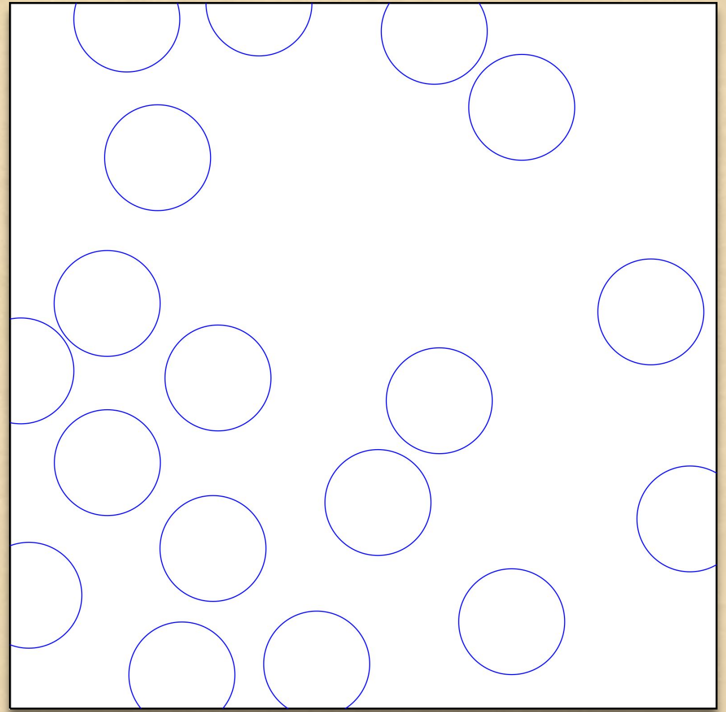
Data Science Dwarves know...

Stitching for Sampling

A new tool for high dimensional simulation

An unusual picture

The image to the right is of a zero overlap Strauss process of rate 100 over a region of area 1 with radius 0.15



Part 01
Spatial Data

Locating things with statistics

Spatial Data

01 Points

Usually in a subset of R^n

02 Modeling

Trees, cities, forest fires, disease outbreaks

03 Random

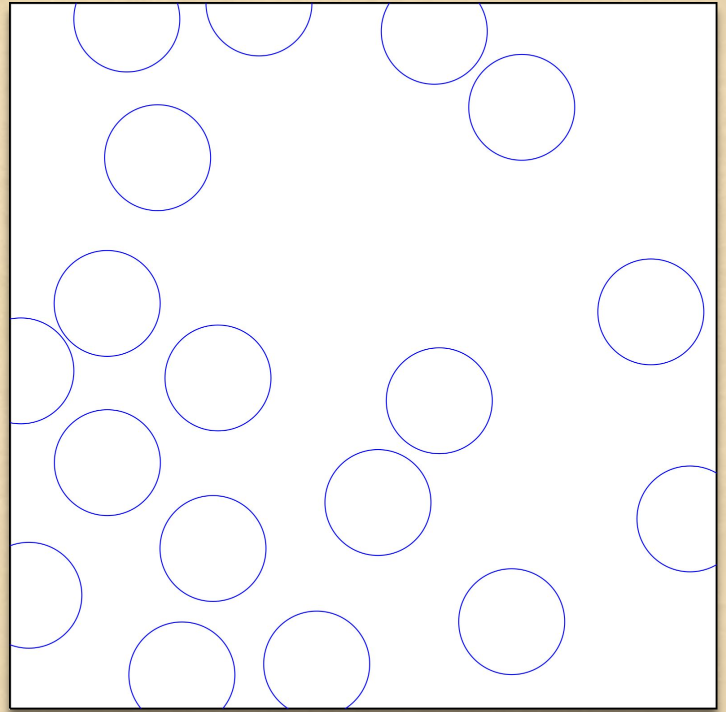
In model, points are placed randomly into the region

04 Strauss model

Points don’t like to be near one another

For instance...

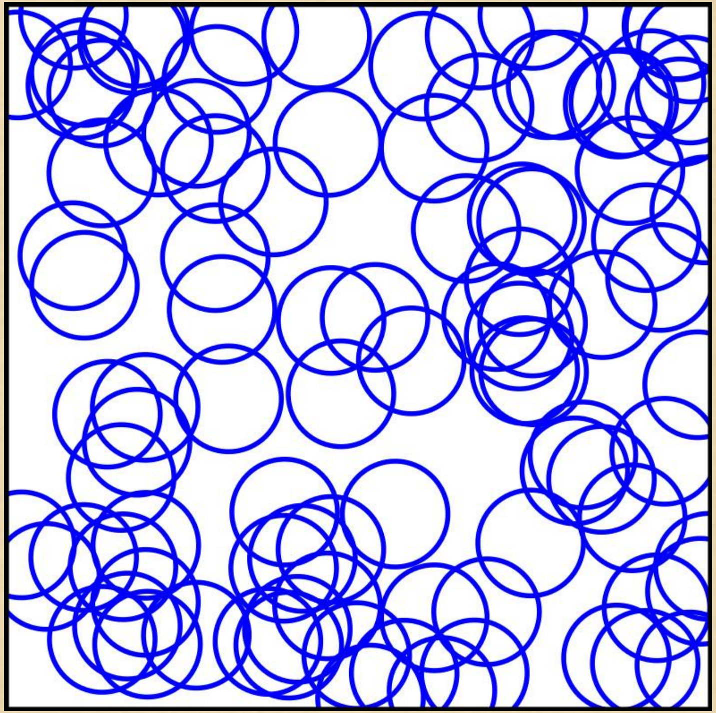
If this was locations of houses in a village in the Shire, they are farther apart than you would expect if they were just dropped uniformly at random



