Simulations in Statistical Physics

Course for MSc physics students

Janos Török

Department of Theoretical Physics

October 15, 2013

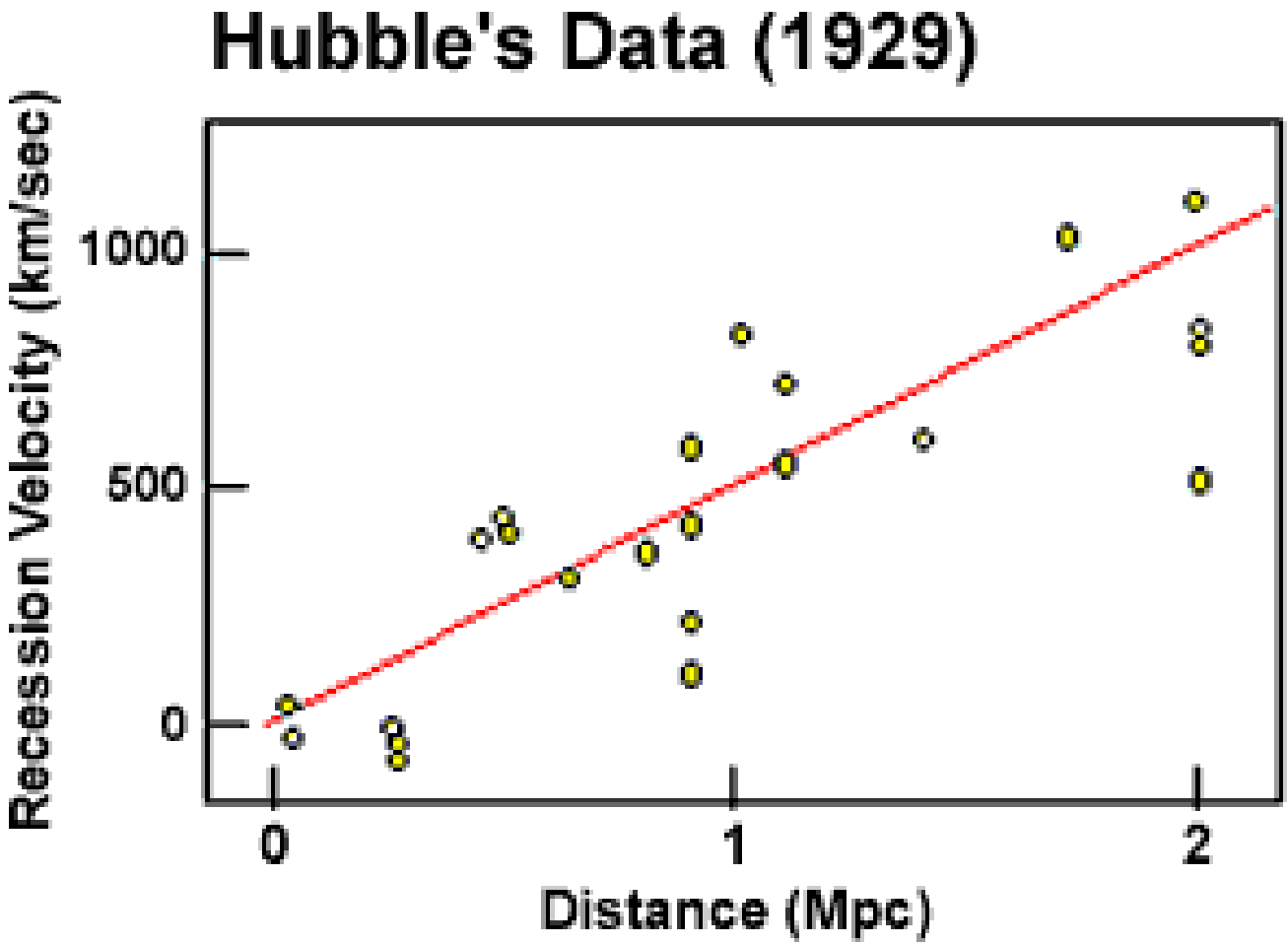
Fitting

Linear regression

$$
\begin{aligned}
y &= \alpha + \beta x \\
\hat{\beta} &= \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2} \\
\hat{\alpha} &= \bar{y} - \hat{\beta}\bar{x} \\
\rho &= \frac{\overline{xy}}{\sqrt{\bar{x}\bar{y}}}
\end{aligned}
\tag{1}
$$

