[8, d, {8}] = If[(d ≥ 8), 40 320, 0];
(*n=8,BAW*)
nrBAW[8, d, {6}] = If[(d ≥ 6), 64 800 + 30 240 (d - 6), 0];
nrBAW[8, d, {4, 1}] = If[(d ≥ 5), 24 000 + 15 120 (d - 5), 0];
nrBAW[8, d, {4}] = If[(d ≥ 4), 19 896 + 36 048 (d - 4) + 11 520 (d - 4) (d - 5), 0];
nrBAW[8, d, {3, 0, 1}] = If[(d ≥ 4), 5760 + 5040 (d - 4), 0];
nrBAW[8, d, {2, 2}] = If[(d ≥ 4), 7800 + 7560 (d - 4), 0];
nrBAW[8, d, {2, 1}] = If[(d ≥ 3), 3318 + 11 544 (d - 3) + 5760 (d - 3) (d - 4), 0];
nrBAW[8, d, {2, 0, 0, 1}] = If[(d ≥ 3), 936 + 1260 (d - 3), 0];
nrBAW[8, d, {1, 1, 1}] = If[(d ≥ 3), 1480 + 2520 (d - 3), 0];
nrBAW[8, d, {0, 3}] = If[(d ≥ 3), 1800 + 3780 (d - 3), 0];
nrBAW[8, d, {2}] =
  If[(d ≥ 2), 164 + 4140 (d - 2) + 8248 (d - 2) (d - 3) + 2912 (d - 2) (d - 3) (d - 4), 0];
nrBAW[8, d, {1, 0, 1}] = If[(d ≥ 2), 162 + 2108 (d - 2) + 1920 (d - 2) (d - 3), 0];
nrBAW[8, d, {0, 2}] = If[(d ≥ 2), 184 + 2532 (d - 2) + 2880 (d - 2) (d - 3), 0];
nrBAW[8, d, {1, 0, 0, 0, 1}] = If[(d ≥ 2), 92 + 252 (d - 2), 0];
nrBAW[8, d, {0, 1, 0, 1}] = If[(d ≥ 2), 118 + 630 (d - 2), 0];
```

```
nrBAW[8, d, {0, 0, 2}] = If[(d ≥ 2), 120 + 840 (d - 2), 0];
nrBAW[8, d, {0, 1}] =
  If[(d ≥ 1), 188 (d - 1) + 1676 (d - 1) (d - 2) + 1456 (d - 1) (d - 2) (d - 3), 0];
nrBAW[8, d, {0, 0, 0, 1}] = If[(d ≥ 1), 134 (d - 1) + 480 (d - 1) (d - 2), 0];
nrBAW[8, d, {0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 42 (d - 1), 0];
nrBAW[8, d, {0}] = If[(d ≥ 0), 88 d (d - 1) + 624 d (d - 1) (d - 2) + 496 d (d - 1) (d - 2) (d - 3),
   0];
nrBAW[8, d, {0, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrBAW[8, d, {1, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 8, 0];
nrBAW[8, d, {0, 1, 0, 0, 0, 1}] = If[(d ≥ 2), 28, 0];
nrBAW[8, d, {2, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 56, 0];
nrBAW[8, d, {0, 0, 1, 0, 1}] = If[(d ≥ 2), 56, 0];
nrBAW[8, d, {1, 1, 0, 0, 1}] = If[(d ≥ 3), 168, 0];
nrBAW[8, d, {3, 0, 0, 0, 1}] = If[(d ≥ 4), 336, 0];
nrBAW[8, d, {0, 0, 0, 2}] = If[(d ≥ 2), 70, 0];
nrBAW[8, d, {1, 0, 1, 1}] = If[(d ≥ 3), 280, 0];
nrBAW[8, d, {0, 2, 0, 1}] = If[(d ≥ 3), 420, 0];
nrBAW[8, d, {2, 1, 0, 1}] = If[(d ≥ 4), 840, 0];
nrBAW[8, d, {4, 0, 0, 1}] = If[(d ≥ 5), 1680, 0];
nrBAW[8, d, {0, 1, 2}] = If[(d ≥ 3), 560, 0];
nrBAW[8, d, {2, 0, 2}] = If[(d ≥ 4), 1120, 0];
nrBAW[8, d, {1, 2, 1}] = If[(d ≥ 4), 1680, 0];
nrBAW[8, d, {3, 1, 1}] = If[(d ≥ 5), 3360, 0];
nrBAW[8, d, {5, 0, 1}] = If[(d ≥ 6), 6720, 0];
nrBAW[8, d, {0, 4}] = If[(d ≥ 4), 2520, 0];
nrBAW[8, d, {2, 3}] = If[(d ≥ 5), 5040, 0];
nrBAW[8, d, {4, 2}] = If[(d ≥ 6), 10 080, 0];
nrBAW[8, d, {6, 1}] = If[(d ≥ 7), 20 160, 0];
nrBAW[8, d, {8}] = If[(d ≥ 8), 40 320, 0];

(*n=9, SAW*)
nrSAW[9, d, {7}] = If[(d ≥ 7), 740 880 + 282 240 (d - 7), 0];
nrSAW[9, d, {5, 1}] = If[(d ≥ 6), 289 800 + 141 120 (d - 6), 0];
