The Wolff Algorithm with External Sources and Boundaries

Jorge L. deLyra
Department of Mathematical Physics
Physics Institute
University of São Paulo

Version 1, July 2006

We will consider here the technical question of how to implement the Wolff cluster algorithm for the Ising model in the presence of arbitrary external sources. We will also consider the use of the algorithm in lattices with fixed rather than periodical boundary conditions, as well as the use of truncated clusters. We will not deal with the technical aspects relating to how well the algorithm itself works, with respect to the elimination of the phenomenon of critical slowing down. For that, the reader is referred to the original papers on the cluster algorithms [1, 2].

1 The Wolff Algorithm

The traditional Metropolis algorithm, when applied do the Ising models close to their critical points, suffers from severe critical slowing down problems. This is due to its single-site updating procedure, which is good for updating the short-wavelength components of the configurations, but very bad at updating the long-wavelength components. Near the critical point the long-wavelength components play an important role due to the emergence of long-distance correlations, and in these circumstances the algorithm tends to diffuse only very slowly through the configuration space of the model. This increases the autocorrelations along the stochastic sequence of configurations and severely degrades the statistical quality of the results.

This is always a problem is the case of quantum field theory, since in that theory we are interested primarily in the critical region of the statistical models. If we are to use the Ising model in the analysis of problems in quantum field theory, we must use simulation techniques that do not suffer from such problems. The cluster algorithms are the answer to our needs, and among them the Wolff algorithm is particularly well suited for the task, due to its simplicity and efficiency. Cluster algorithms are characterized by the updating of whole sets of sites, or clusters, at a time, and in doing this they solve the problem of critical slowing down. The problem of how to make collective updates with a low rate of rejection and in such a way that the required statistical distribution is still produced correctly is a very difficult one, which was solved by the inventors of the cluster algorithms.

In this section we will describe the basic algorithm and show that it satisfied the condition of detailed balance, thus converging to the correct distribution. After that we will generalize the algorithm to the case in which we introduce external sources in the systems, as well as to the case in which we use fixed rather than the more usual periodical boundary conditions. This last topic will also lead us to consider the use of truncated clusters, a variation of the algorithm which may be useful under some circumstances.

Let us start by reviewing the fundamentals of the Metropolis algorithm. In order to do that we must first establish some notation. Consider a model in quantum field theory defined by an action functional $S[\psi]$, where ψ is the field. This action determines the distribution of probabilities for all the possible configurations C of the field, which is given by $P_B[C] = Z \exp(-S[C])$, where Z is a normalization constant and the dependence on the field can be represented either by $S[\psi]$ or by $S[C]$. What a simulation algorithm does is to establish a probability $W[C \to C']$ for going from a configuration C to a configuration C' on a single iteration of the algorithm. By iterating the algorithm indefinitely one produces a sequence of configurations with a distribution of probabilities that is supposed to approach $P_B[C]$.

In order for this to work, it suffices that the algorithm satisfy two conditions: ergodicity and detailed balance. Although not strictly necessary, these conditions are sufficient to ensure convergence to the correct distribution. Ergodicity means that any configuration C' should be reachable from any other configuration C in a finite number of iterations of the algorithm. In other words, the algorithm should never get stuck in any particular configuration that it is unable to leave, and there should be no configuration that it cannot reach from some other configuration. This means that the algorithm should be able to diffuse through the whole space of possible configurations of the field, without leaving out any points or regions within it. Note that no statement is made here about how fast it is able to diffuse. Of course, in practice the quality of the algorithm will depend on it being able to do so fast, because the space of configurations is usually very large.

The condition of detailed balance is the one which will mostly concern us here. It is a relation between the probability $W[C \to C']$ of going from the configuration C to the configuration C' and the probability $W[C' \to C]$ of doing the reversed update, that is, of going from C' to C, and involves the probabilities $P[C]$ and $P[C']$ of the two configurations involved, within the ensemble of all possible configurations. It states that the ratio of these two probabilities should satisfy the condition

$$\frac{W[C \to C']}{W[C' \to C]} = \frac{P[C']}{P[C]}.$$

Since the probabilities of the configurations can be written in terms of the action, this condition can be translated into

$$\frac{W[C \rightarrow C']}{W[C' \rightarrow C]} = \frac{\exp(-S[C'])}{\exp(-S[C])} = \exp(-S[C'] + S[C]) = \exp(-\Delta S),$$

where ΔS is the change in the action due to the update of the configuration. It is possible to show by elementary means, in a simple one-dimensional case, that this condition is sufficient to guarantee that the set of configurations produced by the algorithm converges to the correct distribution. However, that is not our focus here, and we will simply take it as a given fact. Let us now describe in detail the usual Metropolis algorithm for a single local update in the Ising model, and show that it satisfies this condition. Starting from some initial configuration, the Metropolis single-site updating algorithm is given by the following sequence of steps:

1. Choose a single site of the lattice for trying an update, for example in a random way.

2. Calculate the variation ΔS of the action caused by flipping the field at that site.

3. If it happens that $\Delta S \leq 0$, then flip the field with probability equal to 1.

4. If $\Delta S > 0$, then flip the field with probability $\exp(-\Delta S)$, which in this case is a number in the interval $(0, 1)$; if the update is rejected, repeat the current configuration in the stochastic sequence.

5. Loop back to step 1 and repeat the procedure.

As one can see, the algorithm is very simple and easily generalizable to more complex models. It is also possible to improve it in various rather technical ways, that help in some cases but that do not solve the central problem of critical slowing down caused by the local update. It is now easy to verify that this algorithm satisfies the condition of detailed balance. We must consider three cases, depending on whether the variation of the action is positive, zero or negative. If the update is such that $\Delta S > 0$, then we have for its probability $W[C \rightarrow C'] = \exp(-\Delta S)$; in this case the reversed update is such that $\Delta S < 0$, so that we have for its probability $W[C' \rightarrow C] = 1$; it therefore follows that

$$\frac{W[C \rightarrow C']}{W[C' \rightarrow C]} = \frac{\exp(-\Delta S)}{1} = \exp(-\Delta S),$$

as required by the condition. If the update is such that $\Delta S < 0$, then the probability for the update is $W[C \rightarrow C'] = 1$, and the probability for the reversed update is $W[C \rightarrow C'] = \exp(-\Delta' S)$, where $\Delta' S = -\Delta S$ is the variation of the action in the reversed update; it therefore follows that

$$\frac{W[C \rightarrow C']}{W[C' \rightarrow C]} = \frac{1}{\exp(\Delta S)} = \exp(-\Delta S),$$

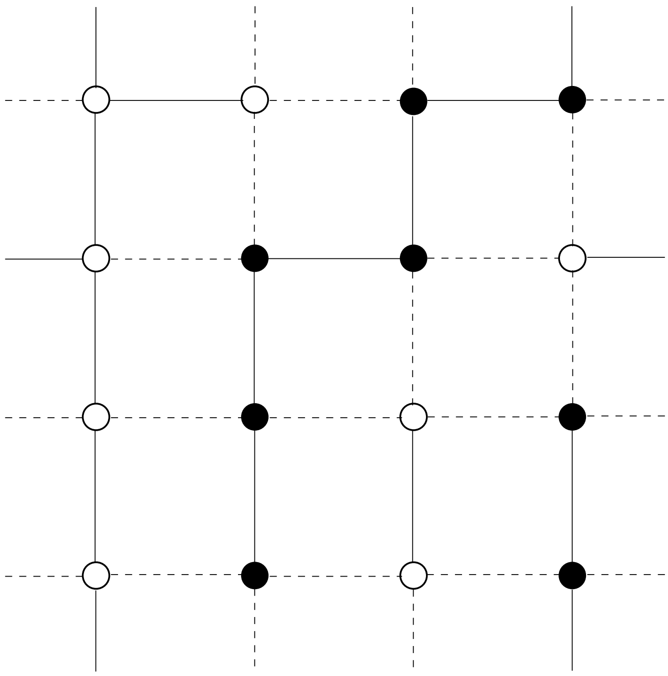

Figure 1: A periodical lattice with the sites and links classified, showing the clusters that may be built in it.

just as in the previous case. If $\Delta S = 0$, then we also have $\Delta' S = 0$, and hence $W[C \to C'] = W[C' \to C] = 1$, so that the ratio of the two updating probabilities is equal to 1; since in this case $\exp(-\Delta S) = \exp(0)$ is also equal to 1, we see that in all three cases the condition is satisfied.