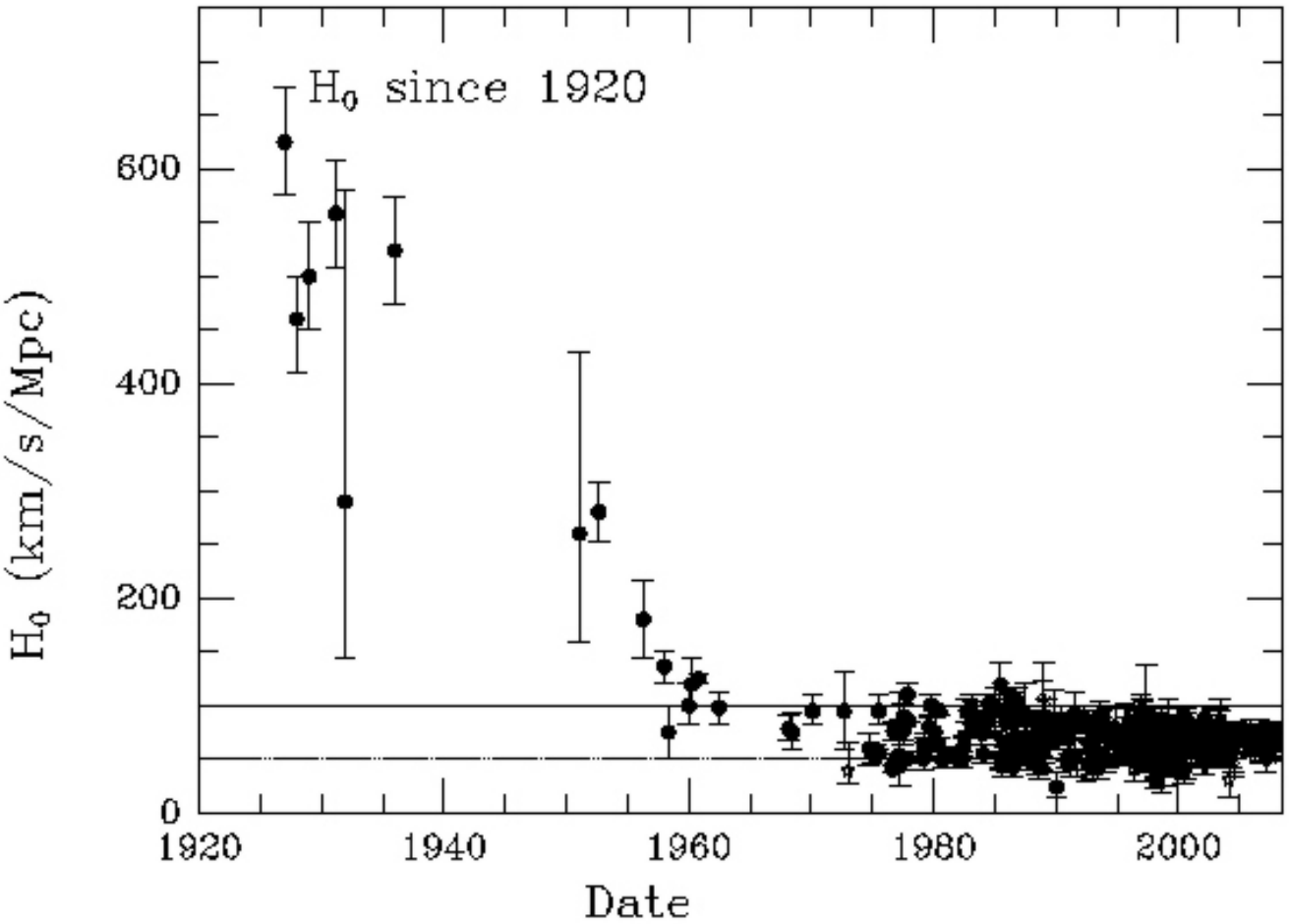
Fitting

Houbble original fit:



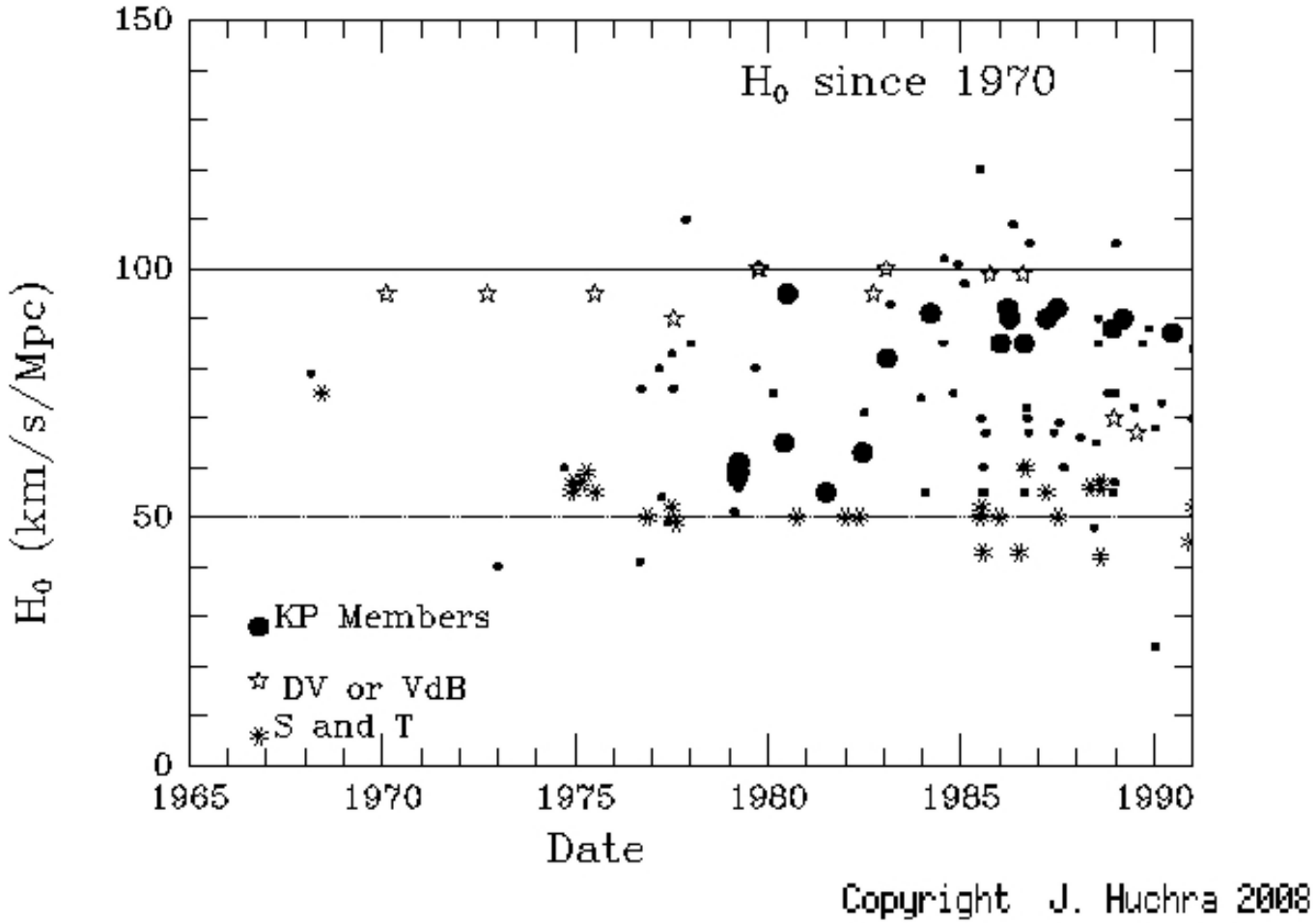
Fitting

Houbble change in time:



Fitting

Houbble change in time:



Finite size scaling

- Correlation length

$$\xi \propto |T - T_c|^{-\nu}$$

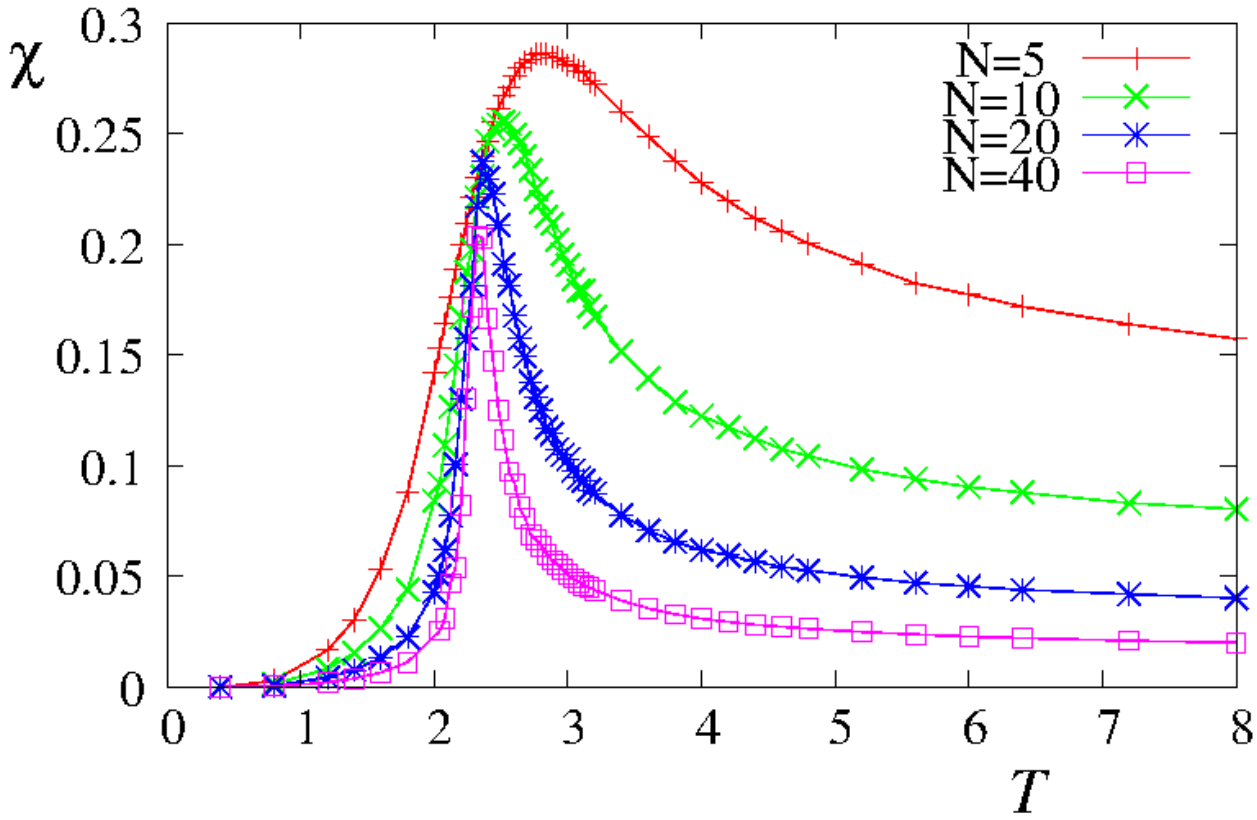
- If L is finite ξ cannot be larger than L

$$L \propto |T(L) - T_c|^{-\nu}$$

- The position and the width of the transition

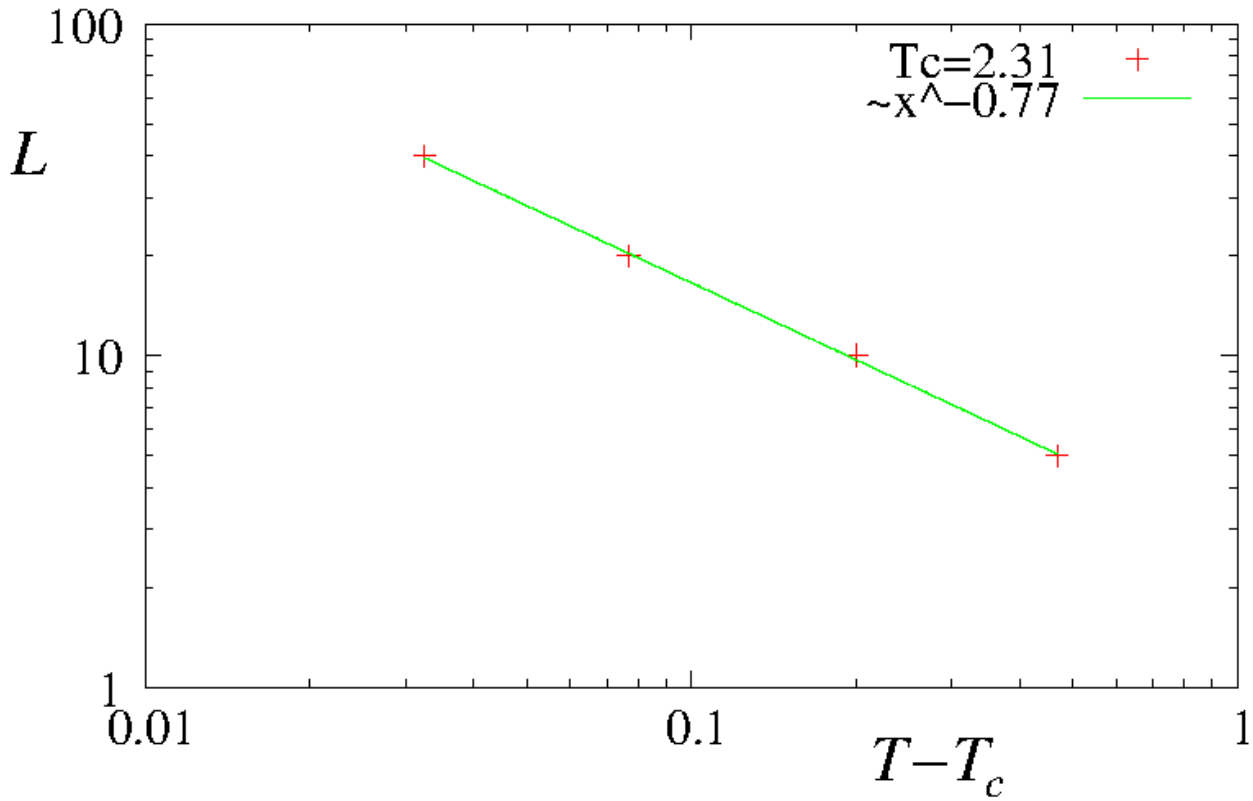
$$|T(L) - T_c| \propto L^{-1/\nu}$$
$$\sigma(L) \propto L^{-1/\nu}$$

