Enumeration of self-avoiding walks via the lace expansion

Nathan Clisby (University of Melbourne)
Richard Liang (UC Berkeley)
Gordon Slade (UBC)

December 19, 2006

The self-avoiding walk model

- A self-avoiding walk (SAW) is a path on a lattice, which starts at the origin and hops successively to neighbouring lattice sites without intersecting itself.
- We count the number of SAWs of length n, c_n, and in particular study the critical behaviour of the generating function
$$C(x) = \sum_{n=0}^{\infty} c_n x^n$$
- For the simple cubic lattice $c_0 = 1$, $c_1 = 6$, $c_2 = 30$, $c_3 = 150$, $c_4 = 726$, $c_5 = 3534$, $\ldots$

Known results

- For the square lattice c_n has been enumerated to very high order via the finite lattice method by Iwan Jensen

$$c_{71} = 4190893020903935054619120005916$$

- Best results for $d > 2$ are via direct enumeration.
- MacDonald et al, simple cubic lattice

$$c_{26} = 549493796867100942$$

Known results

- Chen and Lin (2003), hypercubic lattice, $d = 4$

$$c_{19} = 8639846411760440$$

- Chen and Lin (2003), hypercubic lattice, $d = 5$

$$c_{15} = 192003889675210$$

- Chen and Lin (2003), hypercubic lattice, $d = 6$

$$c_{14} = 373292253262692$$

Known results

- It is universally believed (but not proven) that for dimensions $d < 4$ that

$$c_n = \mathrm{A}\, n^{\gamma-1} \mu^n \, [1 + \mathrm{corrections}]$$

- Improved enumerations allow better estimation of A, μ and γ.
- Trivial upper bound from forbidding immediate return, lower bound from walks with steps only in positive directions:

$$\begin{aligned} d^n &\le c_n \le 2d(2d-1)^{n-1} \\ \Rightarrow d &\le \mu \le 2d-1 \end{aligned}$$

Known results, $d = 2, 3$

- Square lattice $\mu = 2.6381\cdots$, $\gamma = 1.34375 = 43/32$ (exact value from association with $\mathrm{SLE}_{8/3}$).
- Cubic lattice, no exact results available, $\mu = 4.68404(9)$, $\gamma = 1.1575(6)$.

Known results

- For $d = 4$ there is believed to be a logarithmic factor

$$c_n = \mathrm{A}\,(\log n)^{1/4}\mu^n\,[1 + \mathrm{corrections}]$$

- For $d > 4$ it has been rigorously shown that $\gamma = 1$, and that

$$c_n = \mathrm{A}\;\mu^n\,[1 + \mathrm{corrections}]$$

Known results

- $1/d$ expansion for the connective constant

$$\mu = 2d - 1 - \frac{1}{2d} - \frac{3}{(2d)^2} - \frac{16}{(2d)^3} - \frac{102}{(2d)^4} + \cdots$$

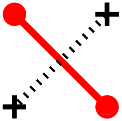
The two-step method

- Wish to reduce the time taken by reducing the complexity.
- Finite lattice method with pruning for SAPs on the square lattice, $\mu = 2.638\cdots$ but complexity is about 1.2!
- Idea is to take two-steps at once, and represent walks which have the same endpoint by a single configuration.
- We refer to the sequence of endpoints as a two-step walk, Ω.
- Will now show an example of how to generate a two-step walk.