nrSAW[9, d, {5}] = If[(d ≥ 5), 305 520 + 424 800 (d - 5) + 108 000 (d - 5) (d - 6), 0];
nrSAW[9, d, {4, 0, 1}] = If[(d ≥ 5), 74 760 + 47 040 (d - 5), 0];
nrSAW[9, d, {3, 2}] = If[(d ≥ 5), 104 580 + 70 560 (d - 5), 0];
nrSAW[9, d, {3, 1}] = If[(d ≥ 4), 68 544 + 151 440 (d - 4) + 54 000 (d - 4) (d - 5), 0];
nrSAW[9, d, {3, 0, 0, 1}] = If[(d ≥ 4), 13 734 + 11 760 (d - 4), 0];
nrSAW[9, d, {2, 1, 1}] = If[(d ≥ 4), 23 940 + 23 520 (d - 4), 0];
nrSAW[9, d, {1, 3}] = If[(d ≥ 4), 32 130 + 35 280 (d - 4), 0];
nrSAW[9, d, {3}] =
  If[(d ≥ 3), 10 518 + 84 024 (d - 3) + 103 824 (d - 3) (d - 4) + 26 592 (d - 3) (d - 4) (d - 5), 0];
nrSAW[9, d, {2, 0, 1}] = If[(d ≥ 3), 8146 + 34 080 (d - 3) + 18 000 (d - 3) (d - 4), 0];
```

```
nrSAW[9, d, {1, 2}] = If[(d ≥ 3), 9204 + 45240 (d - 3) + 27000 (d - 3) (d - 4), 0];
nrSAW[9, d, {2, 0, 0, 0, 1}] = If[(d ≥ 3), 1806 + 2352 (d - 3), 0];
nrSAW[9, d, {1, 1, 0, 1}] = If[(d ≥ 3), 3507 + 5880 (d - 3), 0];
nrSAW[9, d, {1, 0, 2}] = If[(d ≥ 3), 4340 + 7840 (d - 3), 0];
nrSAW[9, d, {0, 2, 1}] = If[(d ≥ 3), 5250 + 11760 (d - 3), 0];
nrSAW[9, d, {1, 1}] =
  If[(d ≥ 2), 220 + 10180 (d - 2) + 30024 (d - 2) (d - 3) + 13296 (d - 2) (d - 3) (d - 4), 0];
nrSAW[9, d, {1, 0, 0, 1}] = If[(d ≥ 2), 286 + 4812 (d - 2) + 4500 (d - 2) (d - 3), 0];
nrSAW[9, d, {0, 1, 1}] = If[(d ≥ 2), 277 + 6880 (d - 2) + 9000 (d - 2) (d - 3), 0];
nrSAW[9, d, {1, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 149 + 392 (d - 2), 0];
nrSAW[9, d, {0, 1, 0, 0, 1}] = If[(d ≥ 2), 231 + 1176 (d - 2), 0];
nrSAW[9, d, {0, 0, 1, 1}] = If[(d ≥ 2), 259 + 1960 (d - 2), 0];
nrSAW[9, d, {1}] =
  If[(d ≥ 1), 140 (d - 1) + 3880 (d - 1) (d - 2) + 9640 (d - 1) (d - 2) (d - 3) +
    3968 (d - 1) (d - 2) (d - 3) (d - 4), 0];
nrSAW[9, d, {0, 0, 1}] =
  If[(d ≥ 3), 256 (d - 1) + 4212 (d - 1) (d - 2) + 4432 (d - 1) (d - 2) (d - 3), 0];
nrSAW[9, d, {0, 0, 0, 0, 1}] = If[(d ≥ 2), 260 (d - 1) + 900 (d - 1) (d - 2), 0];
nrSAW[9, d, {0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 56 (d - 1), 0];
nrSAW[9, d, {1, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 9, 0];
nrSAW[9, d, {0, 1, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 36, 0];
nrSAW[9, d, {2, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 72, 0];
nrSAW[9, d, {0, 0, 1, 0, 0, 1}] = If[(d ≥ 2), 84, 0];
nrSAW[9, d, {1, 1, 0, 0, 0, 1}] = If[(d ≥ 3), 252, 0];
nrSAW[9, d, {3, 0, 0, 0, 0, 1}] = If[(d ≥ 4), 504, 0];
nrSAW[9, d, {0, 0, 0, 1, 1}] = If[(d ≥ 2), 126, 0];
nrSAW[9, d, {1, 0, 1, 0, 1}] = If[(d ≥ 3), 504, 0];
nrSAW[9, d, {0, 2, 0, 0, 1}] = If[(d ≥ 3), 756, 0];
nrSAW[9, d, {2, 1, 0, 0, 1}] = If[(d ≥ 4), 1512, 0];
nrSAW[9, d, {4, 0, 0, 0, 1}] = If[(d ≥ 5), 3024, 0];
nrSAW[9, d, {1, 0, 0, 2}] = If[(d ≥ 3), 630, 0];
