Fast algorithms for self-avoiding walks

Nathan Clisby (MASCOS, University of Melbourne)

November 29, 2007

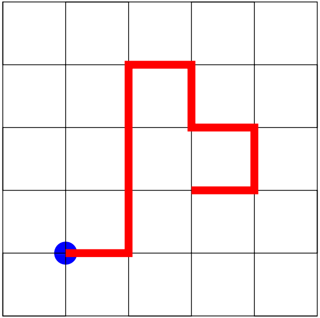
- A self-avoiding walk (SAW) is a path on a lattice, which starts at the origin and hops successively to neighbouring lattice sites without intersecting itself.
- Model of polymers (long chain molecules), where the avoidance constraint acts in the same way as excluded volume when studying the conformations available to a polymer.
- SAWs are the $N \to 0$ limit of the N-vector model, which is an important model in statistical mechanics and serves as a basic example in the theory of critical phenomena.

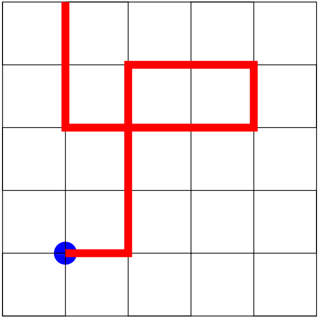
- Count the number of SAWs of length n, c_n.
- Study the critical behaviour of the generating function.

$$C(x) = \sum_{n=0}^{\infty} c_n x^n$$

- $$c_n \sim A\, n^{\gamma-1} \mu^n \,[1 + \text{corrections}]$$
- For the square lattice c_n has been enumerated to very high order via the finite lattice method by Iwan Jensen:

$$c_{71} = 4190893020903935054619120005916$$

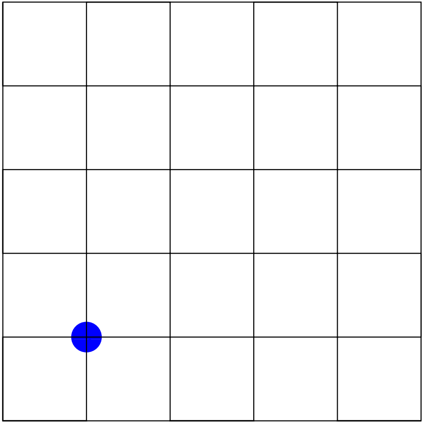


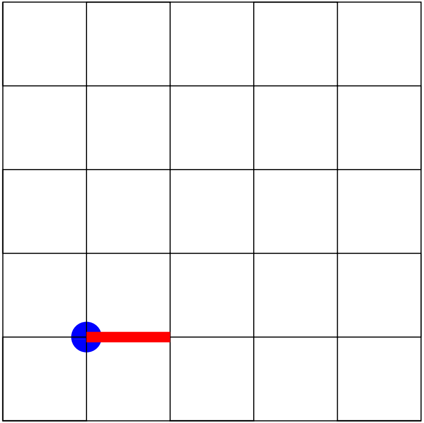


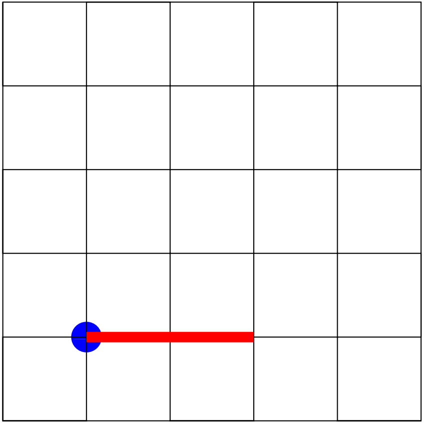
- SAW!
- Not a SAW, due to self intersection.

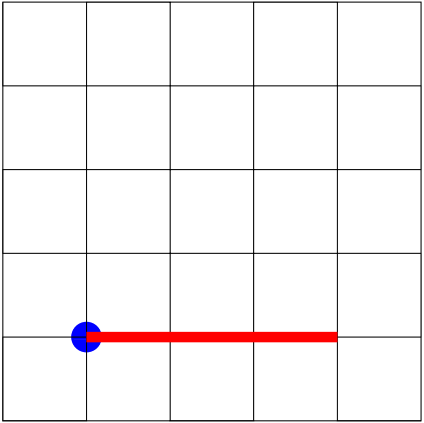
Brute force

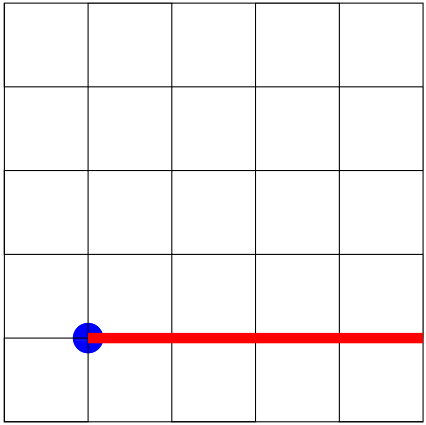
- Simplest method: brute force backtracking algorithm.
- Generate all possible SAWs recursively.