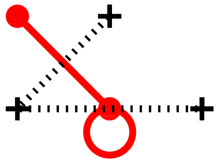
Two step method for SAWs

$+$

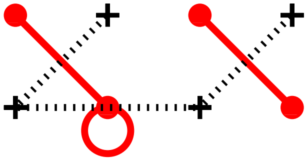
Two step method for SAWs



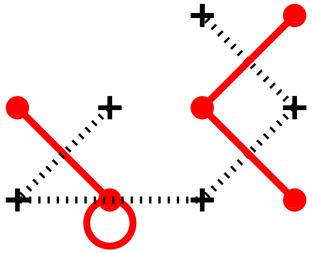
Two step method for SAWs



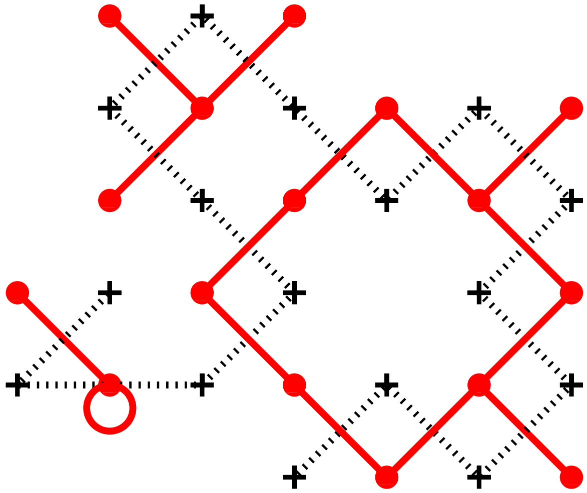
Two step method for SAWs



Two step method for SAWs



Two step method for SAWs



The two-step method

- We define:
  - $\mathcal{C}_\Omega$ as the set of connected components with one cycle.
  - $\mathcal{T}_\Omega$ as the set of trees.
  - If there is a component with more than 1 loop/cycle, the indicator function $I_\Omega = 0$, otherwise it is 1.
- The weight of a two-step walk Ω is then given by

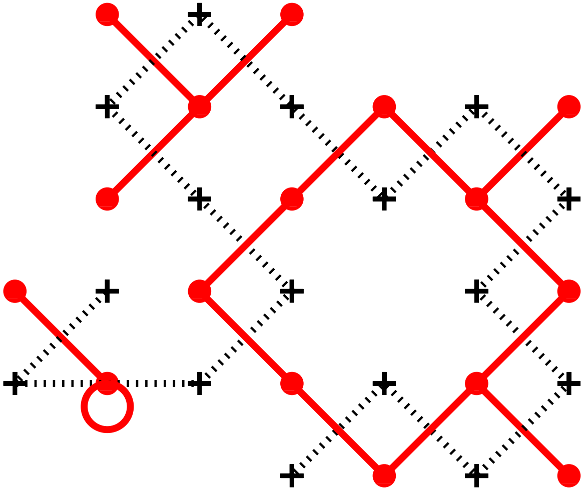
$$W(\Omega) = I_\Omega 2^{|\mathcal{C}_\Omega|} \prod_{T \in \mathcal{T}_\Omega} N_T$$

- Weight can be calculated in linear time in the size of the allocation graph.

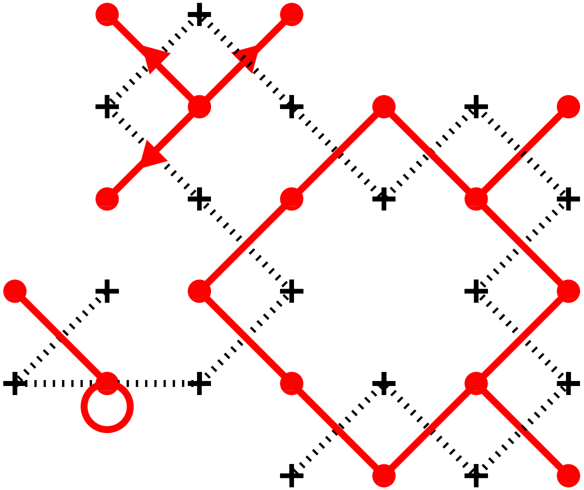
The two-step method, proof by example

- The weight of a two-step walk is the number of admissible orientations of it’s allocation graph.
- We assign directions to each edge in the graph, and an admissible orientation has *in-degree* at most one for each vertex.
- This rule guarantees that no site is visited more than once.