nrSAW[9, d, {0, 1, 1, 1}] = If[(d ≥ 3), 1260, 0];
nrSAW[9, d, {2, 0, 1, 1}] = If[(d ≥ 4), 2520, 0];
nrSAW[9, d, {1, 2, 0, 1}] = If[(d ≥ 4), 3780, 0];
nrSAW[9, d, {3, 1, 0, 1}] = If[(d ≥ 5), 7560, 0];
nrSAW[9, d, {5, 0, 0, 1}] = If[(d ≥ 6), 15120, 0];
nrSAW[9, d, {0, 0, 3}] = If[(d ≥ 3), 1680, 0];
nrSAW[9, d, {1, 1, 2}] = If[(d ≥ 4), 5040, 0];
nrSAW[9, d, {3, 0, 2}] = If[(d ≥ 5), 10080, 0];
nrSAW[9, d, {0, 3, 1}] = If[(d ≥ 4), 7560, 0];
nrSAW[9, d, {2, 2, 1}] = If[(d ≥ 5), 15120, 0];
nrSAW[9, d, {4, 1, 1}] = If[(d ≥ 6), 30240, 0];
nrSAW[9, d, {6, 0, 1}] = If[(d ≥ 7), 60480, 0];
```

```
nrSAW[9, d, {1, 4}] = If[(d ≥ 5), 22 680, 0];
nrSAW[9, d, {3, 3}] = If[(d ≥ 6), 45 360, 0];
nrSAW[9, d, {5, 2}] = If[(d ≥ 7), 90 720, 0];
nrSAW[9, d, {7, 1}] = If[(d ≥ 8), 181 440, 0];
nrSAW[9, d, {9}] = If[(d ≥ 9), 362 880, 0];

(*n=9, BAW*)
nrBAW[9, d, {7}] = If[(d ≥ 7), 740 880 + 282 240 (d - 7), 0];
nrBAW[9, d, {5, 1}] = If[(d ≥ 6), 289 800 + 141 120 (d - 6), 0];
nrBAW[9, d, {5}] = If[(d ≥ 5), 344 880 + 448 800 (d - 5) + 110 880 (d - 5) (d - 6), 0];
nrBAW[9, d, {4, 0, 1}] = If[(d ≥ 5), 74 760 + 47 040 (d - 5), 0];
nrBAW[9, d, {3, 2}] = If[(d ≥ 5), 104 580 + 70 560 (d - 5), 0];
nrBAW[9, d, {3, 1}] = If[(d ≥ 4), 79 056 + 160 560 (d - 4) + 55 440 (d - 4) (d - 5), 0];
nrBAW[9, d, {3, 0, 0, 1}] = If[(d ≥ 4), 13 734 + 11 760 (d - 4), 0];
nrBAW[9, d, {2, 1, 1}] = If[(d ≥ 4), 23 940 + 23 520 (d - 4), 0];
nrBAW[9, d, {1, 3}] = If[(d ≥ 4), 32 130 + 35 280 (d - 4), 0];
nrBAW[9, d, {3}] =
  If[(d ≥ 3), 17 118 + 111 996 (d - 3) + 121 896 (d - 3) (d - 4) + 9088 (d - 3) (d - 4) (d - 5), 0];
nrBAW[9, d, {2, 0, 1}] = If[(d ≥ 3), 9538 + 36 160 (d - 3) + 18 480 (d - 3) (d - 4), 0];
nrBAW[9, d, {1, 2}] = If[(d ≥ 3), 11 316 + 48 360 (d - 3) + 27 720 (d - 3) (d - 4), 0];
nrBAW[9, d, {2, 0, 0, 0, 1}] = If[(d ≥ 3), 1806 + 2352 (d - 3), 0];
nrBAW[9, d, {1, 1, 0, 1}] = If[(d ≥ 3), 3507 + 5880 (d - 3), 0];
nrBAW[9, d, {1, 0, 2}] = If[(d ≥ 3), 4340 + 7840 (d - 3), 0];
nrBAW[9, d, {0, 2, 1}] = If[(d ≥ 3), 5250 + 11 760 (d - 3), 0];
nrBAW[9, d, {1, 1}] =
  If[(d ≥ 2), 432 + 14 662 (d - 2) + 35 796 (d - 2) (d - 3) + 14 544 (d - 2) (d - 3) (d - 4), 0];
nrBAW[9, d, {1, 0, 0, 1}] = If[(d ≥ 2), 342 + 5092 (d - 2) + 4620 (d - 2) (d - 3), 0];
nrBAW[9, d, {0, 1, 1}] = If[(d ≥ 2), 405 + 7440 (d - 2) + 9240 (d - 2) (d - 3), 0];
nrBAW[9, d, {1, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 149 + 392 (d - 2), 0];
nrBAW[9, d, {0, 1, 0, 0, 1}] = If[(d ≥ 2), 231 + 1176 (d - 2), 0];
nrBAW[9, d, {0, 0, 1, 1}] = If[(d ≥ 2), 259 + 1960 (d - 2), 0];
nrBAW[9, d, {1}] =
  If[(d ≥ 1), 408 (d - 1) + 8656 (d - 1) (d - 2) + 16 712 (d - 1) (d - 2) (d - 3) +
    5616 (d - 1) (d - 2) (d - 3) (d - 4), 0];
nrBAW[9, d, {0, 0, 1}] =