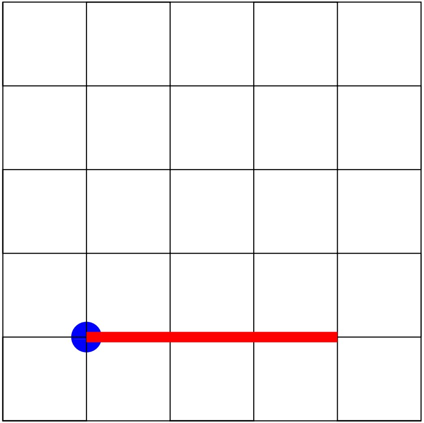


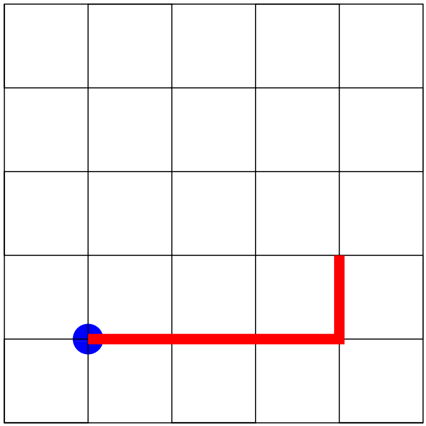


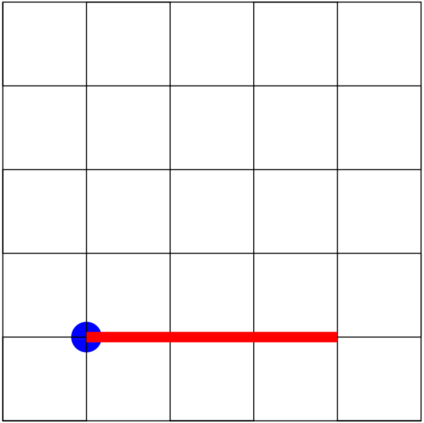


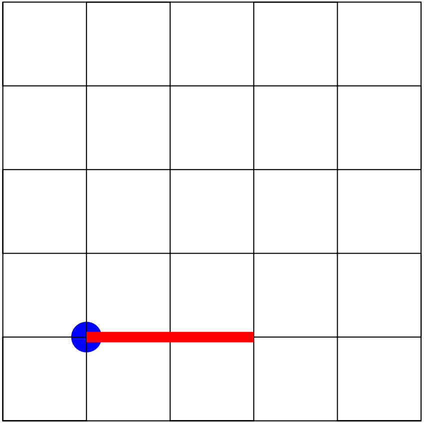


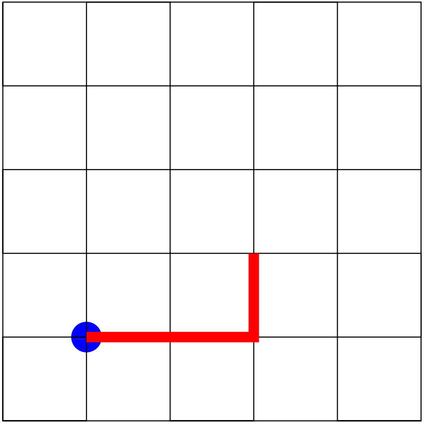


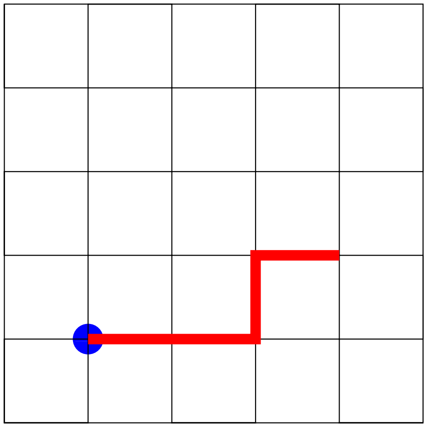


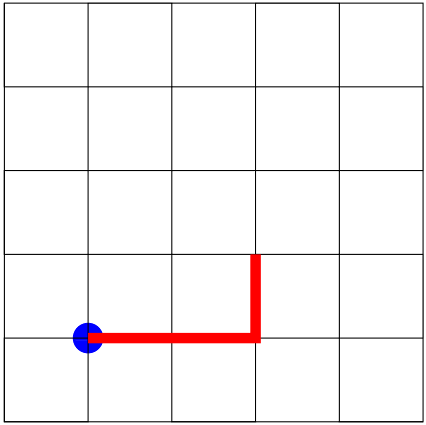


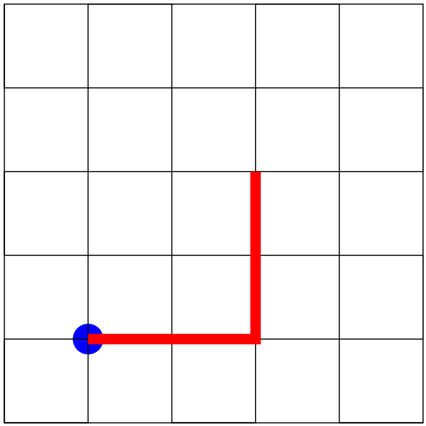


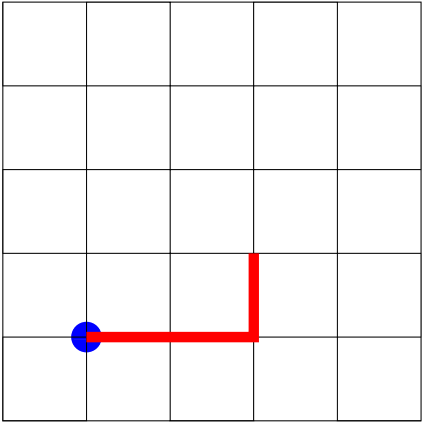


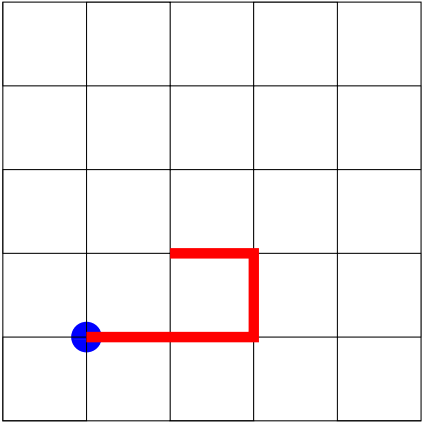


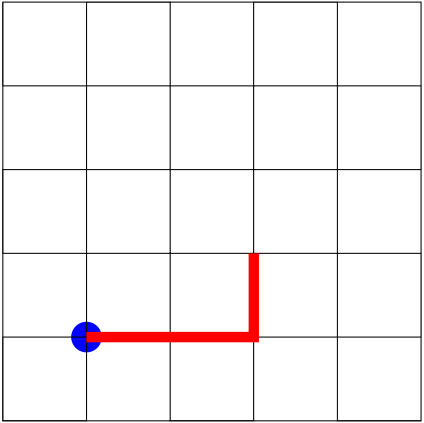


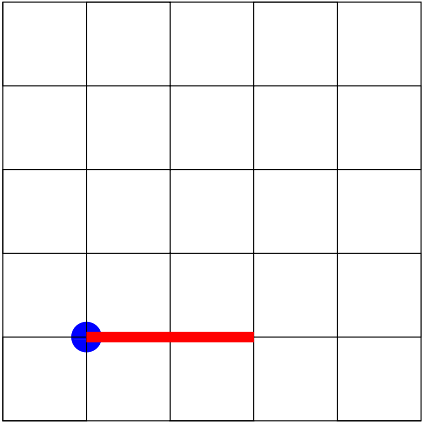


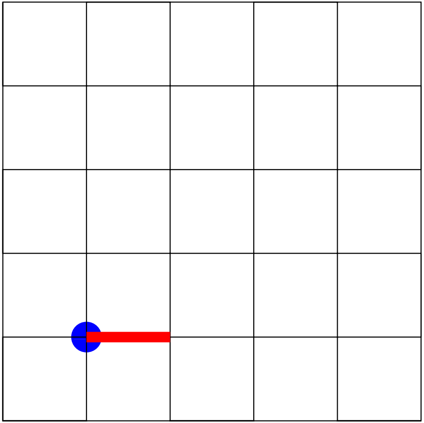


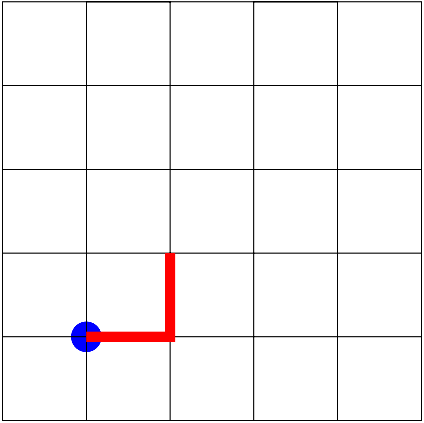


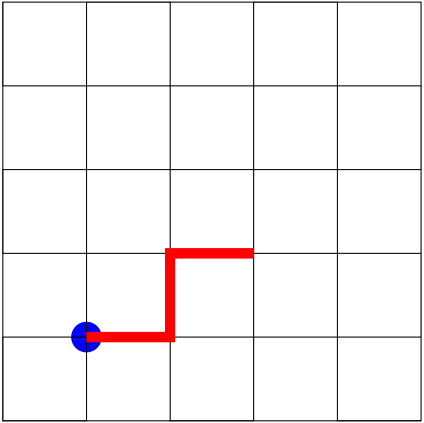


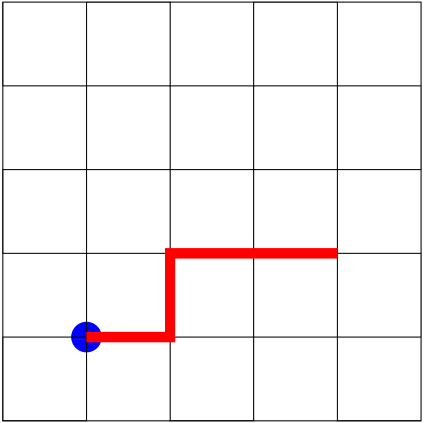