In order to discuss the Wolff algorithm, we must now establish a few more concepts and notations. The basic idea of the algorithm will be to first built a cluster of sites according to certain rules, and then to flip the whole cluster. The cluster will be built by the "activation" of some of the links between two neighboring sites, and requires that we classify the links according to whether it connects two sites where the field has the same orientation, or opposite orientations. At first, let us consider only lattices with periodical boundary conditions and the model in the absence of any external sources. Later on we will extend the algorithm to these other cases. We should recall here that the Ising model is defined by the action

$$S[\psi] = -\beta \sum_{\ell} \psi_- \psi_+,$$

where $0 < \beta < \infty$ is the free parameter of the model, the sum is over the links and ψ_-, ψ_+ are the values of the field at the two ends of a given link. In figure 1 one can see an example of a two-dimensional periodical lattice with all its sites and links dully classified. The sites where the field is positive are shown as black circles and those where the field is negative as white circles. The links that connect two sites with the same orientation are shown as solid lines, and those connecting sites with

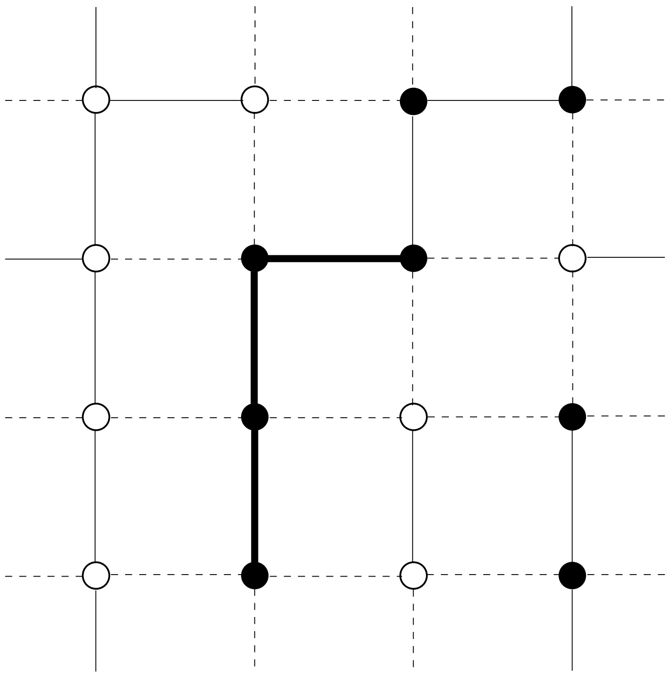

Figure 2: The same periodical lattice, showing a possible cluster that might have been built in it.

opposite orientations as dashed lines. Note that this is true for the border links as well, which establish the periodical boundary conditions. The links represented by solid lines will be denoted by ℓ_+ in the formulas, and those represented by dashed lines by ℓ_-. According to the algorithm only ℓ_+ links can be activated to form clusters, so the sets of sites connected by solid lines illustrate the clusters that can be formed in this particular configuration. We may now state the Wolff algorithm, which is given by the following sequence of steps:

1. Choose a single site of the lattice for starting to build the cluster, in a random way.

2. Consider all the links connected to that initial site; the ℓ_- links are never to be activated; activate the ℓ_+ links with the probability $p_+ = 1 - \exp(-2\beta\psi_-\psi_+)$, thus forming a first cluster of sites, that is, updating the cluster from a single site to possibly a few sites.

3. Given the set of sites added to the cluster in the *previous* update of the cluster (which is *not* the whole cluster), consider all the links that connect those sites to sites that are still outside the cluster; among these, activate the ℓ_+ links with the probability p_+, thus enlarging (updating) the cluster.

4. Loop back to step 3 until the set of links left for activation trial in the next round of the loop is empty; this means that you stop this loop at the first instance of step 3 that adds no new sites to the cluster.