Ising model susceptibility



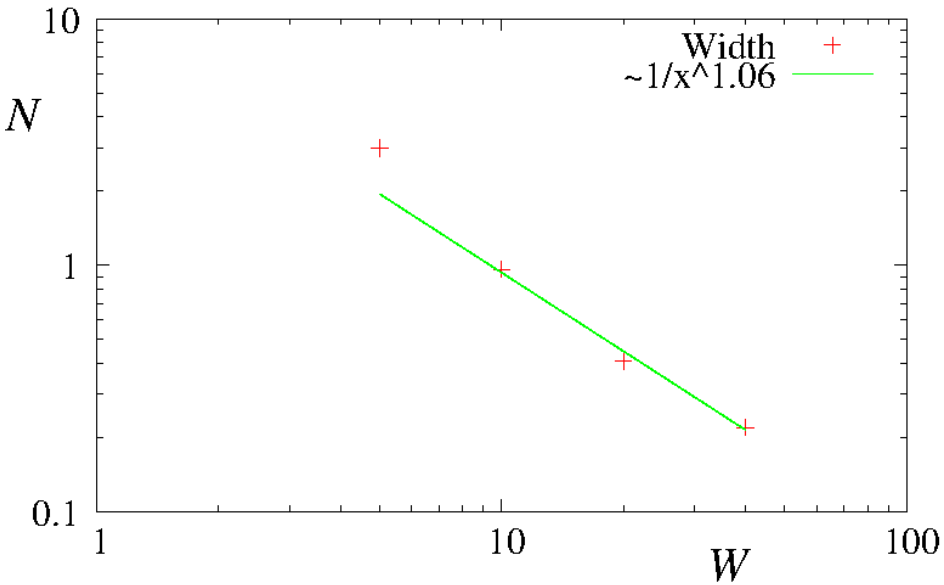
Three parameter fit: Ising model

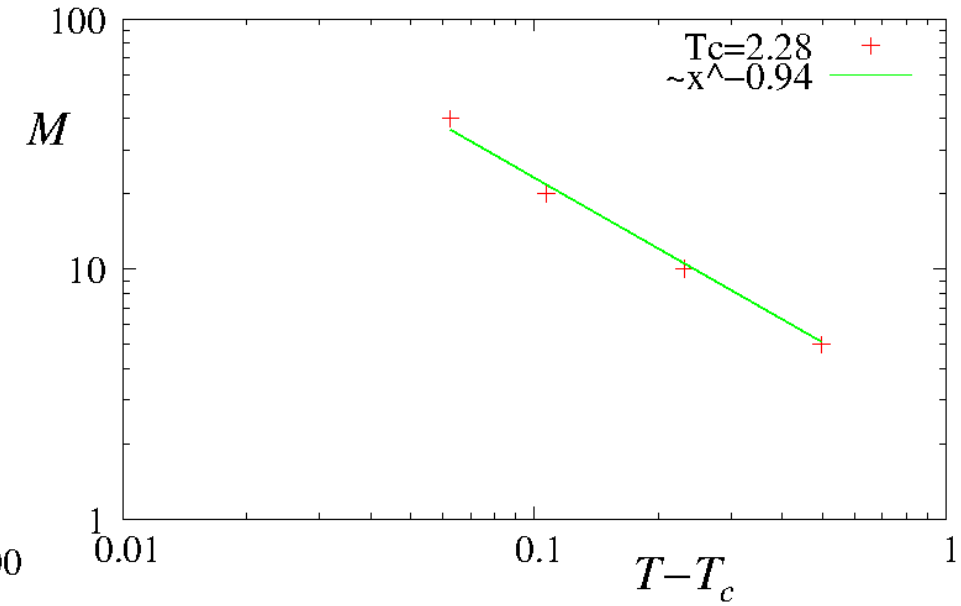
- Theory: $\nu = 1,\ T_c \simeq 2.27$



Finite size scaling: Ising model

- Theory: $\nu = 1,\ T_c \simeq 2.27$





Metropolis algorithm

(Metropoli-Rosenbluth-Rosenbluth-Teller-Teller=MR^{2}T^2 algorithm)

- Sequence of configurations using a Markov chain
- Configuration is generated from the previous one
- Transition probability: equilibrium probability
- Detailed balance:

$$P(x)P(x \to x') = P(x')P(x' \to x)$$

- Rewritten:

$$\frac{P(x \to x')}{P(x' \to x)} = \frac{P(x')}{P(x)} = e^{-\beta \Delta E}$$

- Only the ration of transition probabilities are fixed

Metropolis algorithm

(Metropoli-Rosenbluth-Rosenbluth-Teller-Teller=MR^{2}T^2 algorithm)

$$\frac{P(x \to x')}{P(x' \to x)} = \frac{P(x')}{P(x)} = e^{-\beta \Delta E}$$

- Metropolis:

$$P(x \to x') = \begin{cases} e^{-\beta \Delta E} & \text{if} \Delta E > 0 \\ 1 & \text{otherwise} \end{cases}$$

- Symmetric:

$$P(x \to x') = \frac{e^{-\beta \Delta E}}{1 + e^{-\beta \Delta E}}$$

Metropolis algorithm

Recipes:

- Choose an elementary step $x \to x'$
- Calculate ΔE
- Calculate $P(x \to x')$
- Generate random number $r \in [0, 1]$
- If $r < P(x \to x')$ then new state is x'; otherwise it remains x
- Increase time
- Measure what you want
- Restart

Metropolis algorithm, proposal probability

Transition probability:

$$P(x \to x') = g(x \to x')A(x \to x')$$

- $g(x \to x')$: proposal probability
  - Generally uniform
  - If different interactions are present then it must be incorporated
- $A(x \to x')$: acceptance probability
  - Metropolis
  - Symmetric

Metropolis, *proof*

State flow
Let $E > E'$:

- $x \to x'$

$$P(x)g(x \to x')A(x \to x') = P(x)$$

- $x' \to x$

$$P(x')g(x' \to x)A(x' \to x) = P(x')e^{-\beta\Delta E}$$

- In equilibrium they are equal:

$$\frac{P(x)}{P(x')} = e^{\beta\Delta E}$$

- What we wanted.

Do we need optimization?

- Correlation lenth ξ
- Characteristic time τ_{char}
- Dynamical exponent z

$$\tau_{\mathrm{char}} \propto \xi^z$$

- For 2d Ising model $z \simeq 2.17$
- Simulation time:

$$t_{\mathrm{CPU}} \sim L^{d+z}$$

We need more effective algorithms!

Multri-spin algorithm for 2d Ising model

History...

