Wolff Algorithm for Ising Model

（伊辛模型的沃尔夫算法）

曾祥东 & 孙晓晨

原作者：G. T. Barkema & M. E. J. Newman

Introduction——From 1D to 2D

Hamiltonian function:

$$H(\sigma) = -\sum_{<ij>} J_{ij}\sigma_i\sigma_j - \mu\sum_j h_{ij}\sigma_j$$

$< ij >$:

1D = 2

2D = 4

No phase transition in 1D, but in 2D.

Calculate What?

$$\tau = \xi^z$$

z —— 活性指数

a dynamic exponent whose value depends on the update method

τ —— 相关时间

a correlation time

ξ —— 相关长度

the correlation length

$$\xi \approx L^d$$

d ——维度

the dimensionality of the lattice

d = 2

L —— 大小

scale

Metropolis Algorithm

1. 设定 seed(excitation spins)
2. 自旋相同的点以概率 P_{add} 相互联系
3. 根据相邻点判断是否翻转与概率
4. 依次扩展重复（单个点的翻转）
5. Turn to 1.

Problem:
Critical slowing down at T_C

Wolff Algorithm

1. 设定 seed(excitation spins)
2. 自旋相同的点以概率 P_{add} 相互联系，建立 cluster
3. 每个点只能尝试与周围四个点链接，且仅有一次机会
4. 扩展 cluster
5. 当 cluster 无法扩展，翻转整个 cluster
6. Turn to 1.

Wolff Algorithm: Physics

$$\frac{g(\mu \to \nu)A(\mu \to \nu)}{g(\nu \to \mu)A(\nu \to \mu)} = (1 - P_{\mathrm{add}})^{m-n} \frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \mathrm{e}^{-\beta(E_\nu - E_\mu)}$$

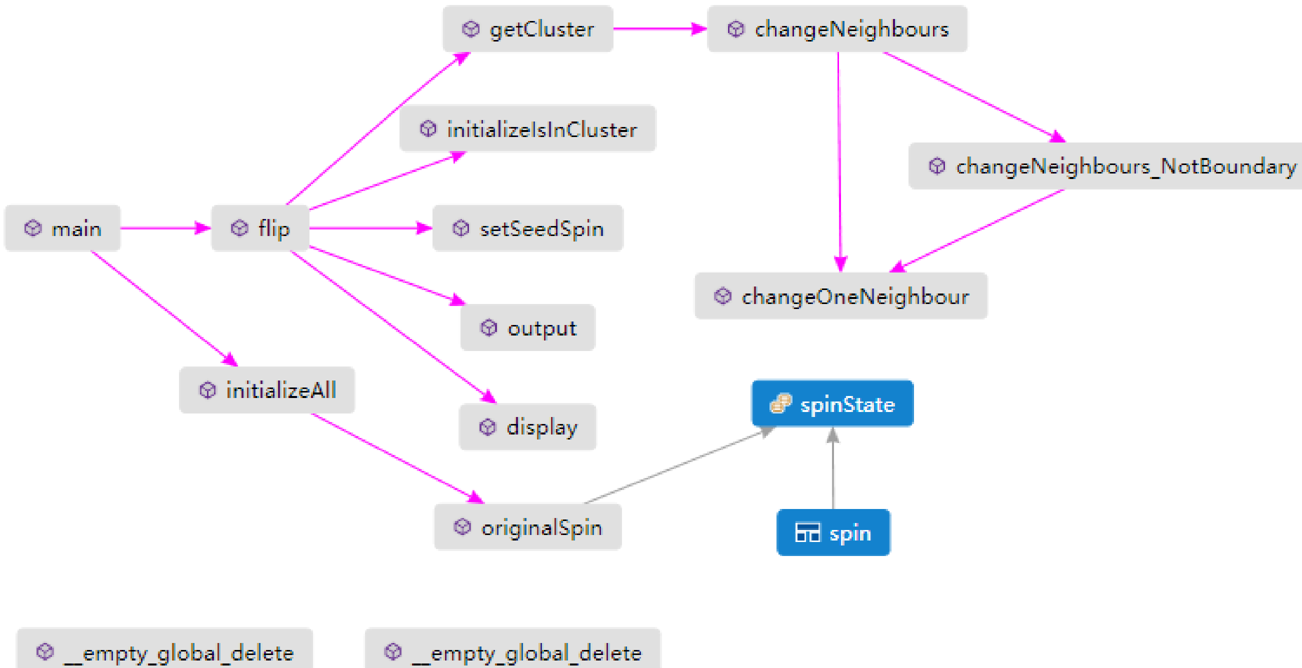
$$\frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \left[\mathrm{e}^{2\beta J}(1 - P_{\mathrm{add}})^{n-m}\right]$$

$$E_\nu - E_\mu = 2J(m - n)$$

$$P_{\mathrm{add}} = 1 - \mathrm{e}^{-2\beta J} = 1 - \mathrm{e}^{-\frac{2J}{k_{\mathrm{B}}T}}$$