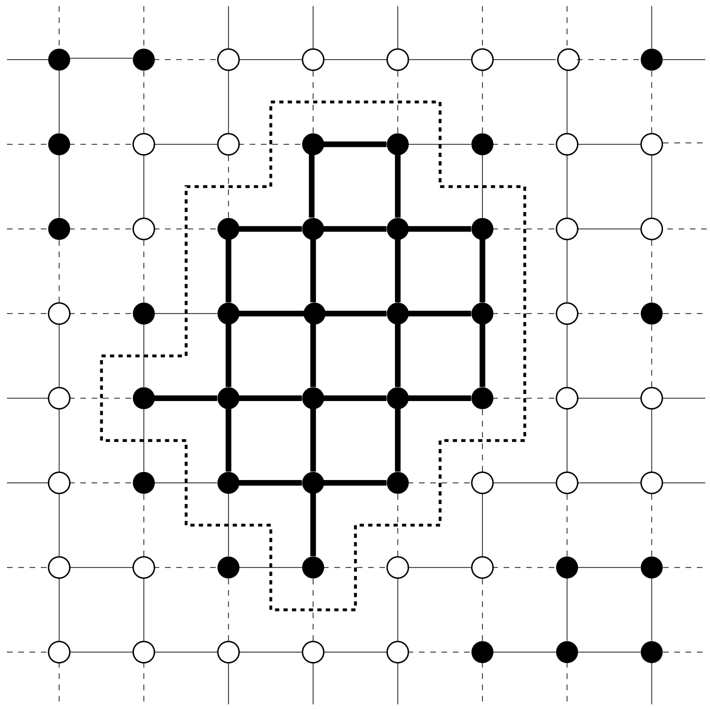

Figure 3: A larger lattice, showing a cluster and its boundary.

5. Flip the whole cluster with probability 1, then loop back to step 1, in order to build and flip a new cluster.

Note that in steps 2 and 3, since only ℓ_+ links can be activated, the argument of the exponential is negative and thus $\exp(-2\beta\psi_-\psi_+)$ is a number in the interval $(0, 1)$. Therefore, so is the probability $p_+ = 1 - \exp(-2\beta\psi_-\psi_+)$. Note also that during the construction of the cluster a certain site may be approached by the growing cluster from two or more different directions, along different links. If that site does not get linked to the cluster at the first trial, it might still get linked at a later trial. The algorithm implies that the same link is never tried more than once for activation, but a single site my be tried more than once for linking to the cluster, if it has more than one link able to be activated. Finally, observe that during construction it might happen that two sites already included in the cluster are connected by a link that has not yet been tried. That link will in fact never be tried by the algorithm, and may be considered as automatically activated, that is, they can simply be activated with probability 1 at the end of the construction. A simple cluster that could result from the algorithm is shown in figure 2, with its sites and activated links, which are shown in a thicker solid line.

Let us now show that this algorithm satisfied the condition of detailed balance. In order to do this, given a certain cluster, we must calculate the probability that it will be built and flipped. We will denote the cluster by $\mathcal{C}$ and the boundary of the cluster, that is, the set of links that connect the cluster with sites outside the cluster, by $\partial\mathcal{C}$. In figure 3 one can see a larger lattice with a cluster that could have been built by the algorithm, as well as its boundary, illustrated by the thick dotted line which crosses all the links that connect the cluster to sites outside the cluster. Since the probability of flipping a cluster is 1, once it has been built, we must simply calculate the total probability of building a certain given cluster. The process of building the cluster might start at any of its sites, so we start the calculation with the probability that one of the sites of the cluster will be chosen in step 1 of the algorithm. Let us call this probability p_i, and there is no need to further elaborate on it.

Irrespective of where the construction starts, the probability that the given cluster will be built is the product of the probabilities that each one of its internal links will be activated, times the product of the probabilities that each one of the links of its borders will *not* be activated. Now, all the internal links are ℓ_+ links, so the probability that they will be activated is given by

$$\prod_{\ell_+ \in \mathcal{C}} p_+.$$

For the ℓ_+ links in the border of the cluster *not* to be activated we have the probability

$$\prod_{\ell_+ \in \partial\mathcal{C}} q_+,$$

where $q_+ = 1 - p_+ = \exp(-2\beta\psi_-\psi_+)$ is the complementary probability to p_+. Since the probability for the ℓ_- links at the border not to be activated is 1, this product is the complete probability for the construction of the boundary. We have therefore the complete probability for the construction of the cluster, and hence the complete probability for the update of the configuration,

$$W[C \to C'] = p_i \left(\prod_{\ell_+ \in \mathcal{C}} p_+ \right) \left(\prod_{\ell_+ \in \partial\mathcal{C}} q_+ \right).$$