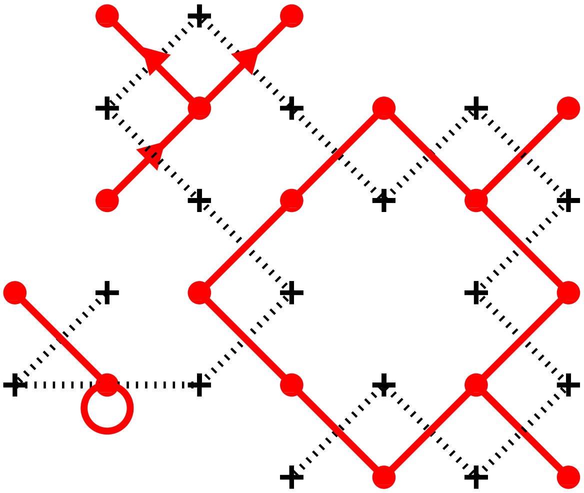
Two step method for SAWs



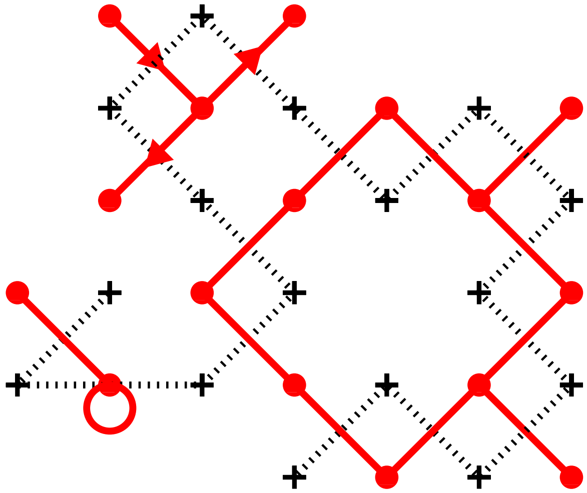
Two step method for SAWs



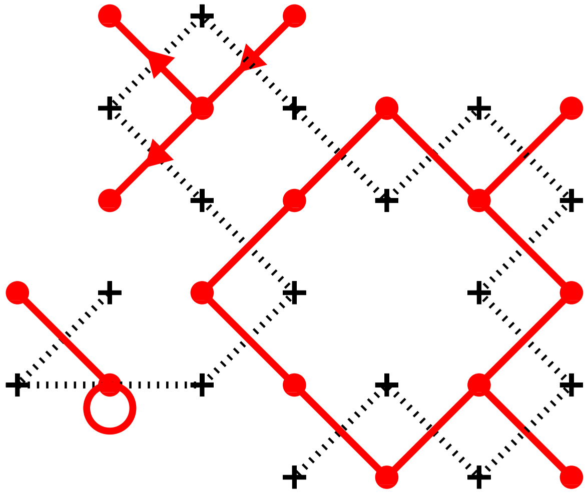
Two step method for SAWs



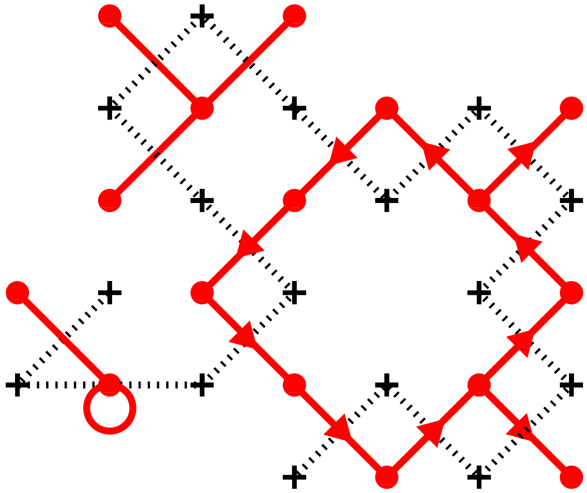
Two step method for SAWs



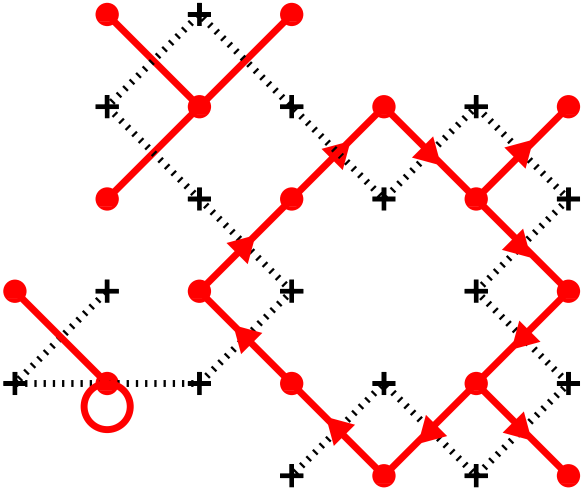
Two step method for SAWs



Two step method for SAWs



Two step method for SAWs



The two-step method

- We can calculate an upper bound for the complexity in the same way as the trivial upper bound for the connective constant, where immediate returns are forbidden.
- If S is the number of sites reachable by a self-avoiding walk in two-steps, the total number of configurations generated by taking l two-steps has an upper bound of

  $$S(S-1)^{l-1} = S(S-1)^{(n-2)/2}$$

  where $n = 2l$ is the number of individual steps.
- Therefore $\lambda \leq \sqrt{S-1}$.