The simplest model

In Poisson point processes,

- points uniform over space
- points don't interact



Poisson point process rate λ

01

Disjoint regions

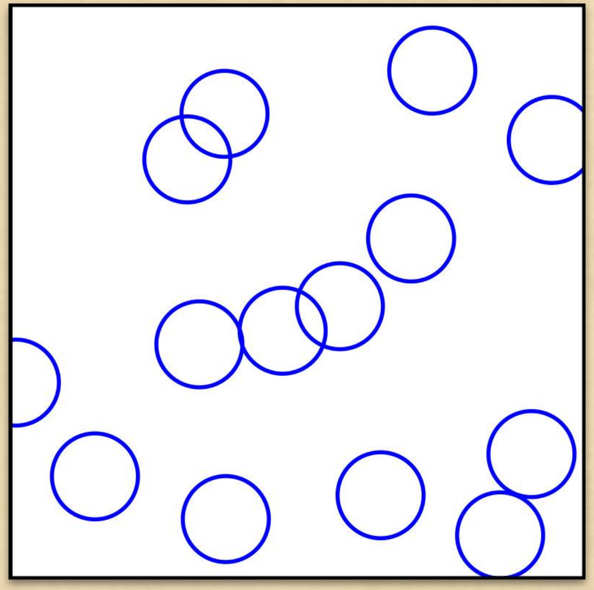
Points in disjoint regions are independent of each other

02

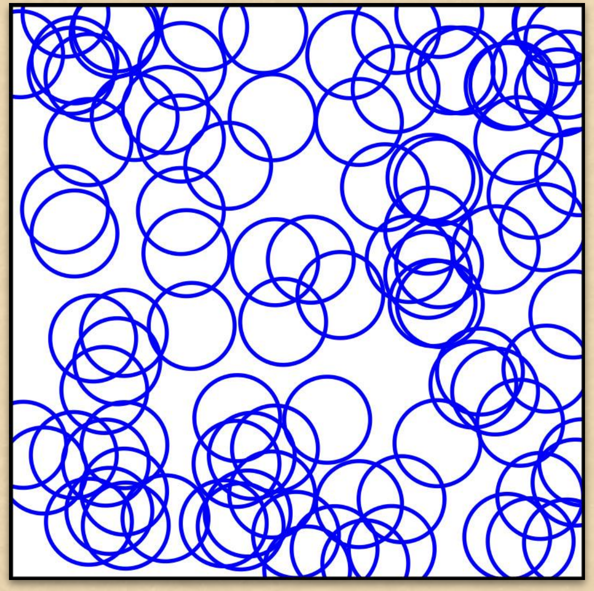
Average # of points

Mean # of points in region proportional to λ and size of region

Two examples

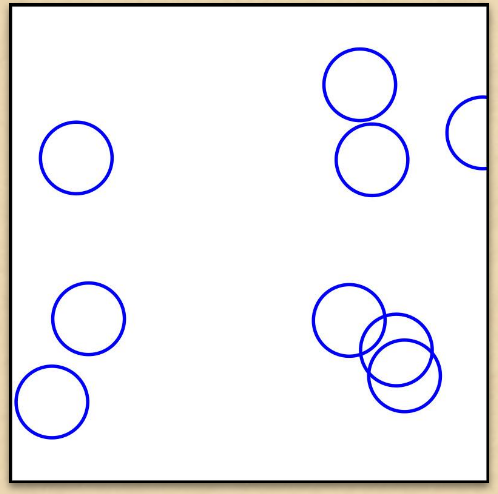


λ = 10



λ = 100

Repulsive point processes



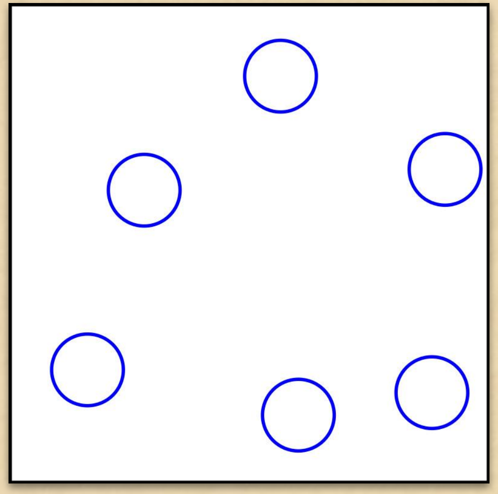
Two pairs overlap, weight = (1 / 2)^2 = 1 / 4

In repulsive point processes, points like to be farther apart

Use a density to accomplish this:

- Give low values when points close
- Give high values when points far

Repulsive point processes



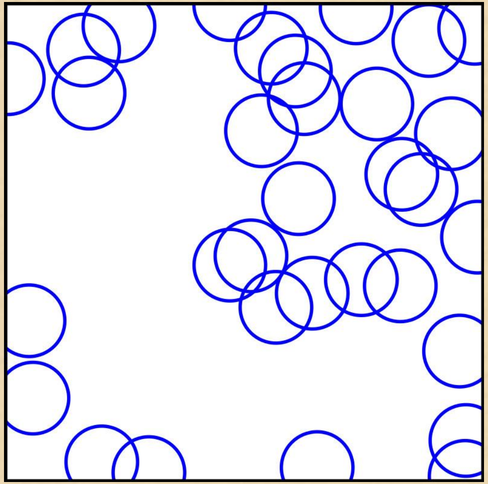
Zero pairs overlap, weight = (1 / 2)^0 = 1

In repulsive point processes, points like to be farther apart

Use a density to accomplish this:

- Give low values when points close
- Give high values when points far

The Strauss Process



$\lambda = 100$, $R = 0.15$, $\gamma = 1 / 2$

Puts a density on a PPP

Has parameters $R > 0$ and γ in $[0, 1]$

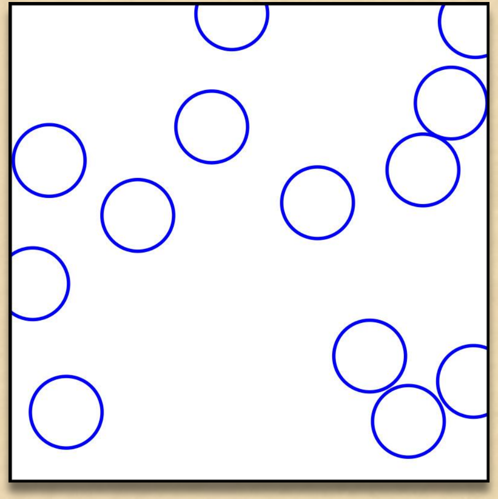
Multiplies density by γ for every pair of points within distance R of each other