  If[(d ≥ 1), 438 (d - 1) + 5088 (d - 1) (d - 2) + 4848 (d - 1) (d - 2) (d - 3), 0];
nrBAW[9, d, {0, 0, 0, 0, 1}] = If[(d ≥ 1), 268 (d - 1) + 924 (d - 1) (d - 2), 0];
nrBAW[9, d, {0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 56 (d - 1), 0];
nrBAW[9, d, {1, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 9, 0];
nrBAW[9, d, {0, 1, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 36, 0];
nrBAW[9, d, {2, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 72, 0];
nrBAW[9, d, {0, 0, 1, 0, 0, 1}] = If[(d ≥ 3), 84, 0];
nrBAW[9, d, {1, 1, 0, 0, 0, 1}] = If[(d ≥ 3), 252, 0];
```

```
nrBAW[9, d, {3, 0, 0, 0, 0, 1}] = If[(d ≥ 4), 504, 0];
nrBAW[9, d, {0, 0, 0, 1, 1}] = If[(d ≥ 2), 126, 0];
nrBAW[9, d, {1, 0, 1, 0, 1}] = If[(d ≥ 3), 504, 0];
nrBAW[9, d, {0, 2, 0, 0, 1}] = If[(d ≥ 3), 756, 0];
nrBAW[9, d, {2, 1, 0, 0, 1}] = If[(d ≥ 4), 1512, 0];
nrBAW[9, d, {4, 0, 0, 0, 1}] = If[(d ≥ 5), 3024, 0];
nrBAW[9, d, {1, 0, 0, 2}] = If[(d ≥ 3), 630, 0];
nrBAW[9, d, {0, 1, 1, 1}] = If[(d ≥ 3), 1260, 0];
nrBAW[9, d, {2, 0, 1, 1}] = If[(d ≥ 4), 2520, 0];
nrBAW[9, d, {1, 2, 0, 1}] = If[(d ≥ 4), 3780, 0];
nrBAW[9, d, {3, 1, 0, 1}] = If[(d ≥ 5), 7560, 0];
nrBAW[9, d, {5, 0, 0, 1}] = If[(d ≥ 6), 15120, 0];
nrBAW[9, d, {0, 0, 3}] = If[(d ≥ 3), 1680, 0];
nrBAW[9, d, {1, 1, 2}] = If[(d ≥ 4), 5040, 0];
nrBAW[9, d, {3, 0, 2}] = If[(d ≥ 5), 10080, 0];
nrBAW[9, d, {0, 3, 1}] = If[(d ≥ 4), 7560, 0];
nrBAW[9, d, {2, 2, 1}] = If[(d ≥ 5), 15120, 0];
nrBAW[9, d, {4, 1, 1}] = If[(d ≥ 6), 30240, 0];
nrBAW[9, d, {6, 0, 1}] = If[(d ≥ 7), 60480, 0];
nrBAW[9, d, {1, 4}] = If[(d ≥ 5), 22680, 0];
nrBAW[9, d, {3, 3}] = If[(d ≥ 6), 45360, 0];
nrBAW[9, d, {5, 2}] = If[(d ≥ 7), 90720, 0];
nrBAW[9, d, {7, 1}] = If[(d ≥ 8), 181440, 0];;
nrBAW[9, d, {9}] = If[(d ≥ 9), 362880, 0];

(*n=10,SAW*)

nrSAW[10, d, {8}] = If[(d ≥ 8), 9031680 + 2903040 (d - 8), 0];
nrSAW[10, d, {6, 1}] = If[(d ≥ 7), 3669120 + 1451520 (d - 7), 0];
nrSAW[10, d, {6}] = If[(d ≥ 6), 5342400 + 5620320 (d - 6) + 1149120 (d - 6) (d - 7), 0];
nrSAW[10, d, {5, 0, 1}] = If[(d ≥ 6), 991200 + 483840 (d - 6), 0];
nrSAW[10, d, {4, 2}] = If[(d ≥ 6), 1411200 + 725760 (d - 6), 0];
nrSAW[10, d, {4, 1}] = If[(d ≥ 5), 1442040 + 2144880 (d - 5) + 574560 (d - 5) (d - 6), 0];
nrSAW[10, d, {4, 0, 0, 1}] = If[(d ≥ 5), 194880 + 120960 (d - 5), 0];
nrSAW[10, d, {3, 1, 1}] = If[(d ≥ 5), 354480 + 241920 (d - 5), 0];
nrSAW[10, d, {2, 3}] = If[(d ≥ 5), 493920 + 362880 (d - 5), 0];
nrSAW[10, d, {4}] =
  If[(d ≥ 4), 508608 + 1990944 (d - 4) + 1585920 (d - 4) (d - 5) + 297600 (d - 4) (d - 5) (d - 6),
   0];
nrSAW[10, d, {3, 0, 1}] = If[(d ≥ 4), 236760 + 533520 (d - 4) + 191520 (d - 4) (d - 5), 0];
nrSAW[10, d, {2, 2}] = If[(d ≥ 4), 297960 + 739800 (d - 4) + 287280 (d - 4) (d - 5), 0];
nrSAW[10, d, {3, 0, 0, 0, 1}] = If[(d ≥ 4), 28896 + 24192 (d - 4), 0];