The two-step method

- $S = 8$ for $d = 2$, and therefore the complexity has an upper bound of
$$\lambda \leq \sqrt{7} = 2.645\cdots$$
- The self-avoidance constraint reduces λ to a value of order 2.4 for $d = 2$.
- $S = 18$ for $d = 3$, and therefore the complexity has an upper bound of
$$\lambda \leq \sqrt{17} = 4.123\cdots < \mu$$
- The self-avoidance constraint reduces λ to something of order 4.0 for $d = 3$.

The lace expansion

- The lace expansion, originally due to Brydges and Spencer, is a method that has been used to study the critical behavior of SAWs, lattice trees and animals, percolation and related models, above their critical dimension.
- The number of SAWs of length n may be obtained from the following recursion relation,

$$c_n = 2dc_{n-1} + \sum_{m=2}^{n} \pi_m c_{n-m}$$

where π_m is the sum over all *connected* graphs of length m with weights ± 1 depending on the number of self-intersections.

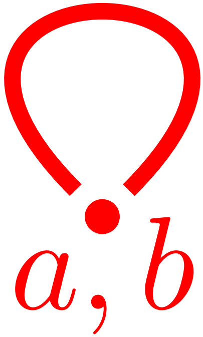
The lace expansion

- The lace expansion is a resummation of this connected graph expansion, and allows us to express π_m in terms of the number of lace graphs of length m with N loops:

$$\pi_m = \sum_N (-1)^N \pi_m^{(N)}$$

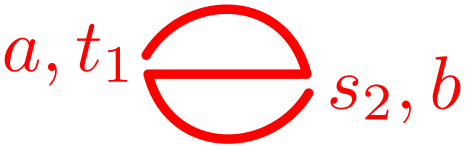
- Lace graphs are less numerous, and therefore easier to count!
- The first of these graphs are paths that avoid themselves until they return to the origin, i.e. graphs which form a single loop. Then there are graphs with $2, 3, 4, \ldots$ loops, which are represented by the following diagrams.

$\pi^{(1)}$



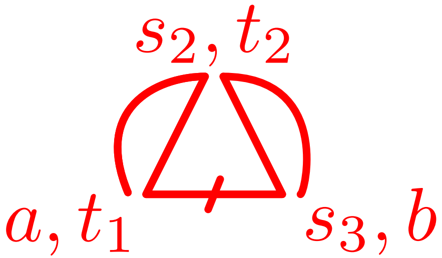
- Start at the origin, must return to the origin.
- Avoidance pattern
  [1]

$\pi^{(2)}$



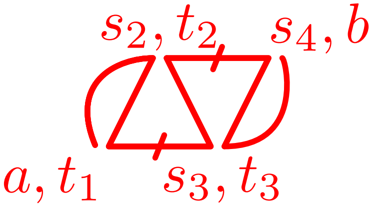
- Start at the origin, return to the origin, continue until the first loop is intersected.
- Avoidance pattern $[1, 2, 3]$

$\pi^{(3)}$



- Avoidance pattern $[1, 2, 3, 4]$, $[3, 4, 5]$

$\pi^{(4)}$

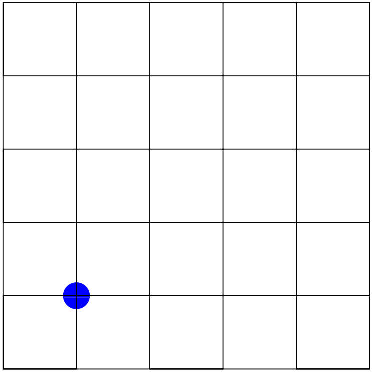


- Avoidance pattern $[1,2,3,4]$, $[3,4,5,6]$, $[5,6,7]$

Backtracking

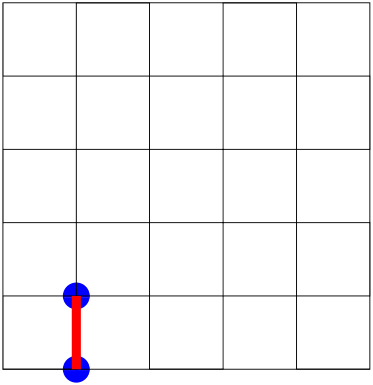
- Will now demonstrate how to use a backtracking algorithm to count lace graphs.
- Surprisingly simple recursive procedure.