The Strauss Process as a density

$$m(P) = \#\text{ pairs of points within distance } R$$

$$f(P) = \gamma^{m(P)}$$

The Hard Disk Model



$\lambda = 100$, $R = 0.15$, $\gamma = 0$

When $\gamma = 0$ this is the *hard disks model* where points cannot lie within distance R of each other at all

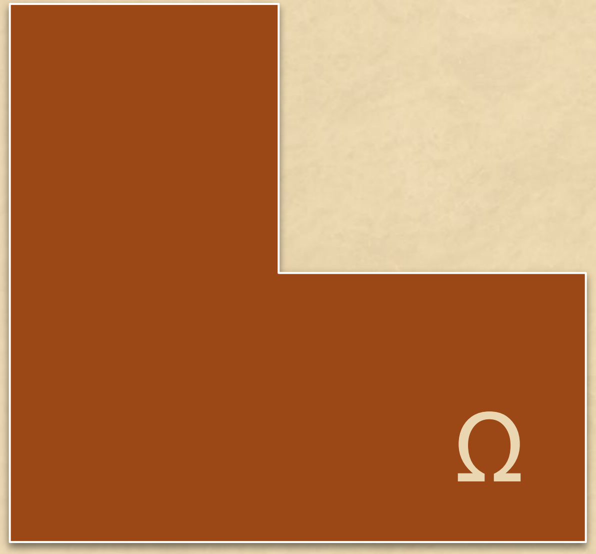
Throughout, I'm going to stick to the hard disks model because it's easier to visualize, and everything I say about it can be generalized to the Strauss model

Part 02

Simulation

Generating draws from distributions

Monte Carlo Methods



High dimensional models can be difficult to study analytically

Monte Carlo Methods draw random samples from the probabilistic models to calculate various properties of the distribution

- Expected value
- Variance
- Normalizing constant

Monte Carlo Methods for Spatial Models

For Spatial Models this means that we need efficient methods to draw samples from the model

Simulating Poisson point process

01 Decide # of points

Poisson distributed with parameter equal to λ times the size of the region

$$X \sim \mathsf{Pois}(\mu)$$

$$\mathbb{P}(X = i) = \exp(-\mu)\frac{\mu^i}{i!}$$

02 Distribute points

Independently, uniformly draw points over the region

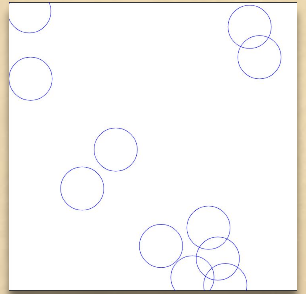
Simulating from the Hard Disk Model

The first method that comes to mind is acceptance rejection, a recursive algorithm

AR()

1) Draw P ~ PPP(λ)
2) If no two points in P are within distance R of each other return (P)
3) Else return(**AR**())

Simulating from the Hard Disk Model



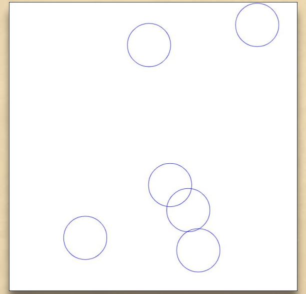
Call **AR**()

The first method that comes to mind is acceptance rejection, a recursive algorithm

AR()

1) Draw $P \sim \mathrm{PPP}(\lambda)$
2) If no two points in P are within distance R of each other return (P)
3) Else return(**AR**())

Simulating from the Hard Disk Model



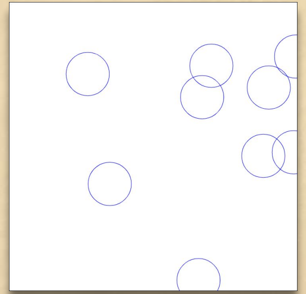
Call **AR**()

The first method that comes to mind is acceptance rejection, a recursive algorithm

AR()

1) Draw $P \sim \text{PPP}(\lambda)$
2) If no two points in P are within distance R of each other return (P)
3) Else return(**AR**())

Simulating from the Hard Disk Model



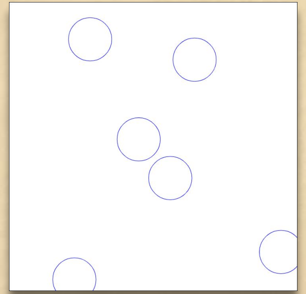
Call **AR**()

The first method that comes to mind is acceptance rejection, a recursive algorithm

AR()

1) Draw $P \sim \text{PPP}(\lambda)$
2) If no two points in P are within distance R of each other return (P)
3) Else return(**AR**())

Simulating from the Hard Disk Model



Return as output

The first method that comes to mind is acceptance rejection, a recursive algorithm

AR()

1) Draw P ~ PPP(λ)
2) If no two points in P are within distance R of each other return (P)
3) Else return(**AR**())

Works well if area of region small

AR() → **AR()** → **AR()**

Three calls to **AR()**

Can measure running time by (random) number of calls to **AR()**

This is a geometrically distributed random variable with mean value:

1 / probability of accepting

Perfect Simulation Algorithm

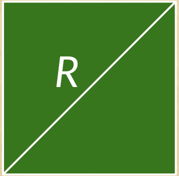
A Perfect Simulation algorithm...