Codes

Code Map



getCluster
changeNeighbours
initializeIsInCluster
changeNeighbours_NotBoundary
main
flip
setSeedSpin
changeOneNeighbour
output
initializeAll
display
spinState
originalSpin
spin
__empty_global_delete
__empty_global_delete

Code: spin

```
enum spinState
{
	SPIN_POS = 1,
	SPIN_NEG = -1
};

struct spin
{
	spinState state;			//SPIN_POS, SPIN_NEG
	bool isInCluster;			//1: in; 0: out
	bool hasChecked;			//1: checked; 0: not checked
};
```

Code: **setSeedSpin**()

```
void setSeedSpin()
{
      seed_X = rand() % $MAX_X;
      seed_Y = rand() % $MAX_Y;
      lattice[seed_X][seed_Y].isInCluster = true;
}
```

Code: **changeOneNeighbour()**

```
void changeOneNeighbour(const int &x, const int &y, int i, int j)
{
	long p = (rand() * rand()) % P_ADD_BASE;

	if (lattice[i][j].isInCluster == false && lattice[i][j].state ==
lattice[x][y].state && p < P_ADD)
	{
		lattice[i][j].isInCluster = true;
		++flag_isChangeNeighbours;
	}

	lattice[x][y].hasChecked = true;
}
```

Code: **getCluster**()

```
void getCluster()
{
	flag_isChangeNeighbours = 0;
	for (int i = 0; i < $MAX_X; ++i)
		for (int j = 0; j < $MAX_Y; ++j)
			if (lattice[i][j].isInCluster == true &&
lattice[i][j].hasChecked == false) changeNeighbours(i, j);
}
```

Code: **flip**() -1

```
void flip(ofstream &file, int &t)
{
      //**********Initialize time flags*********
      flag_isChangeNeighbours = 1;
      flag_time_checkSpin = 0;

      //**********Get cluster begin*********
      setSeedSpin();
      while (flag_isChangeNeighbours != 0)
            getCluster();
 ......
```

Code: **flip**() -2

……

```
//**********Flip begin**********
      for (int i = 0; i < $MAX_X; ++i)
            for (int j = 0; j < $MAX_Y; ++j)
                  if (lattice[i][j].isInCluster == true)
                  {
                        if (lattice[i][j].state == SPIN_POS)
lattice[i][j].state = SPIN_NEG;
                        else lattice[i][j].state = SPIN_POS;
                  }
```