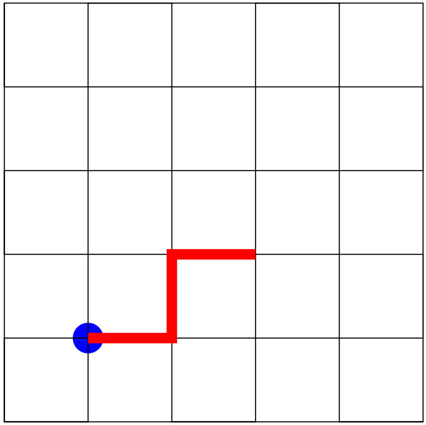












Other methods

- To improve algorithm performance, must either:

Other methods

- To improve algorithm performance, must either:
- Convert SAW enumeration to a simpler problem.
- e.g. lace expansion: enumerate polygons and other graphs with multiple loops. Fewer by a power law factor.

Other methods

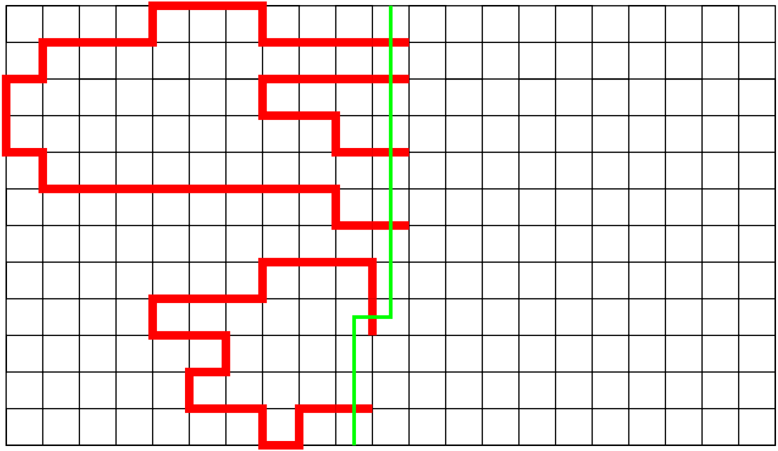
- To improve algorithm performance, must either:
- Convert SAW enumeration to a simpler problem.
- e.g. lace expansion: enumerate polygons and other graphs with multiple loops. Fewer by a power law factor.
- Count many SAWs simultaneously.
- e.g. 2-step and k-step methods which count an exponentially large number of SAWs simultaneously.