- Uses randomness
- Calls itself recursively
- Ends with probability 1
- Output comes exactly from target distribution

Acceptance Rejection

- Credited to John von Neumann (1951)
- Didn't claim to be the inventor
- Also didn't actually bother to prove it worked
- For 45 years, was the only perfect simulation algorithm

Doesn't work well if area of region large

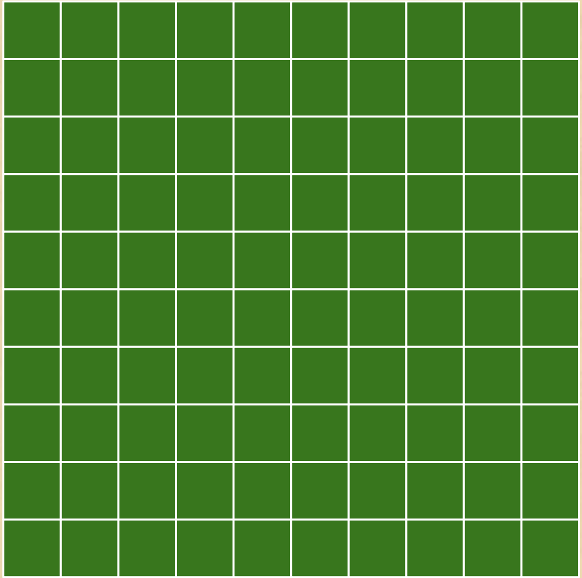


For square of diagonal length *R*, if two or more points then they must be within distance *R* of each other

So chance of accepting is chance of one or zero points appearing, which is

$$\exp\left(-\lambda\frac{R^2}{2}\right)\left[1+\lambda\frac{R^2}{2}\right] < 1$$

Doesn't work well if area of region large



Chance of accepting in large square at most chance that every small square accepts which is

$$\exp\left(-\lambda\frac{R^2}{2}\right)\left[1+\lambda\frac{R^2}{2}\right]^{100}$$

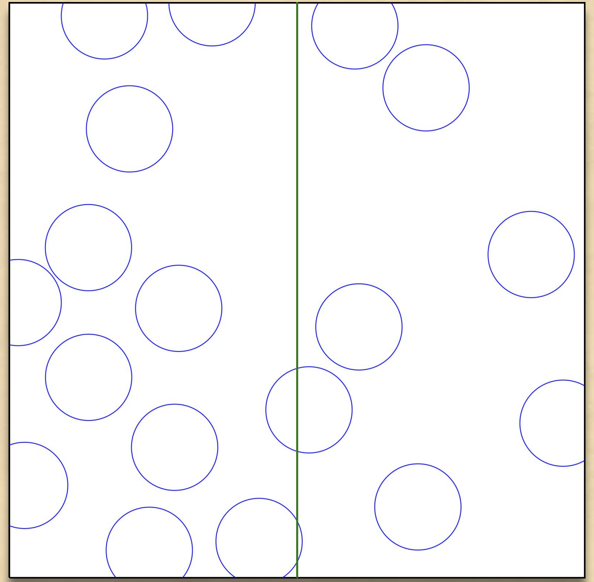
So overall chance of accepting declines exponentially in *area* of region

Part 03

Stitching

Recursion to the rescue

A key property of Hard Disks Model



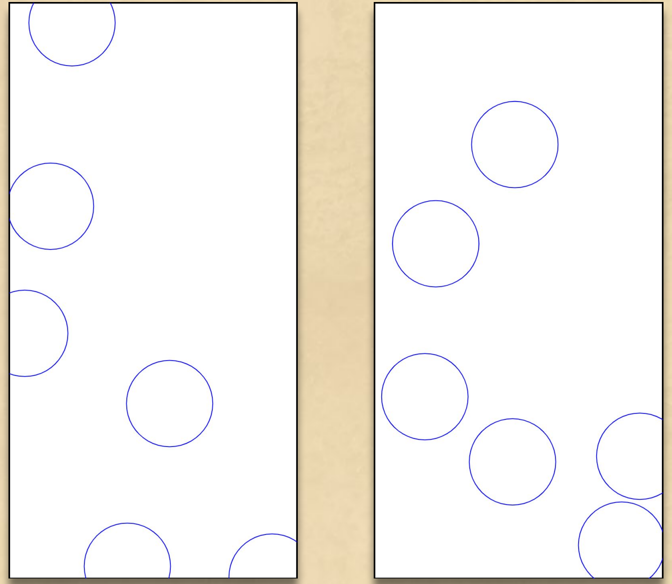
Suppose the square is divided into two rectangles of equal size

If original sample is a valid hard disk draw, then so are both the sample on the left hand side and the right hand side

The idea behind stitching

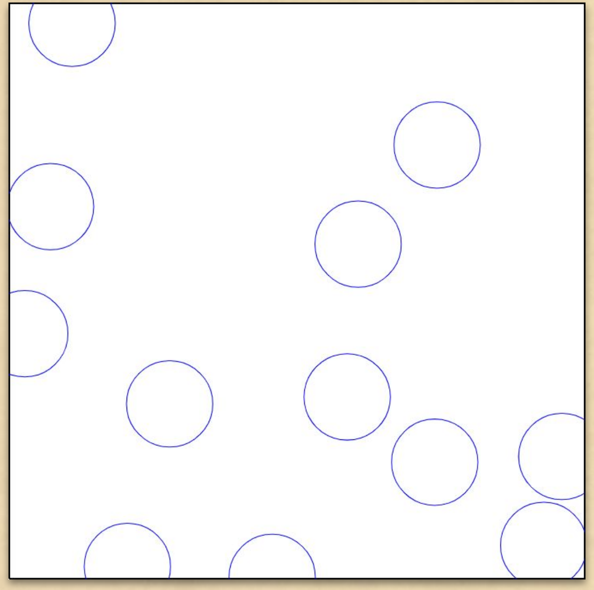
1) Recursively draw sample P_left from left hand side and P_right from right hand side
2) If P_left union P_right is a valid hard disks draw, return it and quit
3) Otherwise start over from scratch drawing the sample

The idea behind stitching



1) Recursively draw sample P_left from left hand side and P_right from right hand side
2) If P_left union P_right is a valid hard disks draw, return it and quit
3) Otherwise start over from scratch drawing the sample

The idea behind stitching



1) Recursively draw sample P_left from left hand side and P_right from right hand side
2) If P_left union P_right is a valid hard disks draw, return it and quit
3) Otherwise start over from scratch drawing the sample

Is this faster?

Yes!