……

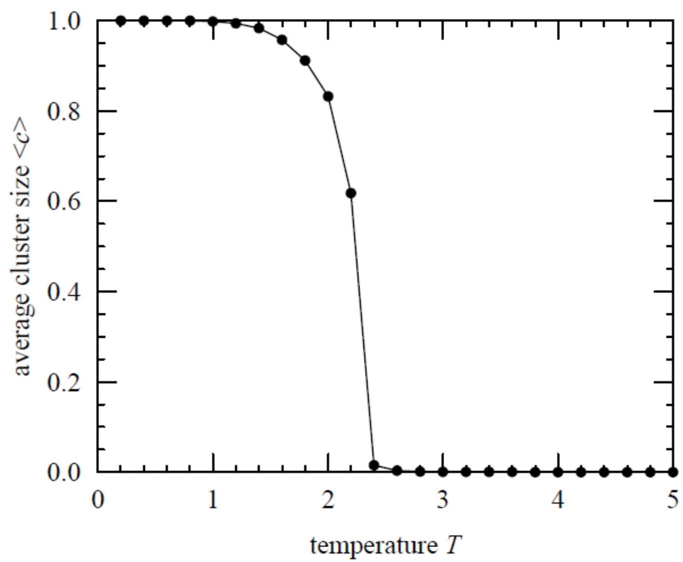
Code: **flip**() -3

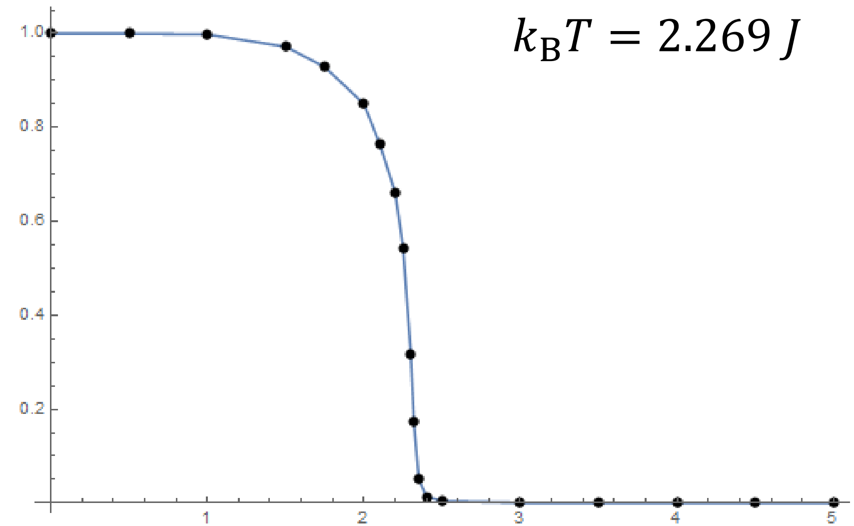
```
 ......
        //**********For next flip*********
        initializeIsInCluster();

        //**********Output & display*********
#ifdef _DISPLAY_OPEN
        display(t);
#endif
#ifdef _OUTPUT_OPEN
        if (t >= FLIP_TIME - 10) output(file);
#endif
}
```