- Operations:
  - Check if neighbor is parallel: XOR
  - sum of antiparallel spins: sum of previous XOR
- Result: discrete energy difference can be 0, 1, 2, 3, 4

Metropolis	0	1	2	3	4
$\Delta E/J$	8	4	0	4	8
$P(x \to x')$	$\exp(-8\beta)$	$\exp(-4\beta)$	0	0	0

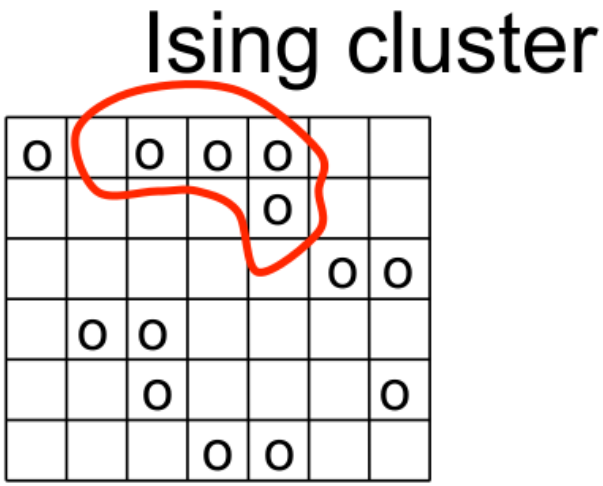
- (of course $P(x \to x')$ in array)
- 4 bit is enough to store result
- Use every fourth bit to store a spin.

Multri-spin algorithm for 2d Ising model

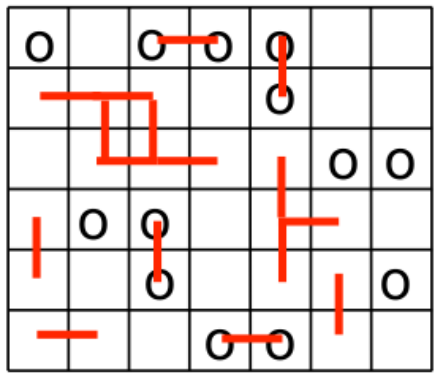
- Historical solution
  - Every fourth bit in the integer is a spin
  - To get neighbors bit shift operation must be made
  - We get `sizeof(int)/4` bits at once
  - Go through the sample in a typewriter style
  - Nowdays may even be slower as array operations are fast
- Use it for ensemble average
  - One member of the array contains the spin of one position
  - Multiple simulation instances
  - With Metropolis algorithm few random numbers are needed (at high T)
- Does not really matter only factors can be won, $t_{\mathrm{CPU}} \sim L^{d+z}$ still holds

Cluster algorithm

- Flip more spins together. How?
- The solution – based on an old relationship between the percolation and the Potts model – is that we consider the spin configuration as a correlated site percolation problem
- Ising cluster: a percolating cluster of parallel spins
- Ising droplets: a percolating subset of an Ising cluster
  $p_B = 1 - \exp(-2\beta J)$



Ising configuration



Ising „droplets”

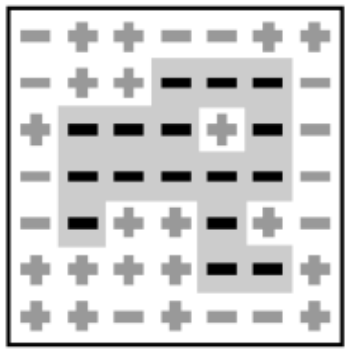
Swendsen-Wang algorithm

- Take an Ising configuration
- With probability $p_B = 1 - \exp(-2\beta J)$ make connection between *parallel* spins
- Identify the droplets by Hoshen-Kopelman algorithm
- Flip each droplet with probability: $1/2$ ($h = 0$)
- Repeat it over

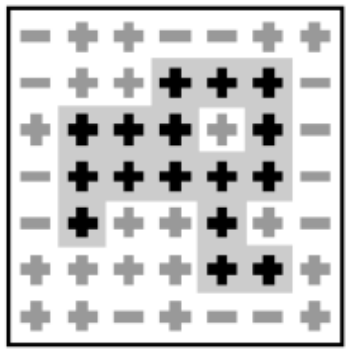
Wolff algorithm

1. Add a random spin to a list of active spins
2. Take a spin from the active list
3. Add each parallel neighboring (not yet visited) spin with probability $p_B = 1 - \exp(-2\beta J)$ to the list of active spins
4. If list of active spins is not empty go to 2.
5. Flip all active spins

A Wolff droplet (gray) before flipping



a



b

The new configuration The droplet contour is still shown, though the bonds are eliminated after flipping

Wolff code

```
bool **cluster;                          // cluster[i][j] = true if i,j belongs
double addProbability;                   // 1 - e^(-2J/kT)

void initializeClusterVariables() {

    // allocate 2-D array for spin cluster labels
    cluster = new bool* [Lx];
    for (int i = 0; i < Lx; i++)
        cluster[i] = new bool [Ly];

    // compute the probability to add a like spin to the cluster
    addProbability = 1 - exp(-2*J/T);
}
```

Wolff code

```
// declare functions to implement Wolff algorithm
void growCluster(int i, int j, int clusterSpin);
void tryAdd(int i, int j, int clusterSpin);

void oneMonteCarloStep() {

    // no cluster defined so clear the cluster array
    for (int i = 0; i < Lx; i++)
    for (int j = 0; j < Lx; j++)
        cluster[i][j] = false;

    // choose a random spin and grow a cluster
    int i = int(qadran() * Lx);
    int j = int(qadran() * Ly);
    growCluster(i, j, s[i][j]);

    ++steps;
}
```