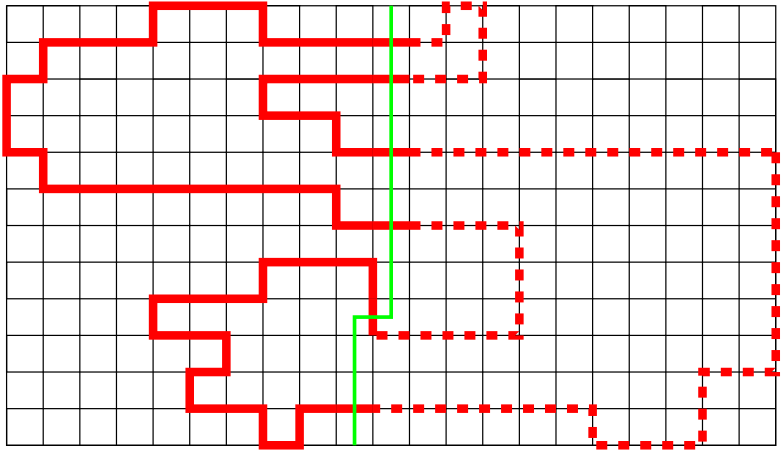
Other methods

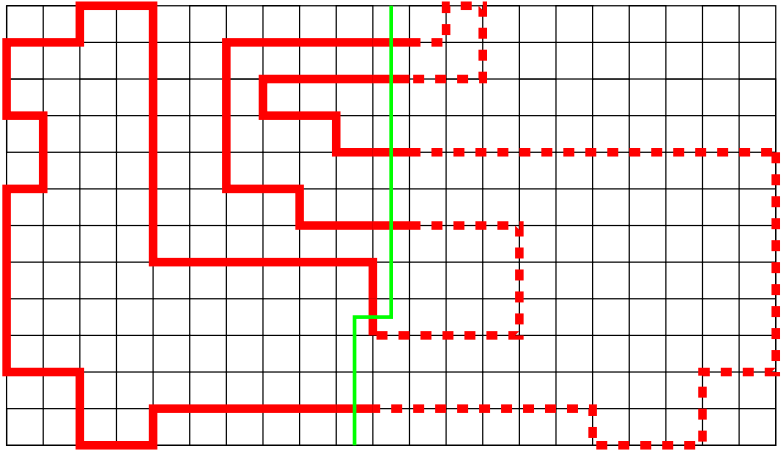
- To improve algorithm performance, must either:
- Convert SAW enumeration to a simpler problem.
- e.g. lace expansion: enumerate polygons and other graphs with multiple loops. Fewer by a power law factor.
- Count many SAWs simultaneously.
- e.g. 2-step and k-step methods which count an exponentially large number of SAWs simultaneously.
- Combination of lace expansion and 2-step currently gives best results for $d = 3$.

Finite lattice method

- Convert sum over all SAWs to a sum over all SAWs cross-sections.
- Possible because if we fix bonds on a boundary, including topological information, then the two sides of the boundary are *independent*.





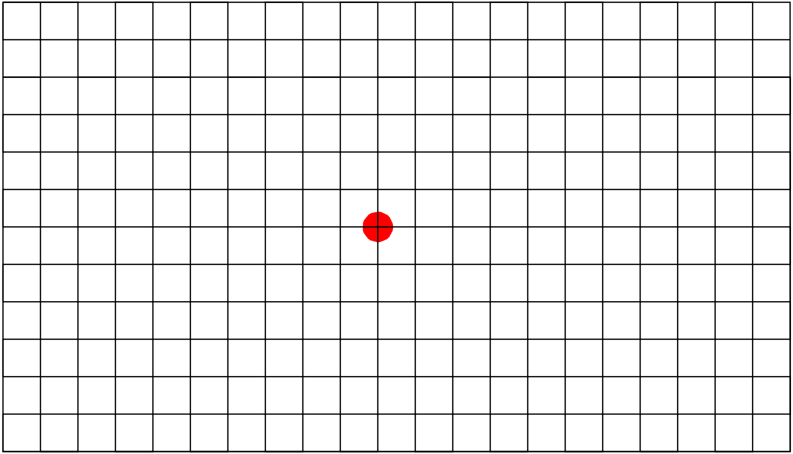


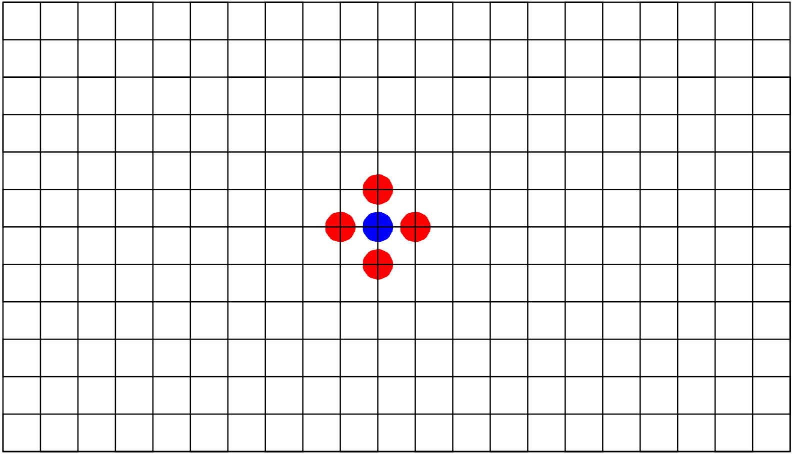
- As the RHS *only* depends on the boundary, we can count all SAWs by counting all SAWs on the LHS which correspond to a particular boundary.
- Then build up the LHS one site at a time by shifting the boundary.