Generating lace expansion graphs



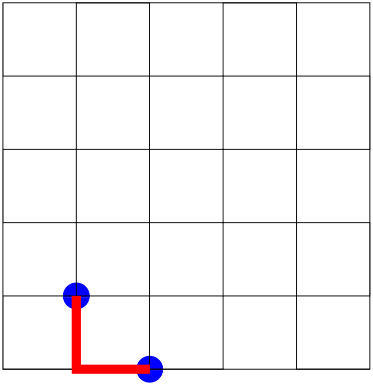
- Start at the origin, must return to the origin.
- $N = 0$
- $m = 0$

Generating lace expansion graphs



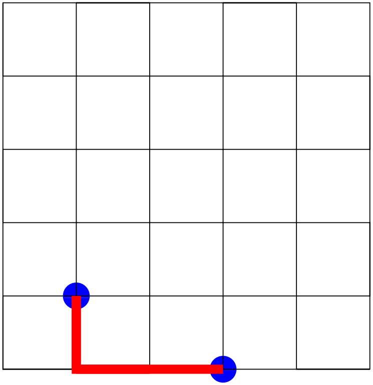
- Start at the origin, must return to the origin.
- $N = 0$
- $m = 1$

Generating lace expansion graphs



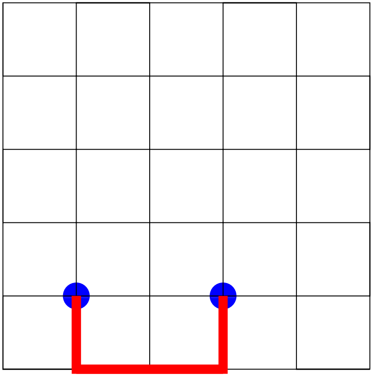
- Start at the origin, must return to the origin.
- $N = 0$
- $m = 2$

Generating lace expansion graphs



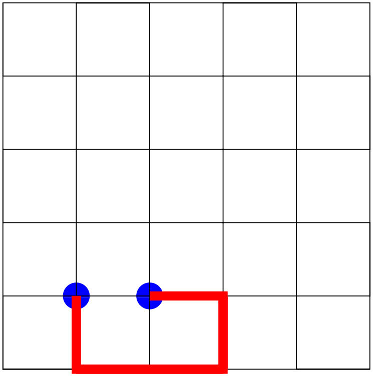
- Start at the origin, must return to the origin.
- $N = 0$
- $m = 3$

Generating lace expansion graphs



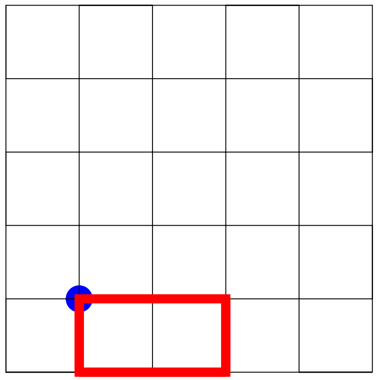
- Start at the origin, must return to the origin.
- $N = 0$
- $m = 4$

Generating lace expansion graphs



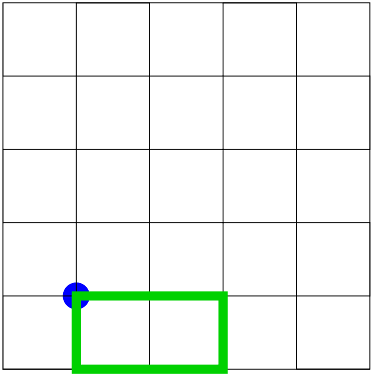
- Start at the origin, must return to the origin.
- $N = 0$
- $m = 5$

Generating lace expansion graphs



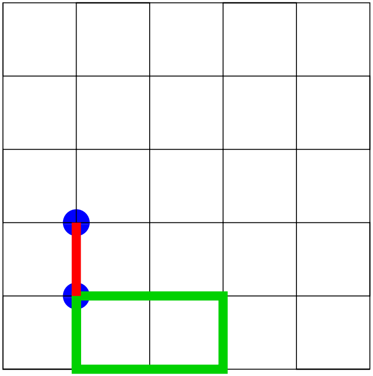
- Loop formed!
- Increment $\pi_6^{(1)}$.
- $N = 0 + 1 = 1$
- $m = 6$

Generating lace expansion graphs



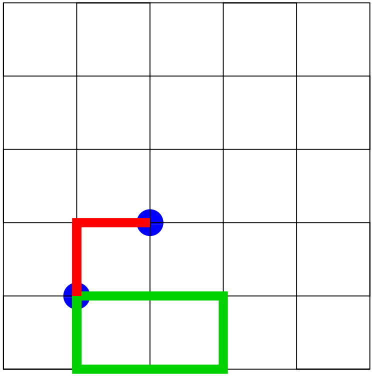
- Step forward until previous sub-walk is encountered.
- $N = 1$
- $m = 6$