Results

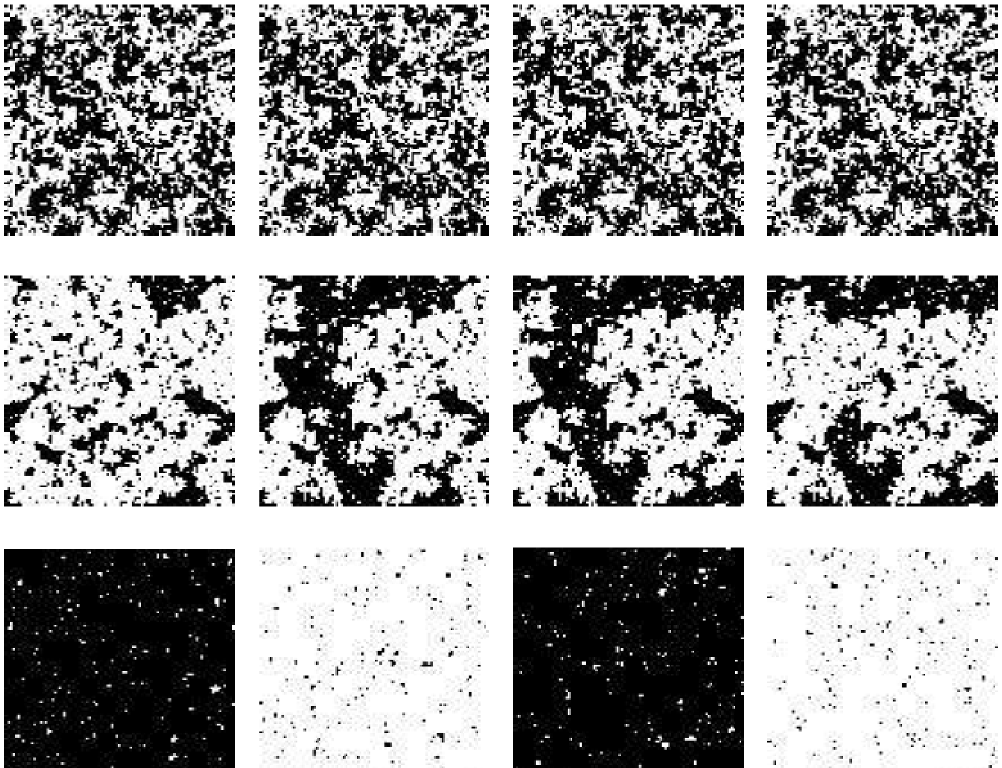
Mean Cluster Size vs Temperature





Theoretical solution:
$k_{\mathrm{B}}T = 2.269\,J$

States Plots



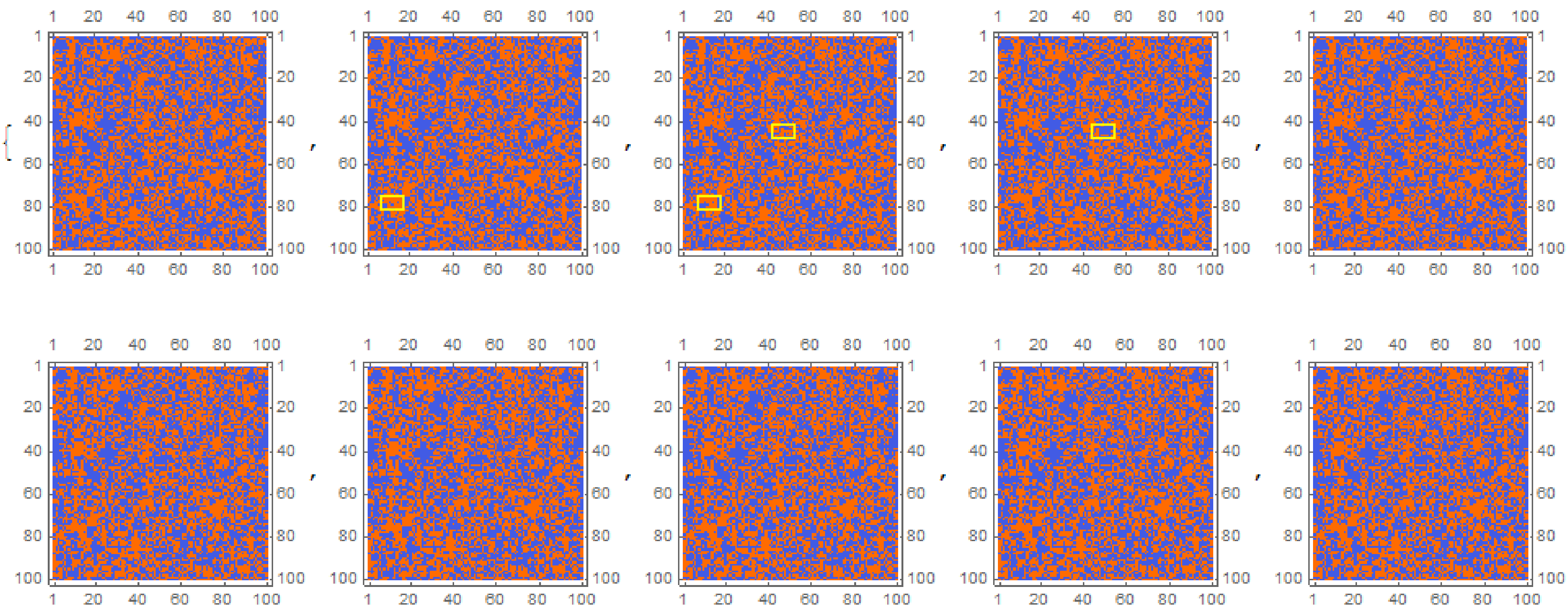
$k_{\mathrm{B}}T = 2.8\,J$