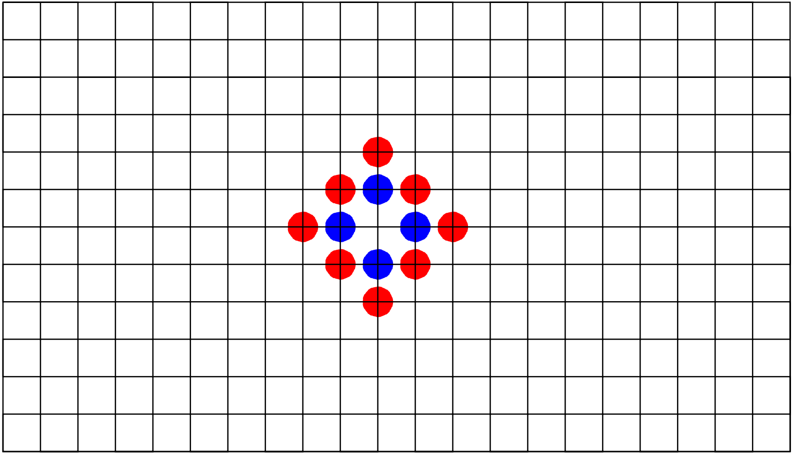
- Algorithmic behaviour of FLM determined by worst case, by boundary states with large number of occupied edges.
- There are many of these boundary states, and each of them represents relatively few SAW configurations.
- Up to $O(n)$ edges in boundary states ensure that algorithmic performance is κ^n.
- There are many fewer boundary states than SAWs.
- Dramatically more efficient than other known methods for $d = 2$.

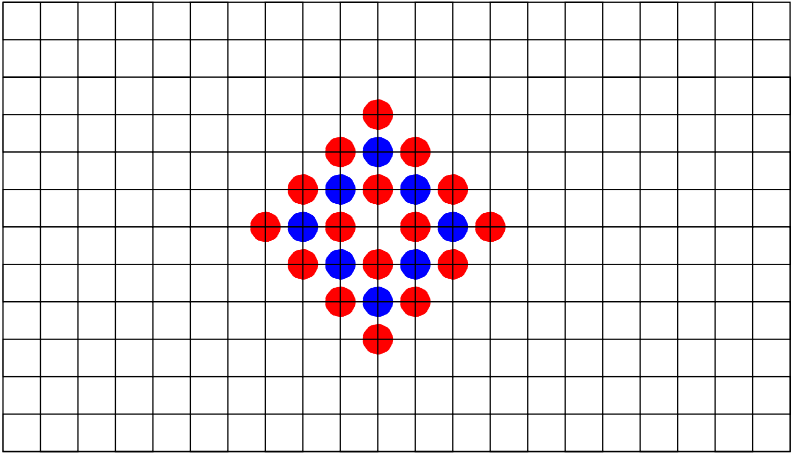
Geometric splitting

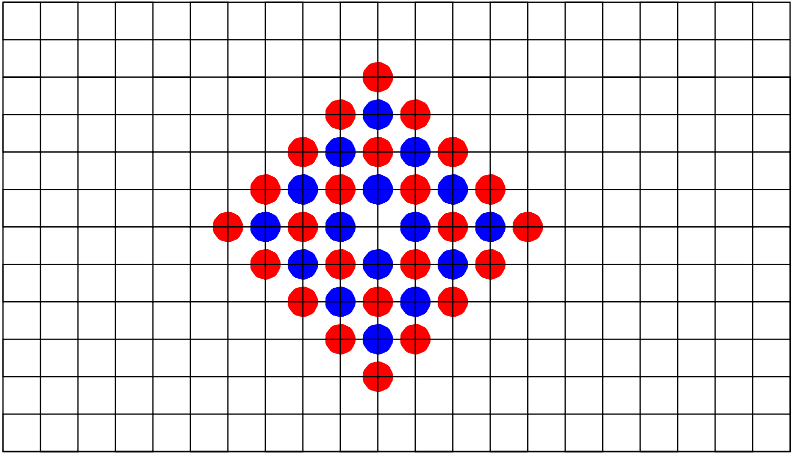
- Try to improve performance by *dynamically* splitting walk.
- Using geometric information allows one to choose an optimal boundary to split walk in half.
- Need a completely different representation of SAWs.