Generating lace expansion graphs



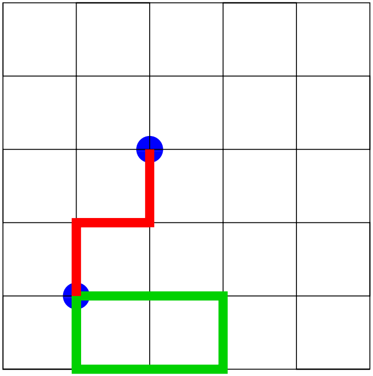
- Step forward until previous sub-walk is encountered.
- $N = 1$
- $m = 7$

Generating lace expansion graphs



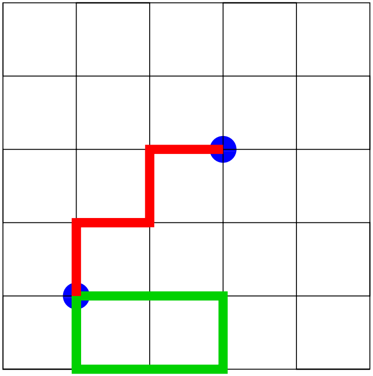
- Step forward until previous sub-walk is encountered.
- $N = 1$
- $m = 8$

Generating lace expansion graphs



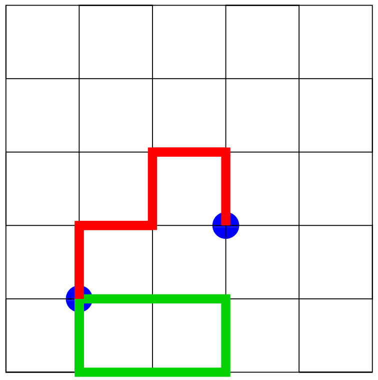
- Step forward until previous sub-walk is encountered.
- $N = 1$
- $m = 9$

Generating lace expansion graphs



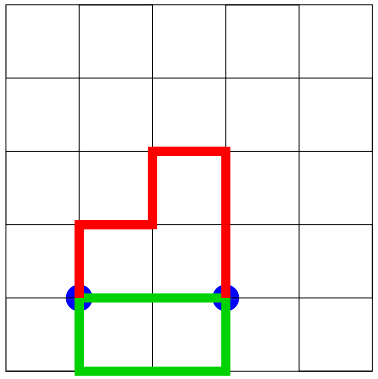
- Step forward until previous sub-walk is encountered.
- $N = 1$
- $m = 10$

Generating lace expansion graphs



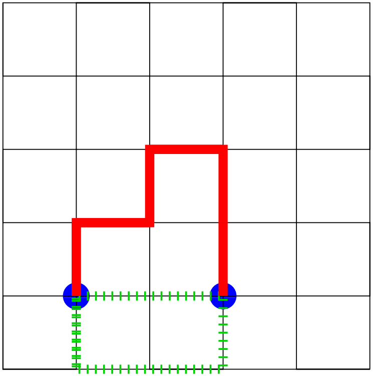
- Step forward until previous sub-walk is encountered.
- $N = 1$
- $m = 11$

Generating lace expansion graphs



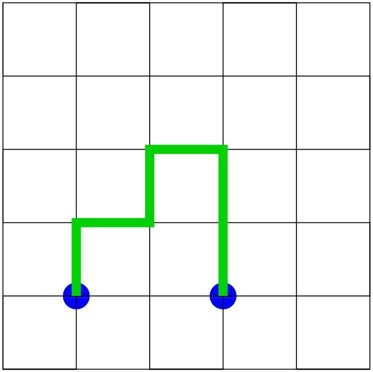
- Loop formed!
- Increment $\pi_{12}^{(2)}$.
- $N = 1 + 1 = 2$
- $m = 12$

Generating lace expansion graphs



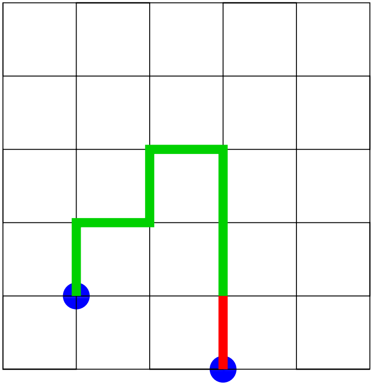
- Forget about the previous sub-walk.
- $N = 2$
- $m = 12$