Expected time to run basic acceptance rejection:

$$\mathbb{E}(T) = \frac{1}{p_{\text{left}} p_{\text{right}} p_{\text{stitch}}}$$

Expected time to run stitching

$$\mathbb{E}(T_{\text{stitch}}) = \left[\frac{1}{p_{\text{left}}} + \frac{1}{p_{\text{right}}}\right] \frac{1}{p_{\text{stitch}}}$$

When to use

Improves running time as long as

$$\frac{1}{p_{\text{left}} p_{\text{right}}} \geq \frac{1}{p_{\text{left}}} + \frac{1}{p_{\text{right}}}$$

Or more simply

$$1 \geq p_{\text{left}} + p_{\text{right}}$$

If this doesn't hold...

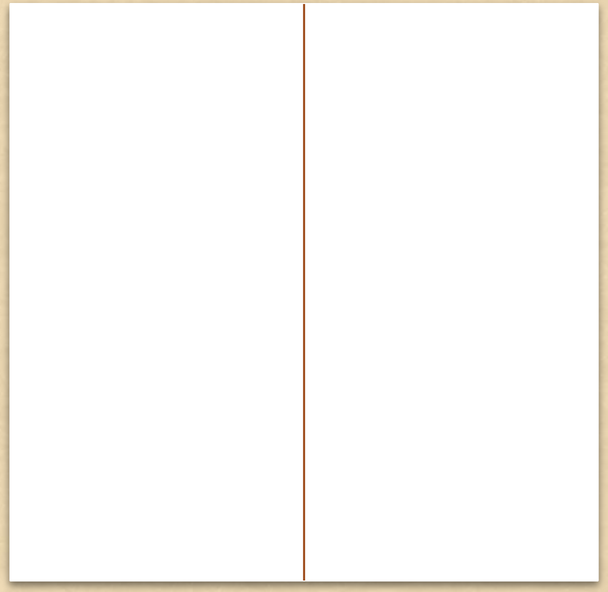
Then have

$$p_{\text{left}} \geq \frac{1}{2}, \; p_{\text{right}} \geq \frac{1}{2}$$

So basic Acceptance Rejection is fast in this case

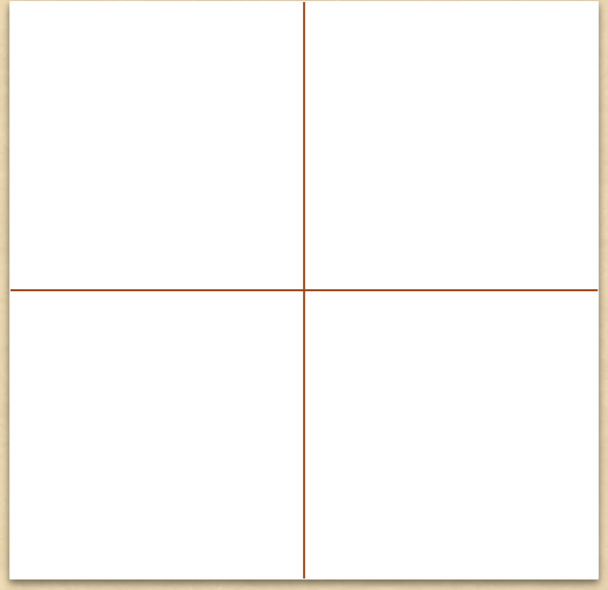
Using Recursion

No need to stop at breaking space in half once!



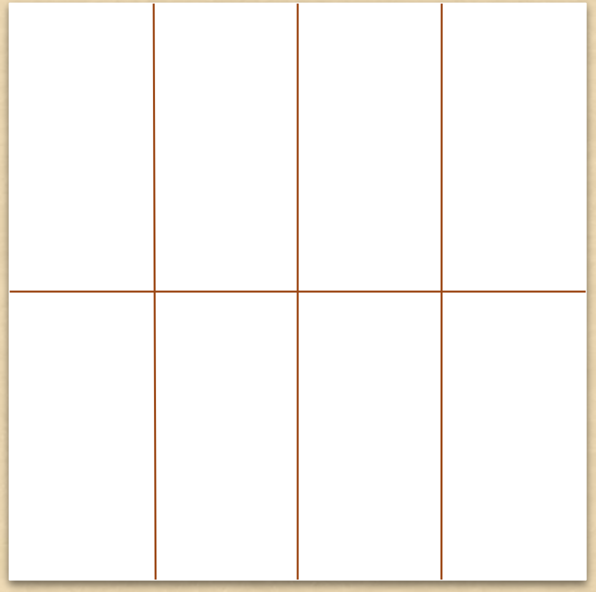
Recursion 2nd level

Use Stitching for left and right hand side



Recursion 3rd level

Break each of those four pieces
into left and right hand side



When to stop

Stop dividing the space when

$$p_{\text{left}} \geq \frac{1}{2}, \; p_{\text{right}} \geq \frac{1}{2}$$

But we don't know these values!

Solution: Try acceptance rejection once, otherwise use recursion

Final algorithm

Stitching_HDM(*S*)

1) Draw *P* ~ PPP(λ, S)
2) If *P* is a hard disk model, return *P*
3) Partition *S* into *S_1* and *S_2*
4) Draw *P_1* using **Stitching_HDM**(*S_1*)
5) Draw *P_2* using **Stitching_HDM**(*S_2*)
6) Let *Q* be the union of *P_1* and *P_2*
7) If *Q* is a hard disk model, return *Q*
8) Otherwise return **Stitching_HDM**(*S*)

Part 04
Performance

Does this really help?