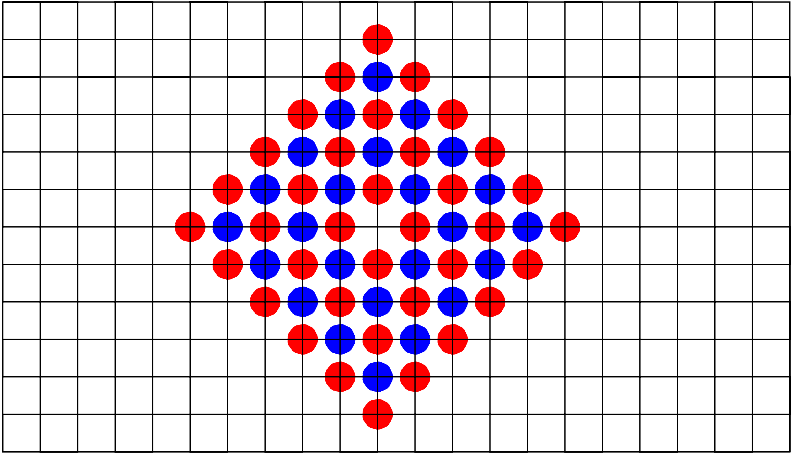




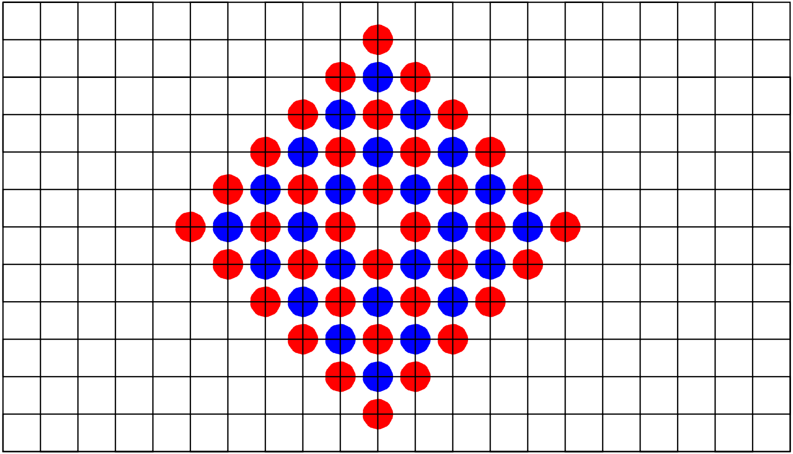


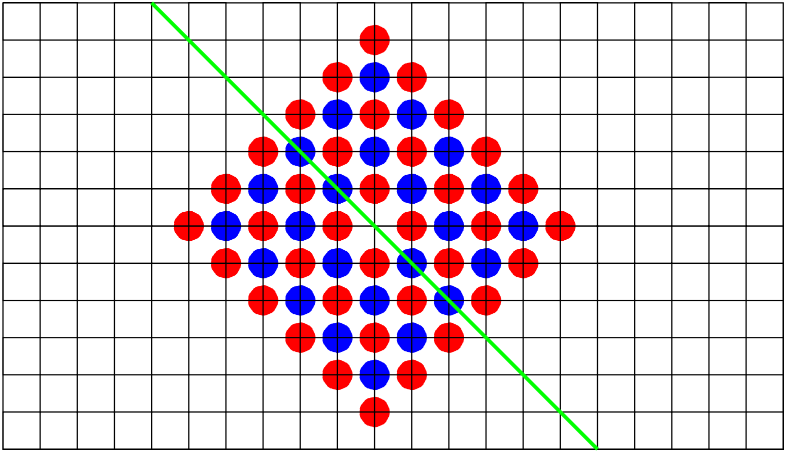


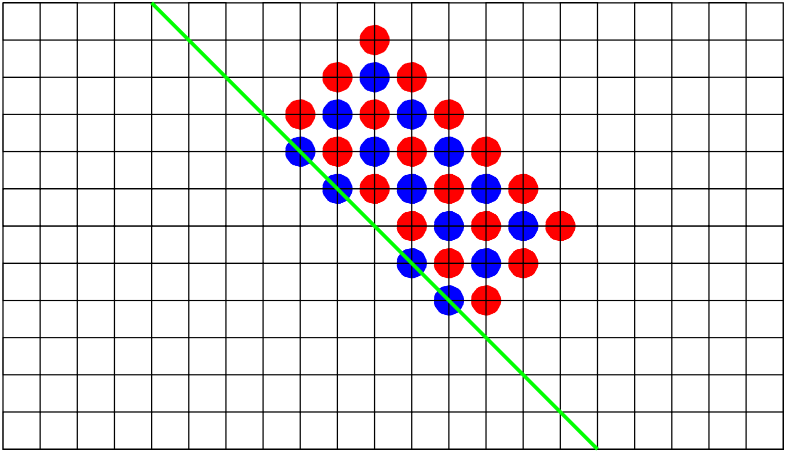


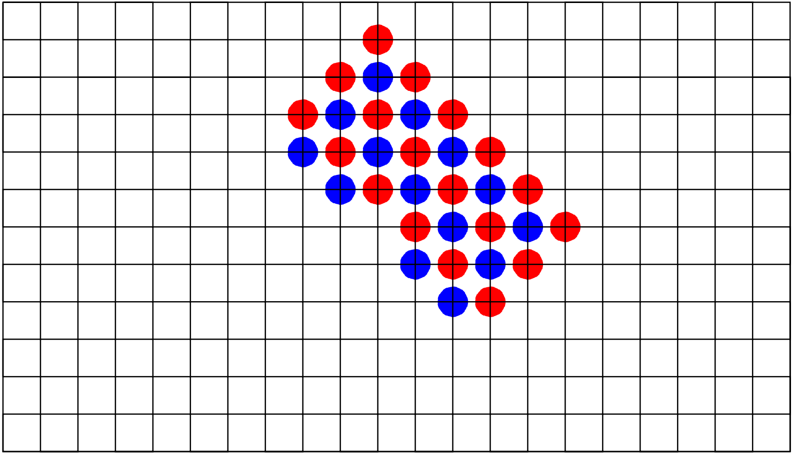


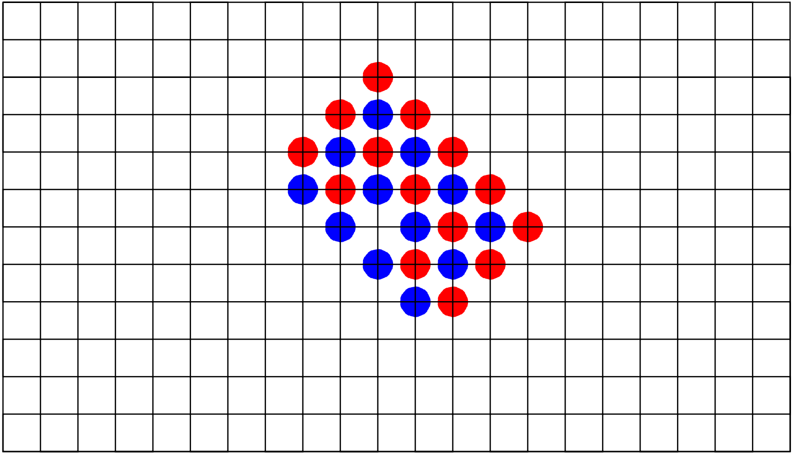
- c_n is the number of walks subject to restriction that walk must only visit allowed sites.
- Initially, list of sites derived from walk diffusion, i.e. no restrictions.
- Place restriction, e.g., by considering different possibilities for the endpoint.