Next we must consider the probability for the reversed update. Since the same set of sites that makes up the cluster $\mathcal{C}$ in the configuration C also makes up the flipped cluster $\mathcal{C}'$ in the configuration C', it is clear that the probability p_i is the same in either case. Now, repeating the argument used for the cluster $\mathcal{C}$ we get for the probability that the cluster $\mathcal{C}'$ will be built

$$W[C' \to C] = p_i \left(\prod_{\ell_+ \in \mathcal{C}'} p_+ \right) \left(\prod_{\ell_+ \in \partial\mathcal{C}'} q_+ \right).$$

Observe now that when we flip the fields certain mappings between links take place. When we go from the cluster $\mathcal{C}$ to the flipped cluster $\mathcal{C}'$, we can see that all the internal ℓ_+ links of $\mathcal{C}$ become internal ℓ_+ links of $\mathcal{C}'$, since the fields at both ends of each link get flipped and hence their product does not change sign. This means that the first products that appears in the two formulas above are equal, and therefore that we may write the ratio of the two probabilities as

$$\frac{W[C \to C']}{W[C' \to C]} = \frac{\prod_{\ell_+ \in \partial\mathcal{C}} q_+}{\prod_{\ell_+ \in \partial\mathcal{C}'} q_+}.$$

For the links at the border, however, we have that the ℓ_+ links of $\partial\mathcal{C}$ become the ℓ_- links of $\partial\mathcal{C}'$, and that the ℓ_- links of $\partial\mathcal{C}$ become the ℓ_+ links of $\partial\mathcal{C}'$, since for these links only the field at one end gets flipped and therefore the sign of the product changes. In addition to this, the term of the action corresponding to each link changes sign, so that the factor q_+ gets mapped to $1/q_+$. If we write the ratio above in terms of the action terms,

$$\frac{W[C \to C']}{W[C' \to C]} = \frac{\prod_{\ell_+ \in \partial\mathcal{C}} \exp(-2\beta\psi_-\psi_+)}{\prod_{\ell_+ \in \partial\mathcal{C}'} \exp(-2\beta\psi_-\psi_+)},$$

we see that we can decompose each one of the two products in two parts,

$$\frac{W[C \to C']}{W[C' \to C]} = \frac{\prod_{\ell_+ \in \partial\mathcal{C}} \exp(-\beta\psi_-\psi_+)}{\prod_{\ell_+ \in \partial\mathcal{C}'} \exp(-\beta\psi_-\psi_+)} \, \frac{\prod_{\ell_+ \in \partial\mathcal{C}} \exp(-\beta\psi_-\psi_+)}{\prod_{\ell_+ \in \partial\mathcal{C}'} \exp(-\beta\psi_-\psi_+)}.$$

Now, due to the mappings of the boundary links which we just discussed, we have that the following two equations hold,

$$\begin{aligned}
\prod_{\ell_+ \in \partial\mathcal{C}} \exp(-\beta\psi_-\psi_+) &= \prod_{\ell_- \in \partial\mathcal{C}'} \exp(\beta\psi_-\psi_+), \\
\prod_{\ell_+ \in \partial\mathcal{C}'} \exp(-\beta\psi_-\psi_+) &= \prod_{\ell_- \in \partial\mathcal{C}} \exp(\beta\psi_-\psi_+),
\end{aligned}$$

so that the ratio of updating probabilities may be written as

$$\begin{aligned}
\frac{W[C \to C']}{W[C' \to C]} &= \frac{\prod_{\ell_+ \in \partial\mathcal{C}} \exp(-\beta\psi_-\psi_+)}{\prod_{\ell_+ \in \partial\mathcal{C}'} \exp(-\beta\psi_-\psi_+)} \, \frac{\prod_{\ell_- \in \partial\mathcal{C}'} \exp(\beta\psi_-\psi_+)}{\prod_{\ell_- \in \partial\mathcal{C}} \exp(\beta\psi_-\psi_+)} \\
&= \frac{\prod_{\ell_+ \in \partial\mathcal{C}'} \exp(\beta\psi_-\psi_+)}{\prod_{\ell_+ \in \partial\mathcal{C}} \exp(\beta\psi_-\psi_+)} \, \frac{\prod_{\ell_- \in \partial\mathcal{C}'} \exp(\beta\psi_-\psi_+)}{\prod_{\ell_- \in \partial\mathcal{C}} \exp(\beta\psi_-\psi_+)}.
\end{aligned}$$