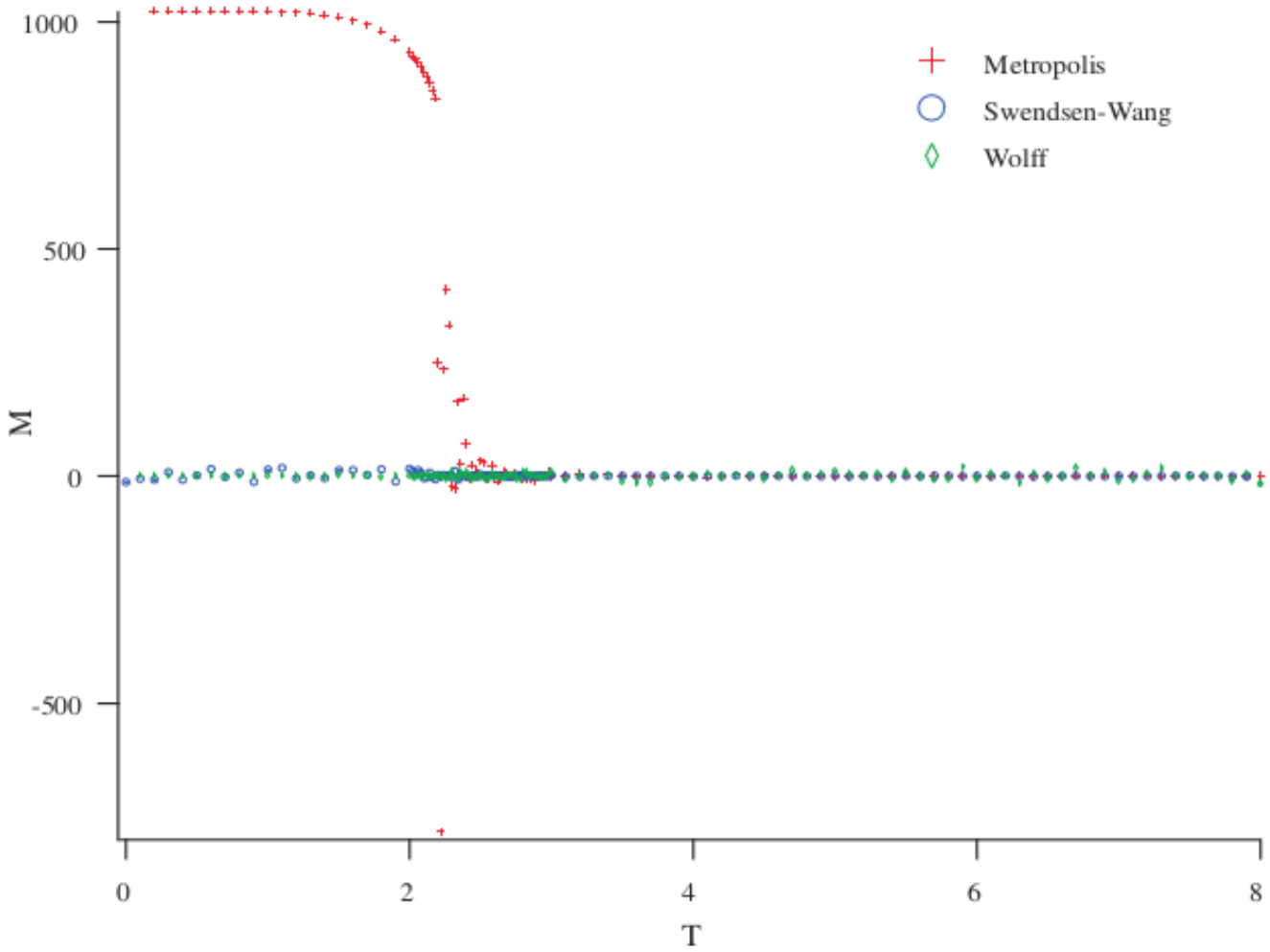
Wolff code

```
void growCluster(int i, int j, int clusterSpin) {

    // mark the spin as belonging to the cluster and flip it
    cluster[i][j] = true;
    s[i][j] = -s[i][j];

    // find the indices of the 4 neighbors
    // assuming periodic boundary conditions
    int iPrev = i == 0    ?  Lx-1 :  i-1;
    int iNext = i == Lx-1 ?  0    :  i+1;
    int jPrev = j == 0    ?  Ly-1 :  j-1;
    int jNext = j == Ly-1 ?  0    :  j+1;

    // if the neighbor spin does not belong to the
    // cluster, then try to add it to the cluster
    if (!cluster[iPrev][j])
        tryAdd(iPrev, j, clusterSpin);
    if (!cluster[iNext][j])
        tryAdd(iNext, j, clusterSpin);
    if (!cluster[i][jPrev])
        tryAdd(i, jPrev, clusterSpin);

   if (!cluster[i][jNext])
       tryAdd(i, jNext, clusterSpin);
}
```

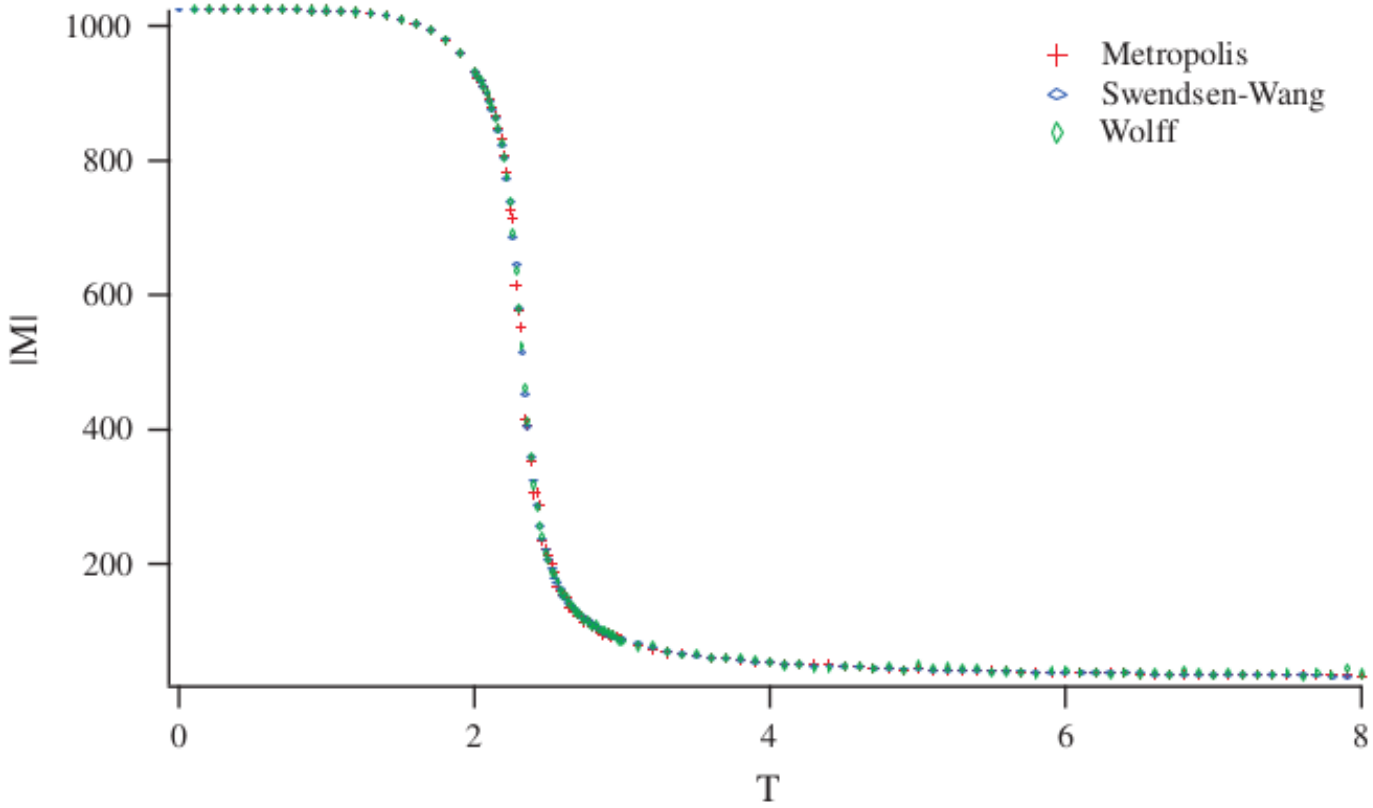
Wolff code

```
void growCluster(int i, int j, int clusterSpin) {

    // mark the spin as belonging to the cluster and flip it
    cluster[i][j] = true;
    s[i][j] = -s[i][j];

    // find the indices of the 4 neighbors
    // assuming periodic boundary conditions
    int iPrev = i == 0    ?  Lx-1 :  i-1;
    int iNext = i == Lx-1 ?  0    :  i+1;
    int jPrev = j == 0    ?  Ly-1 :  j-1;
    int jNext = j == Ly-1 ?  0    :  j+1;

    // if the neighbor spin does not belong to the
    // cluster, then try to add it to the cluster
    if (!cluster[iPrev][j])
        tryAdd(iPrev, j, clusterSpin);
    if (!cluster[iNext][j])
        tryAdd(iNext, j, clusterSpin);
    if (!cluster[i][jPrev])
        tryAdd(i, jPrev, clusterSpin);

   if (!cluster[i][jNext])
       tryAdd(i, jNext, clusterSpin);
}
void tryAdd(int i, int j, int clusterSpin) {
    if (s[i][j] == clusterSpin)
        if (qadran() < addProbability)
            growCluster(i, j, clusterSpin);
}
```