nrSAW[10, d, {2, 1, 0, 1}] = If[(d ≥ 4), 62160 + 60480 (d - 4), 0];
```

```
nrSAW[10, d, {2, 0, 2}] = If[(d ≥ 4), 79 520 + 80 640 (d - 4), 0];
nrSAW[10, d, {1, 2, 1}] = If[(d ≥ 4), 106 680 + 120 960 (d - 4), 0];
nrSAW[10, d, {0, 4}] = If[(d ≥ 4), 141 120 + 181 440 (d - 4), 0];
nrSAW[10, d, {2, 1}] =
  If[(d ≥ 3), 41 524 + 392 436 (d - 3) + 538 080 (d - 3) (d - 4) + 148 800 (d - 3) (d - 4) (d - 5), 0];
nrSAW[10, d, {2, 0, 0, 1}] = If[(d ≥ 3), 22 104 + 92 052 (d - 3) + 47 880 (d - 3) (d - 4), 0];
nrSAW[10, d, {1, 1, 1}] = If[(d ≥ 3), 29 440 + 155 880 (d - 3) + 95 760 (d - 3) (d - 4), 0];
nrSAW[10, d, {0, 3}] = If[(d ≥ 3), 33 210 + 203 580 (d - 3) + 143 640 (d - 3) (d - 4), 0];
nrSAW[10, d, {2, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 3184 + 4032 (d - 3), 0];
nrSAW[10, d, {1, 1, 0, 0, 1}] = If[(d ≥ 3), 7392 + 12 096 (d - 3), 0];
nrSAW[10, d, {1, 0, 1, 1}] = If[(d ≥ 3), 11 060 + 20 160 (d - 3), 0];
nrSAW[10, d, {0, 2, 0, 1}] = If[(d ≥ 3), 13 440 + 30 240 (d - 3), 0];
nrSAW[10, d, {0, 1, 2}] = If[(d ≥ 3), 16 240 + 40 320 (d - 3), 0];
nrSAW[10, d, {2}] =
  If[(d ≥ 2), 396 + 38 580 (d - 2) + 223 560 (d - 2) (d - 3) + 236 928 (d - 2) (d - 3) (d - 4) +
    54 336 (d - 2) (d - 3) (d - 4) (d - 5), 0];
nrSAW[10, d, {1, 0, 1}] =
  If[(d ≥ 2), 598 + 35 344 (d - 2) + 110 360 (d - 2) (d - 3) + 49 600 (d - 2) (d - 3) (d - 4), 0];
nrSAW[10, d, {0, 2}] =
  If[(d ≥ 2), 568 + 38 700 (d - 2) + 141 600 (d - 2) (d - 3) + 74 400 (d - 2) (d - 3) (d - 4), 0];
nrSAW[10, d, {1, 0, 0, 0, 1}] = If[(d ≥ 2), 636 + 10 548 (d - 2) + 9576 (d - 2) (d - 3), 0];
nrSAW[10, d, {0, 1, 0, 1}] = If[(d ≥ 2), 661 + 18 306 (d - 2) + 23 940 (d - 2) (d - 3), 0];
nrSAW[10, d, {0, 0, 2}] = If[(d ≥ 2), 660 + 21 720 (d - 2) + 31 920 (d - 2) (d - 3), 0];
nrSAW[10, d, {1, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 226 + 576 (d - 2), 0];
nrSAW[10, d, {0, 1, 0, 0, 0, 1}] = If[(d ≥ 2), 416 + 2016 (d - 2), 0];
nrSAW[10, d, {0, 0, 1, 0, 1}] = If[(d ≥ 2), 532 + 4032 (d - 2), 0];
nrSAW[10, d, {0, 0, 0, 2}] = If[(d ≥ 2), 560 + 5040 (d - 2), 0];
nrSAW[10, d, {0, 1}] =
  If[(d ≥ 1), 468 (d - 1) + 19 904 (d - 1) (d - 2) + 59 392 (d - 1) (d - 2) (d - 3) +
    27 168 (d - 1) (d - 2) (d - 3) (d - 4), 0];
nrSAW[10, d, {0, 0, 0, 1}] =
  If[(d ≥ 1), 654 (d - 1) + 11 936 (d - 1) (d - 2) + 12 400 (d - 1) (d - 2) (d - 3), 0];
nrSAW[10, d, {0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 486 (d - 1) + 1596 (d - 1) (d - 2), 0];
nrSAW[10, d, {0, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 72 (d - 1), 0];
nrBAW[10, d, {0, 0, 0, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrSAW[10, d, {1, 0, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 10, 0];
nrSAW[10, d, {0, 1, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 45, 0];