Note that in this way we have completed the set of boundary links in both the numerator and the denominator, so that we may write

$$\frac{W[C \to C']}{W[C' \to C]} = \frac{\prod_{\ell \in \partial\mathcal{C}'} \exp(\beta\psi_-\psi_+)}{\prod_{\ell \in \partial\mathcal{C}} \exp(\beta\psi_-\psi_+)} = \frac{\exp(\sum_{\ell \in \partial\mathcal{C}'} \beta\psi_-\psi_+)}{\exp(\sum_{\ell \in \partial\mathcal{C}} \beta\psi_-\psi_+)}.$$

What we see here in the argument of the two exponentials in the last form of this equation is the negative of the part of the action that corresponds to the boundary links, which we may denote by $S_B = -\sum_{\ell\in\partial\mathcal{C}} \beta\psi_-\psi_+$. We may therefore write the ratio of the two updating probabilities as

$$\frac{W[C \to C']}{W[C' \to C]} = \frac{\exp(-S_B[C'])}{\exp(-S_B[C'])} = \exp(-\Delta S_B).$$

Since in the flipping of a cluster the parts of the action corresponding to links that are either completely internal to the cluster or completely external to it do not change at all, the only variation of the action is that due to the boundary links, which means that $\Delta S = \Delta S_B$, and hence we get the final relation

$$\frac{W[C \to C']}{W[C' \to C]} = \exp(-\Delta S),$$

which is indeed the condition of detailed balance, as we set out to show. Please note how important it is for this demonstration the fact that the update of a cluster changes the action exclusively at the boundary links of the cluster, and not at its bulk, that is, at its interior sites and links.

2 Arbitrary External Sources

Let us now discuss what happens when we introduce external sources into the system. In this case we have a new term in the action, which becomes

$$S[\psi] = S_0[\psi] + S_\eta[\psi] = -\beta \sum_\ell \psi_-\psi_+ - \sum_s \eta(s)\psi(s),$$

where the sum in the new term denoted as $S_\eta[\psi]$ is over all the sites of the lattice. The external source $\eta(s)$ is a given, fixed but otherwise arbitrary function of the sites. It is not simply a new parameter of the model because, unlike β, it may change from site to site. It is not a dynamical variable either, since, unlike ψ, it does not fluctuate according to a statistical ensemble. It is what we may call a classical (non-quantum) variable introduced into the model in this way.

The issue at hand now, regarding the stochastic simulations, is how to update the configurations so as to converge to the ensemble given by $Z\exp(-S[\psi])$, with this new action. At a first glance one may think that the most natural thing to do is to modify the algorithm in order to take the new term into account while building the cluster, but one can quickly verify that this does not work, due to the fact that the flipping of the cluster changes this new term at the interior sites of the cluster. Under such circumstances the algorithm no longer satisfies the condition of detailed balance.

The only known alternative left open to us is to modify the probability of acceptance of the clusters, built exactly as before, taking only the term $S_0[\psi]$ into consideration. This means that, instead of flipping the cluster with probability 1, once it is built, we will flip it with a probability that depends on $S_\eta[\psi]$, using a version of the usual Metropolis algorithm at the end of the Wolff algorithm. Here is the complete algorithm with this modification:

1. Choose a single site of the lattice for starting to build the cluster, in a random way.

2. Consider all the links connected to that initial site; the ℓ_- links are never to be activated; activate the ℓ_+ links with the probability $p_+ = 1 - \exp(-2\beta\psi_-\psi_+)$, thus forming a first cluster of sites, that is, updating the cluster from a single site to possibly a few sites.

3. Given the set of sites added to the cluster in the *previous* update of the cluster (which is *not* the whole cluster), consider all the links that connect those sites to sites that are still outside the cluster; among these, activate the ℓ_+ links with the probability p_+, thus enlarging (updating) the cluster.

4. Loop back to step 3 until the set of links left for activation trial in the next round of the loop is empty; this means that you stop this loop at the first instance of step 3 that adds no new sites to the cluster.

5. Calculate the variation ΔS_η of the external-source term of the action caused by flipping the cluster just built.

6. If it happens that $\Delta S_\eta \leq 0$, then flip the cluster with probability equal to 1.

7. If $\Delta S_\eta > 0$, then flip the cluster with probability $\exp(-\Delta S_\eta)$; if the update is rejected, repeat the current configuration in the stochastic sequence.