Perfect simulation for Strauss

Perfect Simulation Timeline

- Propp and Wilson 1996
  Coupling From the Past

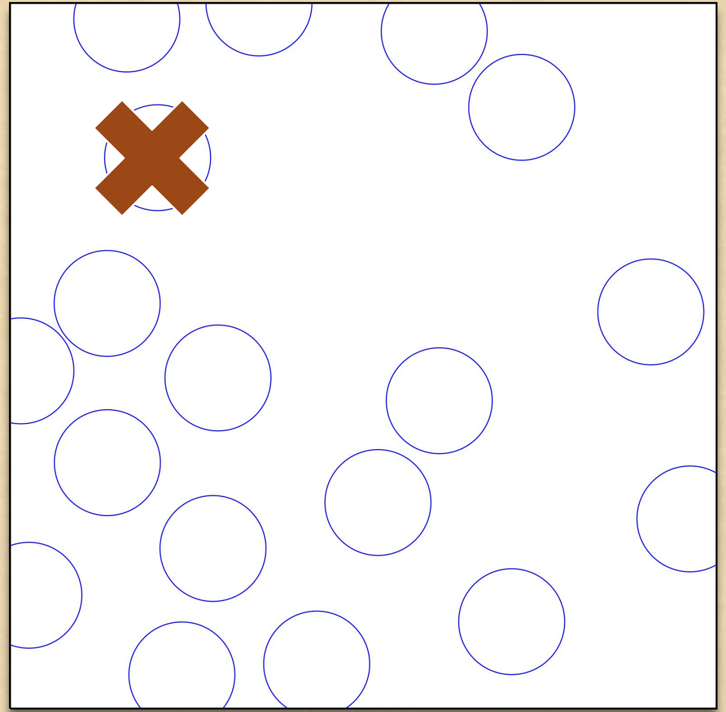
- Kendall and Møller 1999
  Dominated CFTP for Point Process

- Huber 2015
  Birth-Death-Swap for Point Process

- Jerrum and Guo 2019
  Partially Recursive Sampling for Point Process

CFTP based on Markov chains

These *jump chains* make local changes to a state, either removed or adding a single point



[Dis/Ad] vantages of jump chains

Advantages

- When λ small relative to R^2, fast
- Critical Value of λ
- For λ below critical value, polynomial time

Disadvantages

- For λ above critical value, exponential running time

(Dis/Ad) vantages of stitching

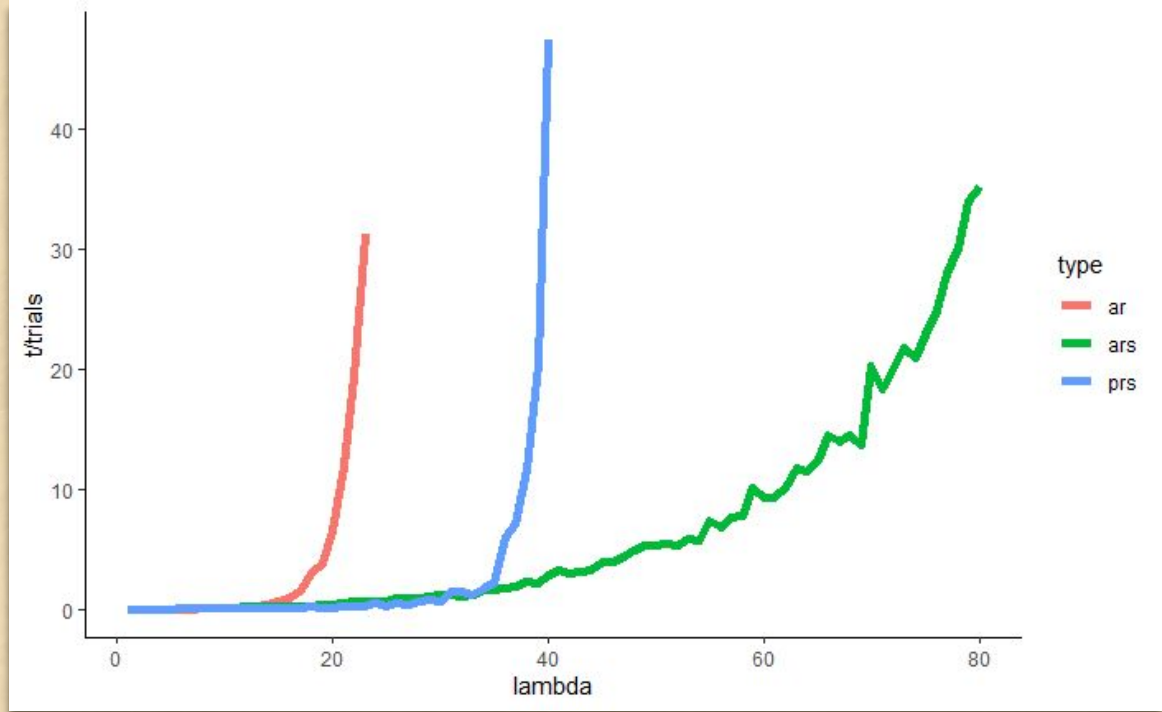
Advantages

- Runs for larger values of lambda
- Easier to code
- Exponential time based on boundary length of split, rather than area like AR