nrSAW[10, d, {2, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 90, 0];
nrSAW[10, d, {0, 0, 1, 0, 0, 0, 1}] = If[(d ≥ 2), 120, 0];
nrSAW[10, d, {1, 1, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 360, 0];
nrSAW[10, d, {3, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 4), 720, 0];
nrSAW[10, d, {0, 0, 0, 1, 0, 1}] = If[(d ≥ 2), 210, 0];
nrSAW[10, d, {1, 0, 1, 0, 0, 1}] = If[(d ≥ 3), 840, 0];
```

```
nrSAW[10, d, {0, 2, 0, 0, 0, 1}] = If[(d ≥ 3), 1260, 0];
nrSAW[10, d, {2, 1, 0, 0, 0, 1}] = If[(d ≥ 4), 2520, 0];
nrSAW[10, d, {4, 0, 0, 0, 0, 1}] = If[(d ≥ 5), 5040, 0];
nrSAW[10, d, {0, 0, 0, 0, 2}] = If[(d ≥ 2), 252, 0];
nrSAW[10, d, {1, 0, 0, 1, 1}] = If[(d ≥ 3), 1260, 0];
nrSAW[10, d, {0, 1, 1, 0, 1}] = If[(d ≥ 3), 2520, 0];
nrSAW[10, d, {2, 0, 1, 0, 1}] = If[(d ≥ 4), 5040, 0];
nrSAW[10, d, {1, 2, 0, 0, 1}] = If[(d ≥ 4), 7560, 0];
nrSAW[10, d, {3, 1, 0, 0, 1}] = If[(d ≥ 5), 15120, 0];
nrSAW[10, d, {5, 0, 0, 0, 1}] = If[(d ≥ 6), 30240, 0];
nrSAW[10, d, {0, 1, 0, 2}] = If[(d ≥ 3), 3150, 0];
nrSAW[10, d, {2, 0, 0, 2}] = If[(d ≥ 4), 6300, 0];
nrSAW[10, d, {0, 0, 2, 1}] = If[(d ≥ 3), 4200, 0];
nrSAW[10, d, {1, 1, 1, 1}] = If[(d ≥ 4), 12600, 0];
nrSAW[10, d, {3, 0, 1, 1}] = If[(d ≥ 5), 25200, 0];
nrSAW[10, d, {0, 3, 0, 1}] = If[(d ≥ 4), 18900, 0];
nrSAW[10, d, {2, 2, 0, 1}] = If[(d ≥ 5), 37800, 0];
nrSAW[10, d, {4, 1, 0, 1}] = If[(d ≥ 6), 75600, 0];
nrSAW[10, d, {6, 0, 0, 1}] = If[(d ≥ 7), 151200, 0];
nrSAW[10, d, {1, 0, 3}] = If[(d ≥ 4), 16800, 0];
nrSAW[10, d, {0, 2, 2}] = If[(d ≥ 4), 25200, 0];
nrSAW[10, d, {2, 1, 2}] = If[(d ≥ 5), 50400, 0];
nrSAW[10, d, {4, 0, 2}] = If[(d ≥ 6), 100800, 0];
nrSAW[10, d, {1, 3, 1}] = If[(d ≥ 5), 75600, 0];
nrSAW[10, d, {3, 2, 1}] = If[(d ≥ 6), 151200, 0];
nrSAW[10, d, {5, 1, 1}] = If[(d ≥ 7), 302400, 0];
nrSAW[10, d, {7, 0, 1}] = If[(d ≥ 8), 604800, 0];
nrSAW[10, d, {0, 5}] = If[(d ≥ 5), 113400, 0];
nrSAW[10, d, {2, 4}] = If[(d ≥ 6), 226800, 0];
nrSAW[10, d, {4, 3}] = If[(d ≥ 7), 453600, 0];
nrSAW[10, d, {6, 2}] = If[(d ≥ 8), 907200, 0];
nrSAW[10, d, {8, 1}] = If[(d ≥ 9), 1814400, 0];
nrSAW[10, d, {10}] = If[(d ≥ 10), 3628800, 0];
(*n=10,BAW*)
nrBAW[10, d, {8}] = If[(d ≥ 8), 9031680 + 2903040 (d - 8), 0];
nrBAW[10, d, {6, 1}] = If[(d ≥ 7), 3669120 + 1451520 (d - 7), 0];
nrBAW[10, d, {6}] = If[(d ≥ 6), 5781600 + 5827680 (d - 6) + 1169280 (d - 6) (d - 7), 0];
nrBAW[10, d, {5, 0, 1}] = If[(d ≥ 6), 991200 + 483840 (d - 6), 0];
nrBAW[10, d, {4, 2}] = If[(d ≥ 6), 1411200 + 725760 (d - 6), 0];
nrBAW[10, d, {4, 1}] = If[(d ≥ 5), 1577880 + 2228400 (d - 5) + 584640 (d - 5) (d - 6), 0];
nrBAW[10, d, {4, 0, 0, 1}] = If[(d ≥ 5), 194880 + 120960 (d - 5), 0];
nrBAW[10, d, {3, 1, 1}] = If[(d ≥ 5), 354480 + 241920 (d - 5), 0];