$k_{\mathrm{B}}T = 2.3\,J$

$k_{\mathrm{B}}T = 1.8\,J$

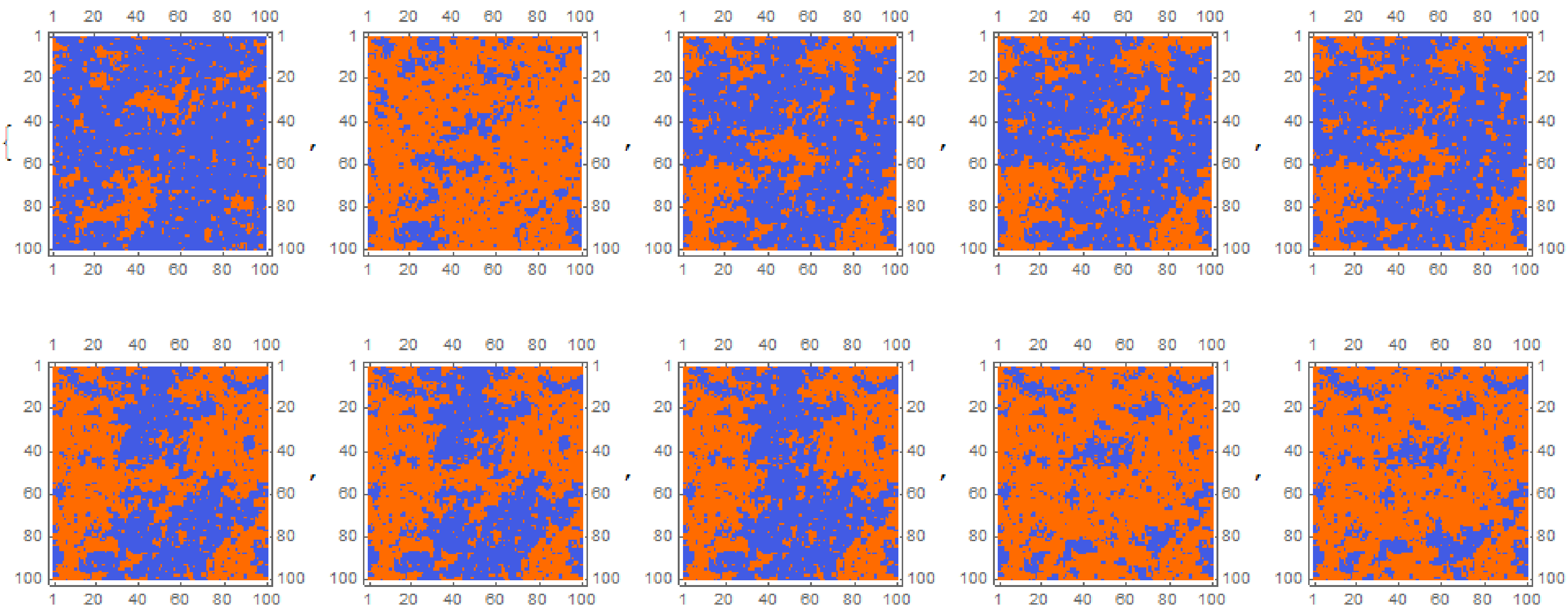
$k_{\mathrm{B}}T = 2.8\ J$

考验眼力!

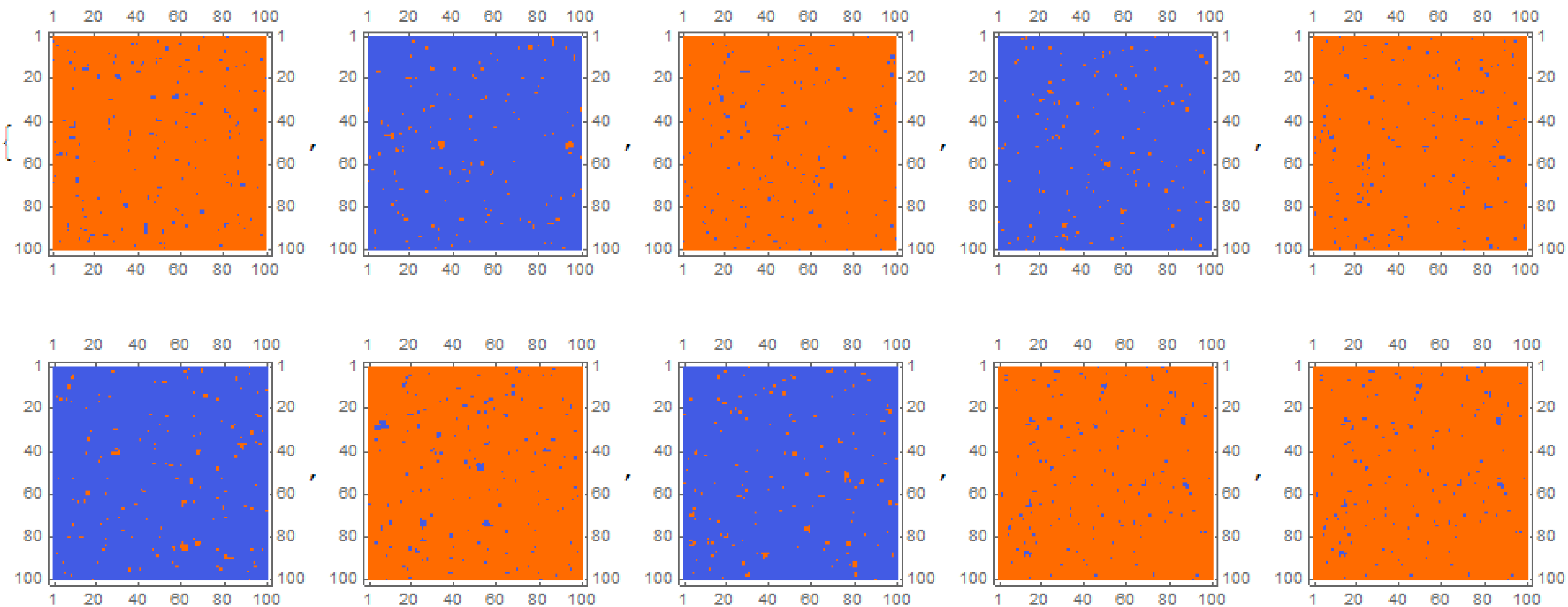


1000 steps, plots for last 10 steps.