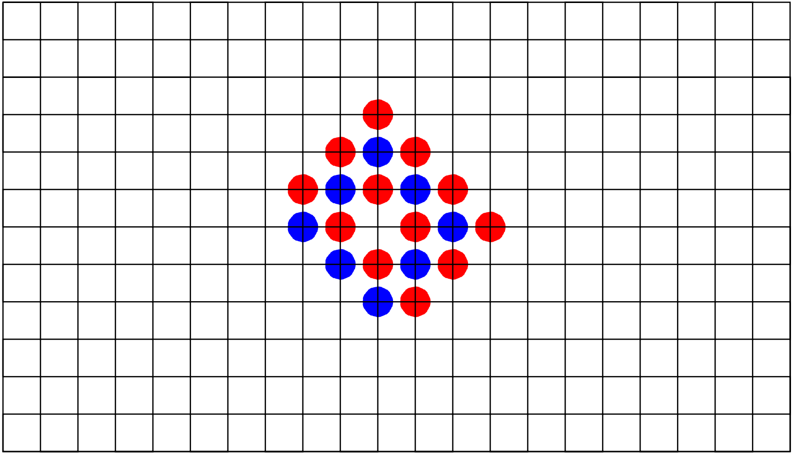


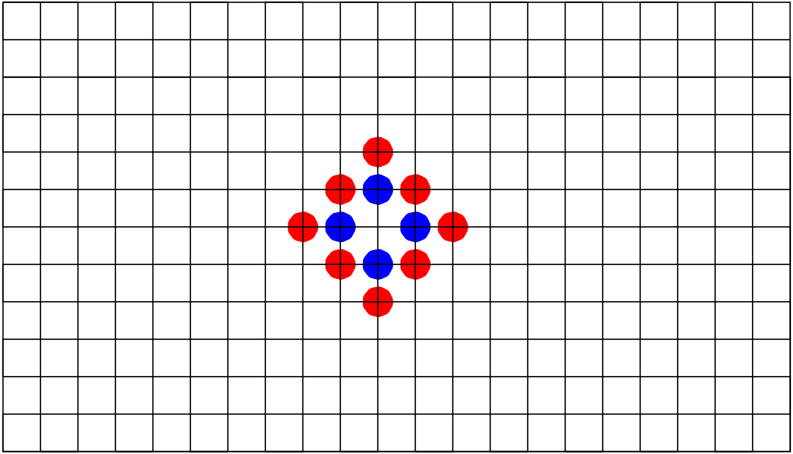


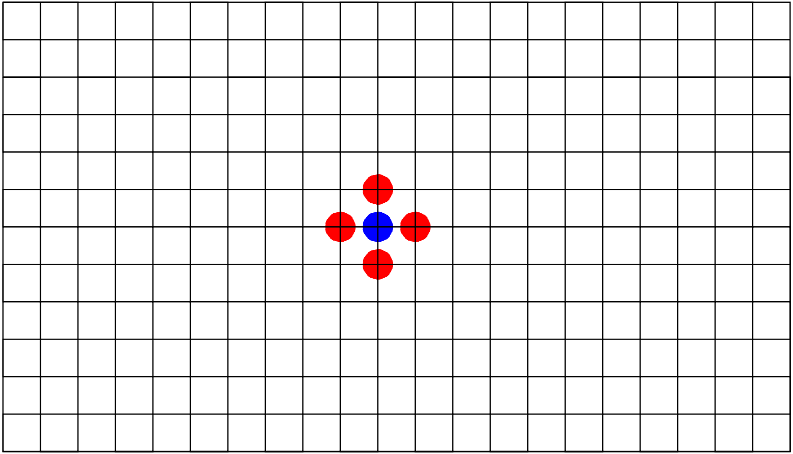


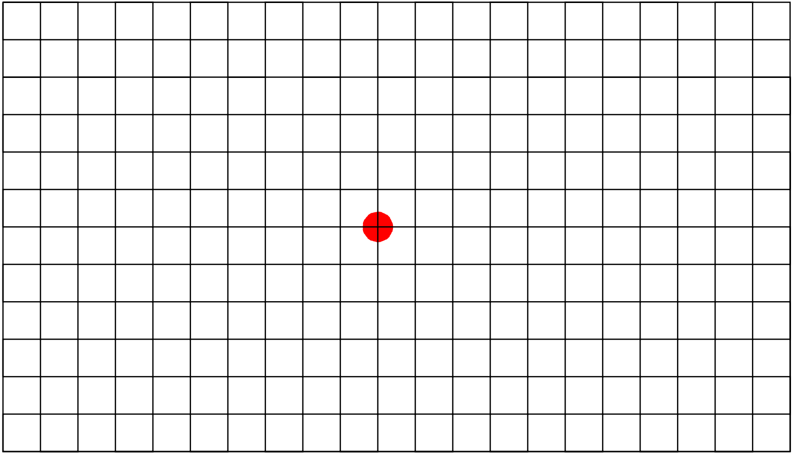




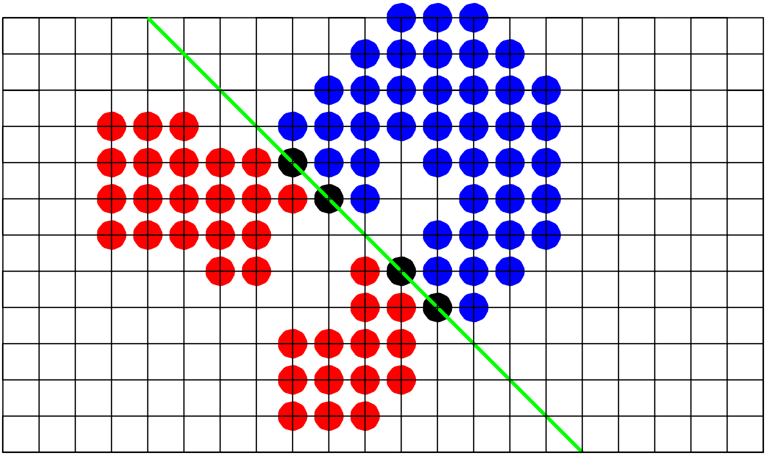




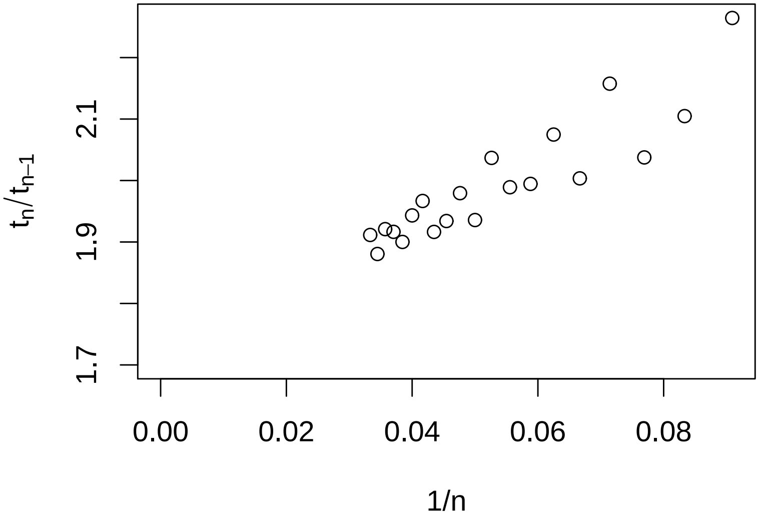


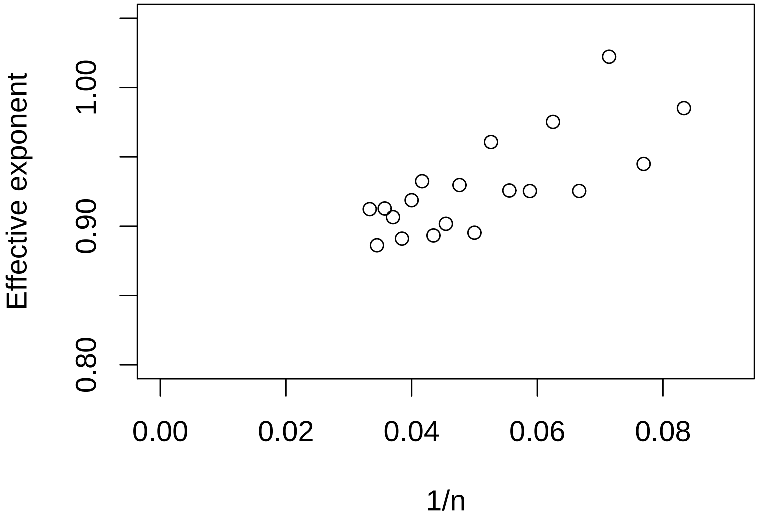


- Find ’principal axis’ of configuration, direction in which it is most spread out.
- This also defines an orthogonal boundary which cuts walk in half.
- Choose a region which is close to the boundary, and split it across the boundary.
- One choice stretches the walk in the direction of the axis, eventually leading to a configuration that can be split in half.
- Other choice squashes the walk, and will tend to lead to a more compact configuration, e.g. a disc for $d = 2$.
- Apply algorithm recursively to the set of sites which results from each choice.



- $\tau(c_n) \sim 2\tau(c_{n/2}) \times$ time required to split walk.
- Algorithm can't be worse than brute force.
- There exist tightly packed configurations with volume n, and hence cross-sections of area $n/n^{1/d} = n^{(d-1)/d}$.