nrBAW[10, d, {2, 3}] = If[(d ≥ 5), 493920 + 362880 (d - 5), 0];
```

```
nrBAW[10, d, {4}] =
  If[(d ≥ 4), 683 904 + 2 377 728 (d - 4) + 1 753 632 (d - 4) (d - 5) + 314 880 (d - 4) (d - 5) (d - 6),
   0];
nrBAW[10, d, {3, 0, 1}] = If[(d ≥ 4), 260 760 + 554 640 (d - 4) + 194 880 (d - 4) (d - 5), 0];
nrBAW[10, d, {2, 2}] = If[(d ≥ 4), 334 080 + 771 480 (d - 4) + 292 320 (d - 4) (d - 5), 0];
nrBAW[10, d, {3, 0, 0, 0, 1}] = If[(d ≥ 4), 28 896 + 24 192 (d - 4), 0];
nrBAW[10, d, {2, 1, 0, 1}] = If[(d ≥ 4), 62 160 + 60 480 (d - 4), 0];
nrBAW[10, d, {2, 0, 2}] = If[(d ≥ 4), 79 520 + 80 640 (d - 4), 0];
nrBAW[10, d, {1, 2, 1}] = If[(d ≥ 4), 106 680 + 120 960 (d - 4), 0];
nrBAW[10, d, {0, 4}] = If[(d ≥ 4), 141 120 + 181 440 (d - 4), 0];
nrBAW[10, d, {2, 1}] =
  If[(d ≥ 3), 61 380 + 484 164 (d - 3) + 599 376 (d - 3) (d - 4) + 157 440 (d - 3) (d - 4) (d - 5), 0];
nrBAW[10, d, {2, 0, 0, 1}] = If[(d ≥ 3), 24 444 + 95 652 (d - 3) + 48 720 (d - 3) (d - 4), 0];
nrBAW[10, d, {1, 1, 1}] = If[(d ≥ 3), 34 200 + 163 080 (d - 3) + 97 440 (d - 3) (d - 4), 0];
nrBAW[10, d, {0, 3}] = If[(d ≥ 3), 40 410 + 214 380 (d - 3) + 146 160 (d - 3) (d - 4), 0];
nrBAW[10, d, {2, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 3184 + 4032 (d - 3), 0];
nrBAW[10, d, {1, 1, 0, 0, 1}] = If[(d ≥ 3), 7392 + 12 096 (d - 3), 0];
nrBAW[10, d, {1, 0, 1, 1}] = If[(d ≥ 3), 11 060 + 20 160 (d - 3), 0];
nrBAW[10, d, {0, 2, 0, 1}] = If[(d ≥ 3), 13 440 + 30 240 (d - 3), 0];
nrBAW[10, d, {0, 1, 2}] = If[(d ≥ 3), 16 240 + 40 320 (d - 3), 0];
nrBAW[10, d, {2}] =
  If[(d ≥ 2), 1052 + 74 060 (d - 2) + 339 144 (d - 2) (d - 3) + 305 520 (d - 2) (d - 3) (d - 4) +
    63 360 (d - 2) (d - 3) (d - 4) (d - 5), 0];
nrBAW[10, d, {1, 0, 1}] =
  If[(d ≥ 2), 1070 + 45 408 (d - 2) + 123 552 (d - 2) (d - 3) + 52 480 (d - 2) (d - 3) (d - 4), 0];
nrBAW[10, d, {0, 2}] =
  If[(d ≥ 2), 1048 + 52 932 (d - 2) + 160 968 (d - 2) (d - 3) + 78 720 (d - 2) (d - 3) (d - 4), 0];
nrBAW[10, d, {1, 0, 0, 0, 1}] = If[(d ≥ 2), 704 + 10 932 (d - 2) + 9744 (d - 2) (d - 3), 0];
nrBAW[10, d, {0, 1, 0, 1}] = If[(d ≥ 2), 861 + 19 266 (d - 2) + 24 360 (d - 2) (d - 3), 0];
nrBAW[10, d, {0, 0, 2}] = If[(d ≥ 2), 940 + 23 000 (d - 2) + 32 480 (d - 2) (d - 3), 0];
nrBAW[10, d, {1, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 226 + 576 (d - 2), 0];
nrBAW[10, d, {0, 1, 0, 0, 0, 1}] = If[(d ≥ 2), 416 + 2016 (d - 2), 0];
nrBAW[10, d, {0, 0, 1, 0, 1}] = If[(d ≥ 2), 532 + 4032 (d - 2), 0];
nrBAW[10, d, {0, 0, 0, 2}] = If[(d ≥ 2), 560 + 5040 (d - 2), 0];
nrBAW[10, d, {0, 1}] =
  If[(d ≥ 1), 1042 (d - 1) + 33 076 (d - 1) (d - 2) + 78 520 (d - 1) (d - 2) (d - 3) +
    31 680 (d - 1) (d - 2) (d - 3) (d - 4), 0];
nrBAW[10, d, {0, 0, 0, 1}] =
  If[(d ≥ 1), 954 (d - 1) + 13 452 (d - 1) (d - 2) + 13 120 (d - 1) (d - 2) (d - 3), 0];