Generating lace expansion graphs



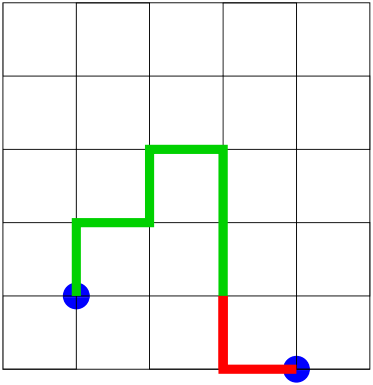
- Step forward until previous sub-walk is encountered.
- $N = 2$
- $m = 12$

Generating lace expansion graphs



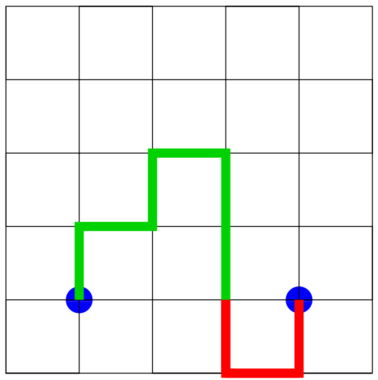
- Step forward until previous sub-walk is encountered.
- $N = 2$
- $m = 13$

Generating lace expansion graphs



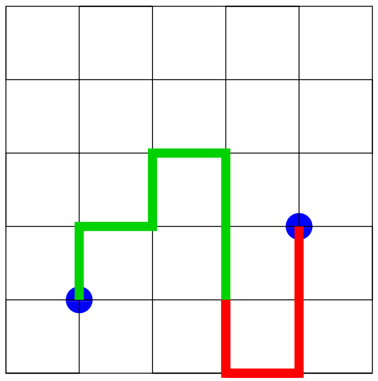
- Step forward until previous sub-walk is encountered.
- $N = 2$
- $m = 14$

Generating lace expansion graphs



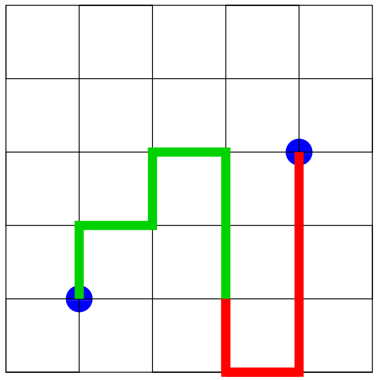
- Step forward until previous sub-walk is encountered.
- $N = 2$
- $m = 15$

Generating lace expansion graphs



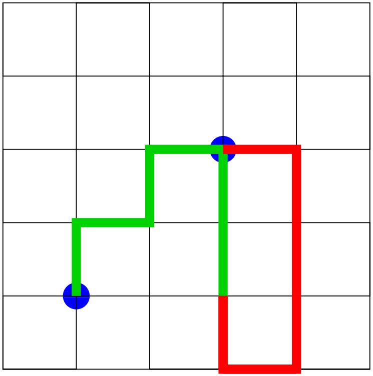
- Step forward until previous sub-walk is encountered.
- $N = 2$
- $m = 16$

Generating lace expansion graphs



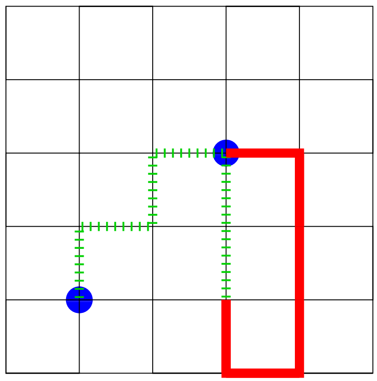
- Step forward until previous sub-walk is encountered.
- $N = 2$
- $m = 17$

Generating lace expansion graphs



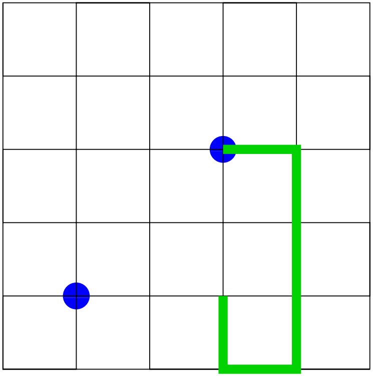
- Loop formed!
- Increment $\pi_{18}^{(3)}$.
- $N = 2 + 1 = 3$
- $m = 18$