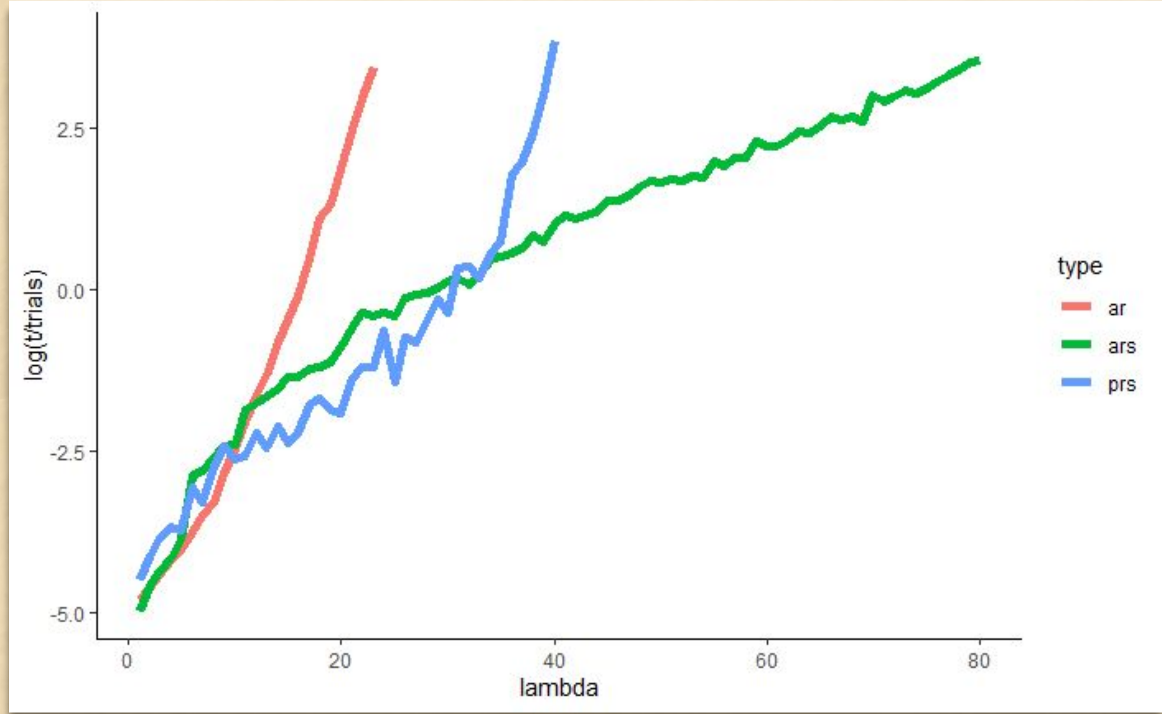
Disadvantages

- Always exponential running time

Experimental running time

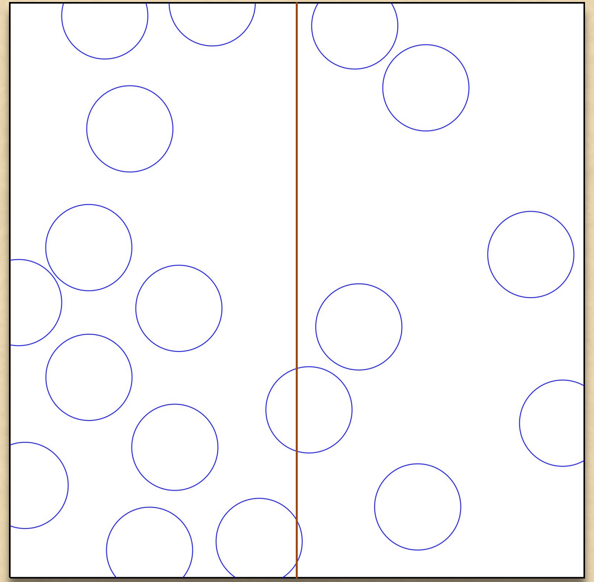


Experimental log running time



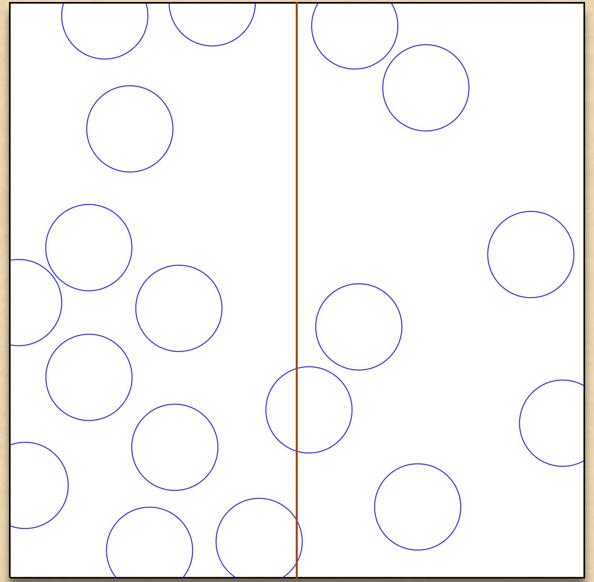
Why the improvement?

Acceptance needed at *length of boundary* between halves, not the *areas* of the halves



Why the improvement?

Acceptance easier at boundary when both sides already have fewer points than in PPP



Part 05

Correctness

How do we know this works?

Fundamental Theorem of Perfect Simulation

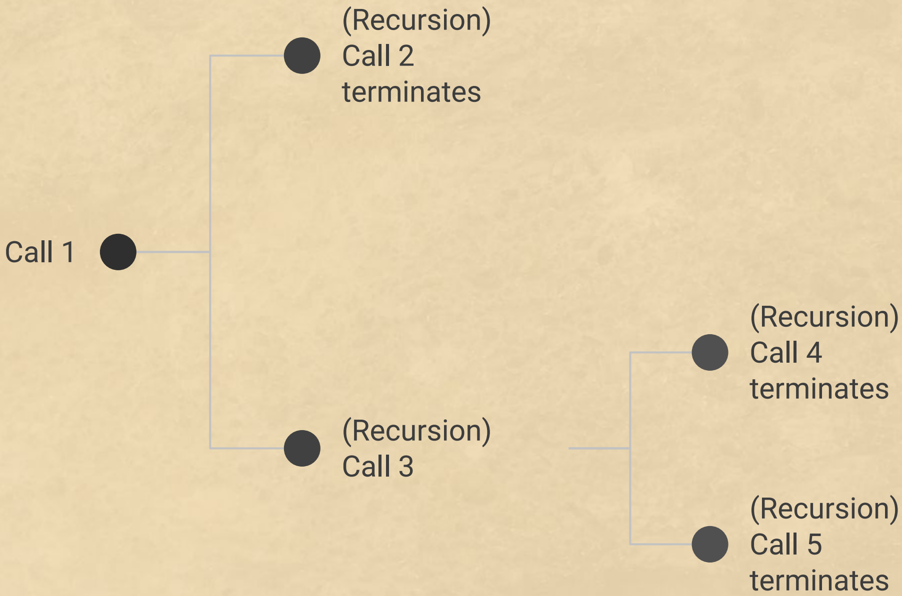
Suppose a probabilistic recursive algorithm has two properties:

01 It is *locally correct*.

02 It terminates with probability 1.

Then it is *globally correct*.

Can use recursion but must terminate with probability 1



Locally correct

A recursive algorithm is *locally correct* if when you replace recursive calls with an *oracle* that comes directly from the target distribution, the overall algorithm is correct.