nrBAW[10, d, {0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 494 (d - 1) + 1624 (d - 1) (d - 2), 0];
nrBAW[10, d, {0, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 72 (d - 1), 0];
nrBAW[10, d, {0}] =
  If[(d ≥ 0), 440 d (d - 1) + 11 680 d (d - 1) (d - 2) + 24 720 d (d - 1) (d - 2) (d - 3) +
```

```
    9216 d (d - 1) (d - 2) (d - 3) (d - 4), 0];
nrBAW[10, d, {0, 0, 0, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 1), 1, 0];
nrBAW[10, d, {1, 0, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 10, 0];
nrBAW[10, d, {0, 1, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 2), 45, 0];
nrBAW[10, d, {2, 0, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 90, 0];
nrBAW[10, d, {0, 0, 1, 0, 0, 0, 1}] = If[(d ≥ 2), 120, 0];
nrBAW[10, d, {1, 1, 0, 0, 0, 0, 1}] = If[(d ≥ 3), 360, 0];
nrBAW[10, d, {3, 0, 0, 0, 0, 0, 1}] = If[(d ≥ 4), 720, 0];
nrBAW[10, d, {0, 0, 0, 1, 0, 1}] = If[(d ≥ 2), 210, 0];
nrBAW[10, d, {1, 0, 1, 0, 0, 1}] = If[(d ≥ 3), 840, 0];
nrBAW[10, d, {0, 2, 0, 0, 0, 1}] = If[(d ≥ 3), 1260, 0];
nrBAW[10, d, {2, 1, 0, 0, 0, 1}] = If[(d ≥ 4), 2520, 0];
nrBAW[10, d, {4, 0, 0, 0, 0, 1}] = If[(d ≥ 5), 5040, 0];
nrBAW[10, d, {0, 0, 0, 0, 2}] = If[(d ≥ 2), 252, 0];
nrBAW[10, d, {1, 0, 0, 1, 1}] = If[(d ≥ 3), 1260, 0];
nrBAW[10, d, {0, 1, 1, 0, 1}] = If[(d ≥ 3), 2520, 0];
nrBAW[10, d, {2, 0, 1, 0, 1}] = If[(d ≥ 4), 5040, 0];
nrBAW[10, d, {1, 2, 0, 0, 1}] = If[(d ≥ 4), 7560, 0];
nrBAW[10, d, {3, 1, 0, 0, 1}] = If[(d ≥ 5), 15 120, 0];
nrBAW[10, d, {5, 0, 0, 0, 1}] = If[(d ≥ 6), 30 240, 0];
nrBAW[10, d, {0, 1, 0, 2}] = If[(d ≥ 3), 3150, 0];
nrBAW[10, d, {2, 0, 0, 2}] = If[(d ≥ 4), 6300, 0];
nrBAW[10, d, {0, 0, 2, 1}] = If[(d ≥ 3), 4200, 0];
nrBAW[10, d, {1, 1, 1, 1}] = If[(d ≥ 4), 12 600, 0];
nrBAW[10, d, {3, 0, 1, 1}] = If[(d ≥ 5), 25 200, 0];
nrBAW[10, d, {0, 3, 0, 1}] = If[(d ≥ 4), 18 900, 0];
nrBAW[10, d, {2, 2, 0, 1}] = If[(d ≥ 5), 37 800, 0];
nrBAW[10, d, {4, 1, 0, 1}] = If[(d ≥ 6), 75 600, 0];
nrBAW[10, d, {6, 0, 0, 1}] = If[(d ≥ 7), 151 200, 0];
nrBAW[10, d, {1, 0, 3}] = If[(d ≥ 4), 16 800, 0];
nrBAW[10, d, {0, 2, 2}] = If[(d ≥ 4), 25 200, 0];
nrBAW[10, d, {2, 1, 2}] = If[(d ≥ 5), 50 400, 0];
nrBAW[10, d, {4, 0, 2}] = If[(d ≥ 6), 100 800, 0];
nrBAW[10, d, {1, 3, 1}] = If[(d ≥ 5), 75 600, 0];
nrBAW[10, d, {3, 2, 1}] = If[(d ≥ 6), 151 200, 0];
nrBAW[10, d, {5, 1, 1}] = If[(d ≥ 7), 302 400, 0];
nrBAW[10, d, {7, 0, 1}] = If[(d ≥ 8), 604 800, 0];
nrBAW[10, d, {0, 5}] = If[(d ≥ 5), 113 400, 0];
nrBAW[10, d, {2, 4}] = If[(d ≥ 6), 226 800, 0];
nrBAW[10, d, {4, 3}] = If[(d ≥ 7), 453 600, 0];
nrBAW[10, d, {6, 2}] = If[(d ≥ 8), 907 200, 0];
nrBAW[10, d, {8, 1}] = If[(d ≥ 9), 1 814 400, 0];
nrBAW[10, d, {10}] = If[(d ≥ 10), 3 628 800, 0];
```