- Need to fix *at least* the boundary sites to be able to split a walk in two. In practice need to constrain other sites so that walk remains on correct side of boundary.
- Hence best case scenario is that performance is $\sigma^{n^{(d-1)/d}}$. i.e. $\sigma^{\sqrt{n}}$ for $d = 2$.


t_n/t_{n-1}
1.7
1.9
2.1
0.00
0.02
0.04
0.06
0.08
1/n


Effective exponent
0.80
0.90
1.00
0.00
0.02
0.04
0.06
0.08
1/n

- Bad news: slow in practice (with work may be useful for $d > 3$).

- Bad news: slow in practice (with work may be useful for $d > 3$).
- Good news: asymptotically fast! Complexity is $\sigma^{n^{\alpha}}$, $\alpha \approx 0.85$, rather than exponential.

Algorithmic complexity

	Space	Time
brute force	n	μ^n
lace expansion	n	μ^n
2-step	n	λ^n
k-step	$n\mu^k$	ξ^n
finite lattice method	κ^n	κ^n
geometric splitting	n^{d+1}	σ^{n^α}

- Using geometric information allows for fast algorithms (FLM).
- Using *dynamic* geometric information has given us an algorithm that is asymptotically faster.

- Using geometric information allows for fast algorithms (FLM).
- Using *dynamic* geometric information has given us an algorithm that is asymptotically faster.
- Hopefully idea will lead to algorithms that are *fast*, rather than asymptotically fast!