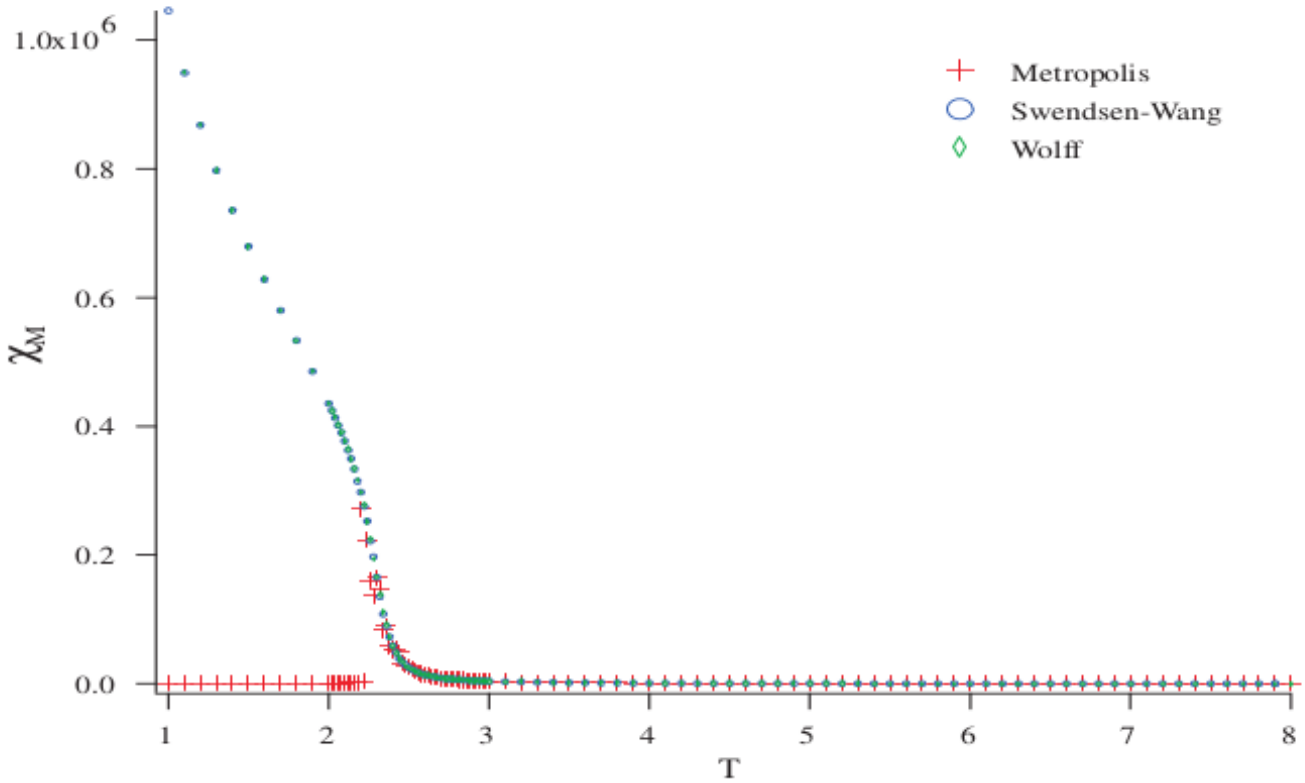
Comparison magnetization



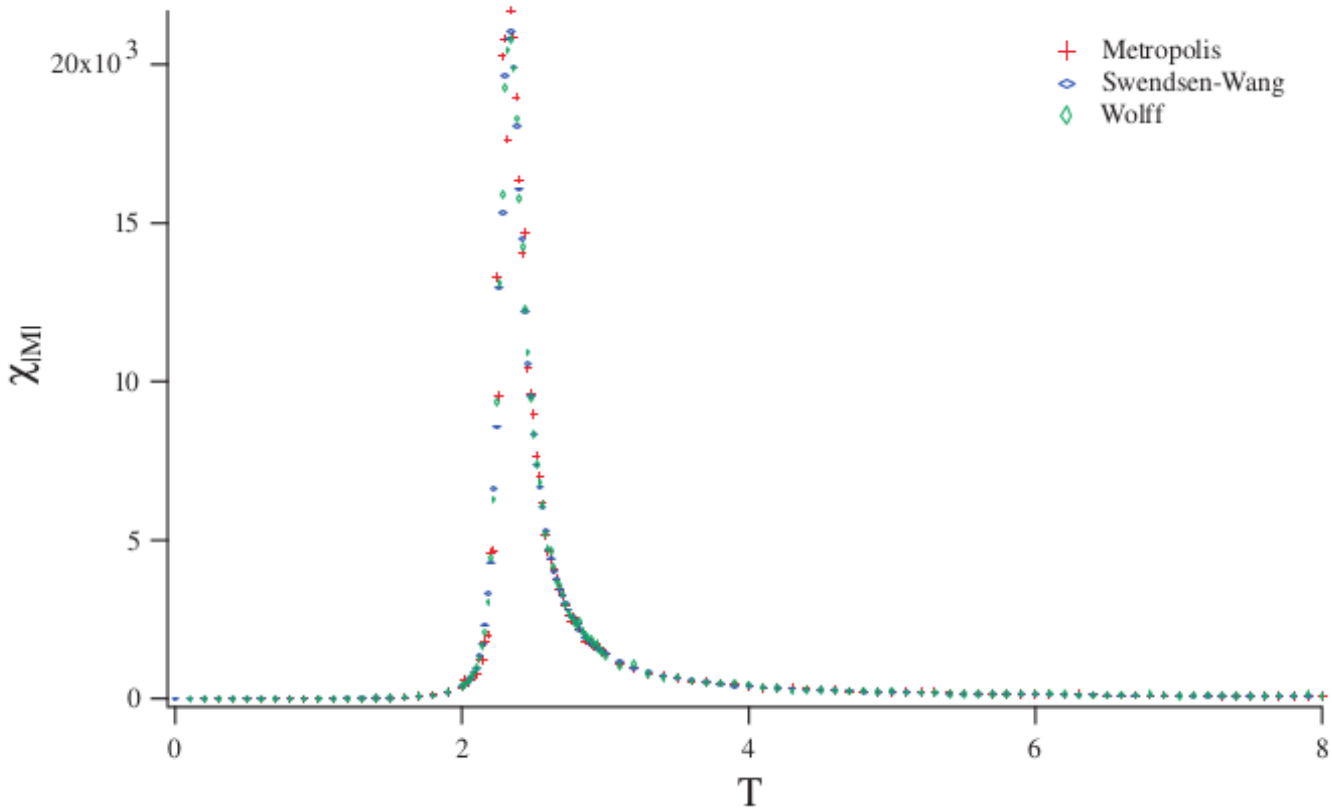
Comparison magnetization



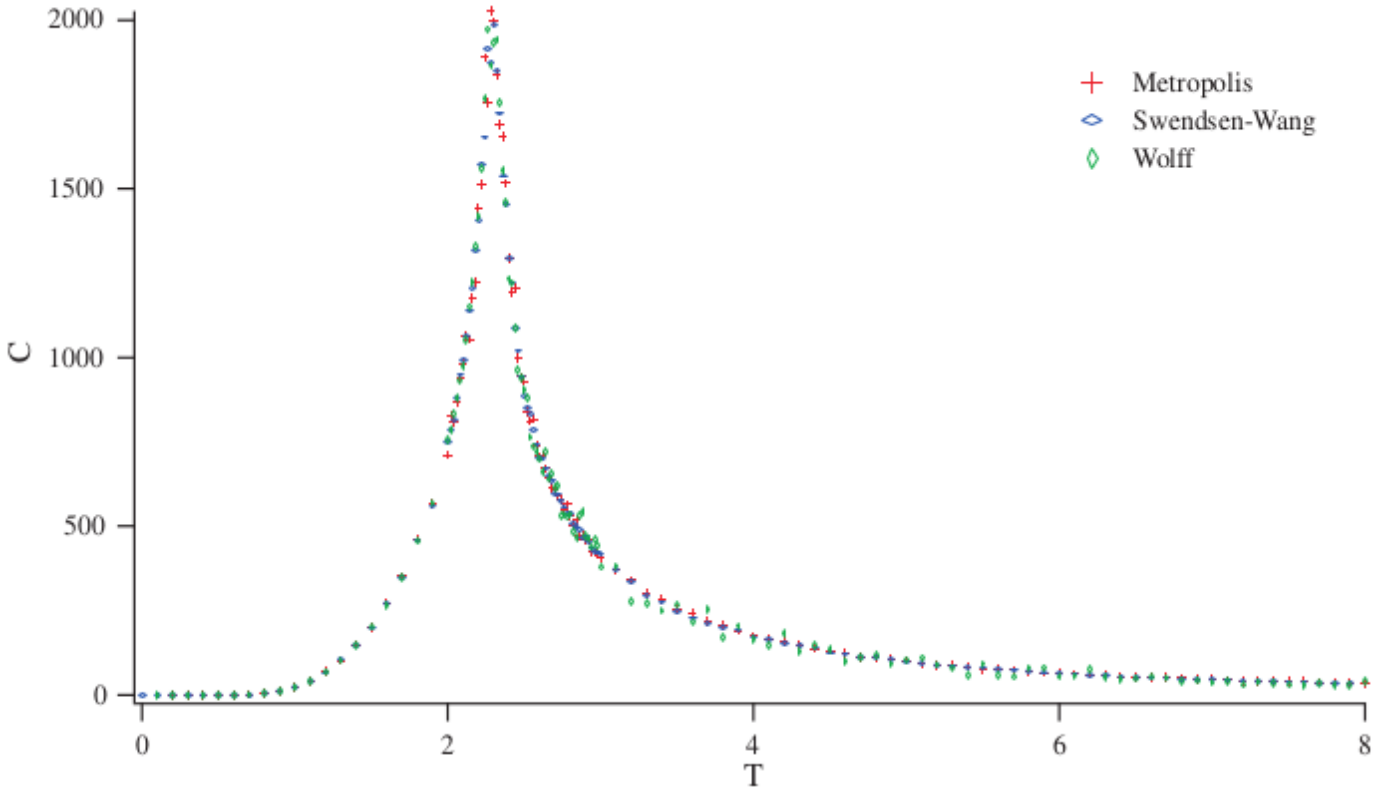
Comparison magnetization



Comparison magnetization



Comparison magnetization



Wolff algorithm *proof*

- Energy:
  - Exterior
  - Interior
  - Boundary
- We modify $g(x \to x')$ to be exactly $\exp(-\Delta E \beta)$ thus $P(x \to x') = 1$
- Good for parallelization!

Other ensembles

Microcanonical ensemble

- Daemon with bag with tolerance (both directions)
  - Pick a move, and calculate energy change
  - If energy change does not fit into bag reject it
  - Otherwise add energy change to bag
- In case of conservation the dynamic exponent z is larger!

Other ensembles

Conserved order parameter: Kawasaki dynamics

- Elementary step:
  - Exchange up-down spin pairs (can be anywhere) simultaneously
  - Apply Metropolis to net energy change!
  - Diffusive dynamics is more physical: pick neighboring spins
- In case of conservation the dynamic exponent z is larger!