Oracles are always correct

Call 1

- Oracle gives answer
- Oracle gives answer

Globally correct

A recursive algorithm is *globally correct* if its final output comes from the target distribution

Local correction for stitching HDM

Stitching_HDM(*S*)

1) Draw *P* ~ PPP(λ, S)
2) If *P* is a hard disk model, return *P*
3) Partition *S* into *S*_1 and *S*_2
4) Draw *P*_1 using **Stitching_HDM**(*S*_1)
5) Draw *P*_2 using **Stitching_HDM**(*S*_2)
6) Let Q be the union of *P*_1 and *P*_2
7) If Q is a hard disk model, return Q
8) Otherwise return **Stitching_HDM**(*S*)

Stitching_HDM_oracle(*S*)

1) Draw *P* ~ PPP(λ, S)
2) If *P* is a hard disk model, return *P*
3) Partition *S* into *S*_1 and *S*_2
4) Draw *P*_1 from HDM over *S*_1
5) Draw *P*_2 from HDM over *S*_2
6) Let Q be the union of *P*_1 and *P*_2
7) If Q is a hard disk model, return Q
8) Otherwise return from HDM over *S*

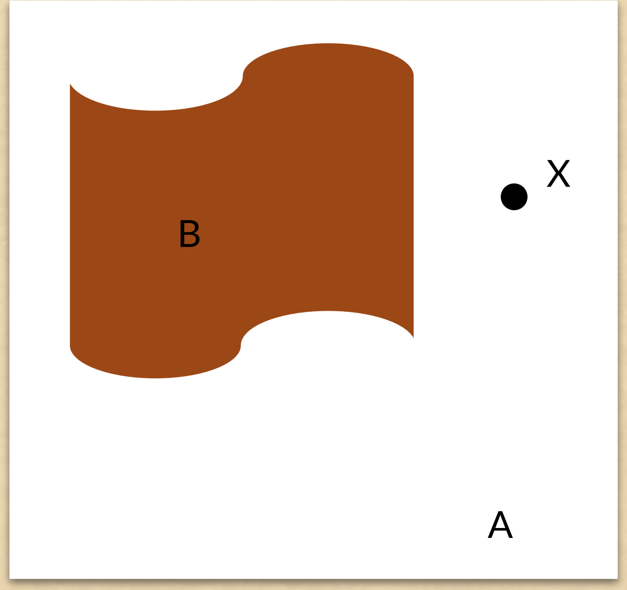
Note: HDM = Hard Disks Model

A nice property of uniforms

Suppose

- B is a subset of A
- X is uniform over A
- X happens to fall into B

Then X is uniform over B



Correctness of oracle version

Property of uniforms

- Suppose X ~ Unif(A)
- For B a subset of A, suppose X is in B
- Then [X | X in B] ~ Unif(B)

So *P*_1 union *P*_2 a valid HDM means that their union is uniform over valid HDM

Stitching_HDM_oracle(*S*)

1) Draw *P* ~ PPP(λ, S)
2) If *P* is a hard disk model, return *P*
3) Partition *S* into *S*_1 and *S*_2
4) Draw *P*_1 from HDM over *S*_1
5) Draw *P*_2 from HDM over *S*_2
6) Let *Q* be the union of *P*_1 and *P*_2
7) If Q is a hard disk model, return Q
8) Otherwise return from HDM over *S*

Note: HDM = Hard Disks Model

FTPS Proof Outline

01 Make truncated version of algorithm that stops after n recursions and uses oracles thereafter

02 Through local correctness + induction, truncated version has correct output

03 Since original algorithm terminates with probability 1, as n goes to infinity, truncated alg output equals original alg output

Part 06
Last Thoughts

What is the takeaway?

Stitching can be done for densities as well!

Partition the state space

$$S = S_1 \cup S_2, S_1 \cap S_2 = \emptyset$$

Factor the density

$$f(S) = f(S_1)f(S_2)f_{\text{stitch}}(S_1, S_2)$$

Gives a way to do Strauss process for γ > 0

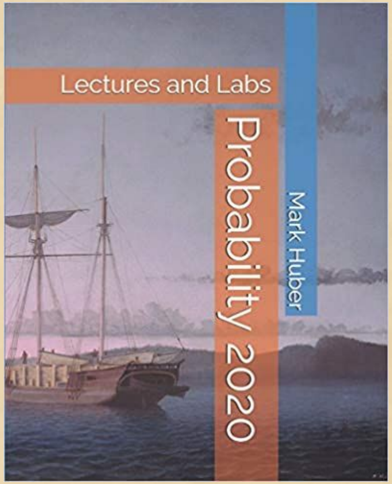
When to use

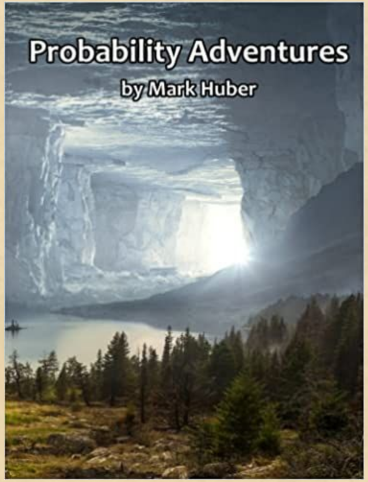
When to use Stitching?

- Designed for Point Processes
- Broader application: works whenever density product of three parts, two of which look like original problem
- Can be effective in situation where traditional Markov chain methods too slow

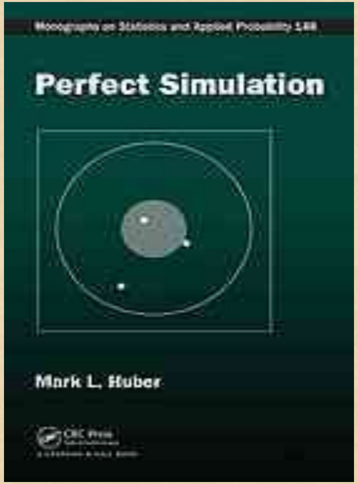
https://arxiv.org/abs/2012.08665

Want to learn more?





Free to download
Probability textbooks



Held by the Library

http://www.markhuberdatascience.org/

Thanks for watching!

Thanks!

Does anyone have any questions?
youremail@freepik.com
+91 620 421 838
yourcompany.com