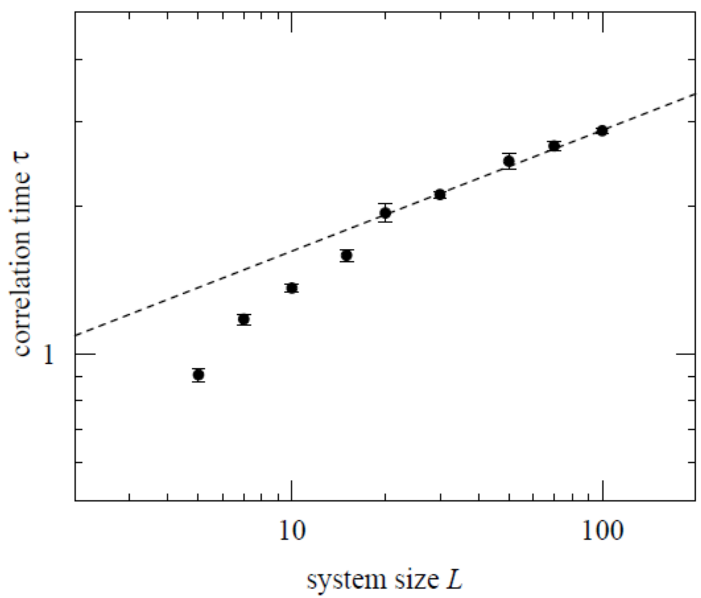
$k_B T = 2.3\ J$

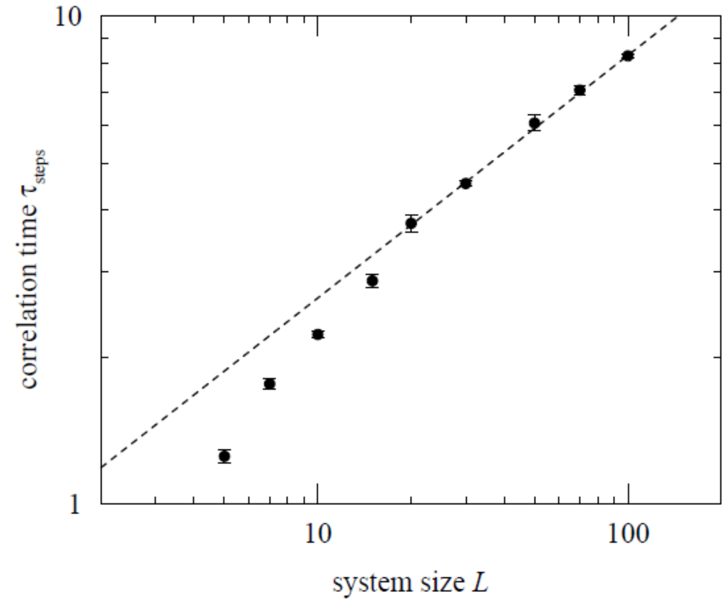


$k_B T = 1.8\ J$



Correlation Time vs System Size





But ours…

Swendsen-Wang Algorithm

1. 设定 seed(excitation spins)
2. 自旋相同的点以概率 P_{add} 相互联系，建立 cluster
3. 每个点只能尝试与周围四个点链接，且仅有一次机会
4. 扩展 cluster
5. 当 cluster 无法扩展，设定下一个 seed(excitation spins)
6. 当所有点都经历过扩展 cluster 过程的时候（每个点只属于一个cluster)，以 0.5 的概率翻转每个 cluster
7. Turn to 1.

Generalization

Algorithms:

- Wolff algorithm
- Swendsen-Wang algorithm
- Neidermayer's algorithm
- Limited cluster algorithm
- Multigrid algorithm

Models:

- Potts models
- Continuous spin models

Thank you!

Thanks for Xiao Chuan.