Generating lace expansion graphs



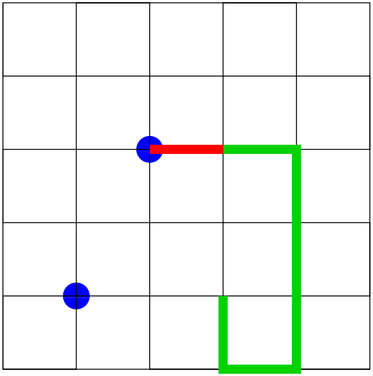
- Forget about the previous sub-walk.
- $N = 3$
- $m = 18$

Generating lace expansion graphs



- Step forward until previous sub-walk is encountered.
- $N = 3$
- $m = 18$

Generating lace expansion graphs



- Step forward until previous sub-walk is encountered.
- $N = 3$
- $m = 19$
- etc.

Backtracking details

- Similar to SAW backtracking, but with additional overhead.
- Choice of data structure is very important in order to do this efficiently.

Why are there fewer lace graphs?

- Lace graphs can be thought of as generalised polygons, there are less of them because they are less spatially extended than SAWs.
- But only by a polynomial factor! Algorithmic complexity is unchanged, and is still given by the connective constant μ. i.e. for any N
$$\pi_m^{(N)} \sim \mu^m$$
- For $d = 3$, $n = 30$,
$$c_{30} \approx 525\pi_{30}$$

Enumeration results

- Simple cubic lattice

$$\begin{aligned} c_{30} &= 270569905525454674614 \\ c_{26} &= 549493796867100942 \\ c_{30}/c_{26} &= 492.3\cdots \end{aligned}$$

- Hypercubic lattice, $d = 4$

$$\begin{aligned} c_{24} &= 124852857467211187784 \\ c_{19} &= 863984411760440 \\ c_{24}/c_{19} &= 14450.8\cdots \end{aligned}$$

Enumeration results

- Hypercubic lattice, $d = 5$

$$\begin{aligned} c_{24} &= 6374252557029958121090 \\ c_{15} &= 192003889675210 \\ c_{24}/c_{15} &= 3.3 \times 10^{8} \end{aligned}$$

- Hypercubic lattice, $d = 6$

$$\begin{aligned} c_{24} &= 8689265092167904101731532 \\ c_{14} &= 373292253262692 \\ c_{24}/c_{14} &= 2.3 \times 10^{10} \end{aligned}$$

- Our enumerations for $n = 24$ allow us to calculate c_{24} for any dimension.

Enumeration results

- Mean square end-to-end distance series calculated to the same order.
- SAPs enumerated to the same order, except for p_{32} for $d = 3$ and p_{26} for $d = 4$.
- For $d = 3$ used around 15000 CPU hours on VPAC (probably comparable to time used by MacDonald et al.)
- For $d > 3$ used around 5000 CPU hours on VPAC.

$1/d$ expansion for μ

- $1/d$ expansion for the connective constant, with error estimate,

$$\begin{aligned}\mu &= 2d - 1 - \frac{1}{2d} - \frac{3}{(2d)^2} - \frac{16}{(2d)^3} - \frac{102}{(2d)^4} \\ &\quad - \frac{729}{(2d)^5} - \frac{5533}{(2d)^6} - \frac{42229}{(2d)^7} - \frac{288761}{(2d)^8} \\ &\quad - \frac{1026328}{(2d)^9} + \frac{21070667}{(2d)^{10}} + \frac{780280468}{(2d)^{11}} + O\left(\frac{1}{(2d)^{12}}\right)\end{aligned}$$

- Last two terms in the series are positive.

Analysing the series

- At this stage, expect to slightly improve estimates for γ and μ for $d = 3$.
- Will improve upon existing series results for $d \geq 4$, estimates of μ will be competitive with those obtained via the PERM algorithm by Owczarek and Prellberg.

The k-step method?

- Is it possible to extend the two-step method to k-step?
- Potentially much faster, because improvement comes from μ^k walks being replaced by $O(k^d)$ k-step walks ($d \equiv \mathrm{dimension}$), one for each reachable end site.
- Can map the problem of updating weight factor when overlaps between different subwalks occur to the enumeration of maximum independent sets in a graph.
- The maximum independent set problem for general graphs is NP-complete, and it appears that this is likely to be the case for the graphs produced by the k-step method with $k > 2$.

Other applications

- Monte-Carlo (two-step).
- SAWs on other lattices (two-step and lace expansion).
- Polymers near a boundary (two-step).
- Self-avoiding trails (two-step).
- Enumeration versions of the travelling salesman and hamiltonian path problems (two-step).