8. Loop back to step 1, in order to build and possibly flip a new cluster.

It is apparent at once that this algorithm is inevitably somewhat less efficient than the pure Wolff algorithm, in terms of diffusion speed through configuration space, but the important thing is that is still free from the severe critical slowing down problems of the Metropolis algorithm. It is not difficult to check that the modified algorithm satisfies the condition of detailed balance. Since the cluster is built just like before, we already know that the ratio of the corresponding building probabilities is given by

$$\frac{W_0[C \to C']}{W_0[C' \to C]} = \exp(-\Delta S_0).$$

We must now multiply this by the flipping probability, which can easily be obtained from our previous argument regarding the Metropolis algorithm,

$$\frac{W_\eta[C \to C']}{W_\eta[C' \to C]} = \exp(-\Delta S_\eta),$$

so that we have for the ratio of the complete probabilities $W = W_0 W_\eta$,

$$\frac{W[C \to C']}{W[C' \to C]} = \exp(-\Delta S_0 - \Delta S_\eta) = \exp(-\Delta S),$$

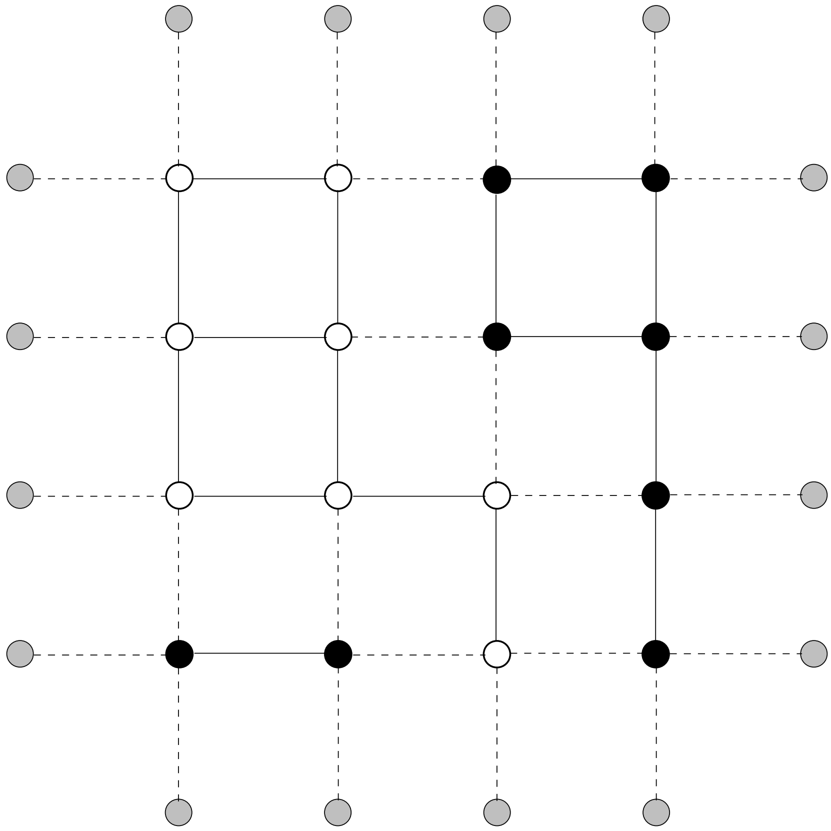

Figure 4: A fixed-boundary lattice with the sites and links classified, showing the clusters that may be built in it, as well as the boundary sites.

which is the condition of detailed balance, thus showing that the algorithm does work. Note, however, that there is no way to control the rejection rate of the clusters built, because the variation of S_η will depend on the average size of the clusters and we have no control over that. It is possible, therefore, that the algorithm will become very inefficient under certain conditions, with a high rejection rate, such as, for example, in the case of a very large external source which is constant over the whole lattice.

3 Fixed Boundary Conditions

Let us now discuss what happens when we use a lattice with fixed instead of periodical boundary conditions. In this case the lattice has a boundary, where the field is kept at fixed values, which are not necessarily constant over the boundary, and which, even in the Ising model, are not necessarily just ± 1. In figure 4 one can see an example of such a lattice, with all its sites and links dully classified. The sites at the external boundary are shown in gray. Note that all the links to the sites at the boundary are represented by dashed lines, because one cannot ever activate these links, since that would force us to flip the fields at the boundary, and of course

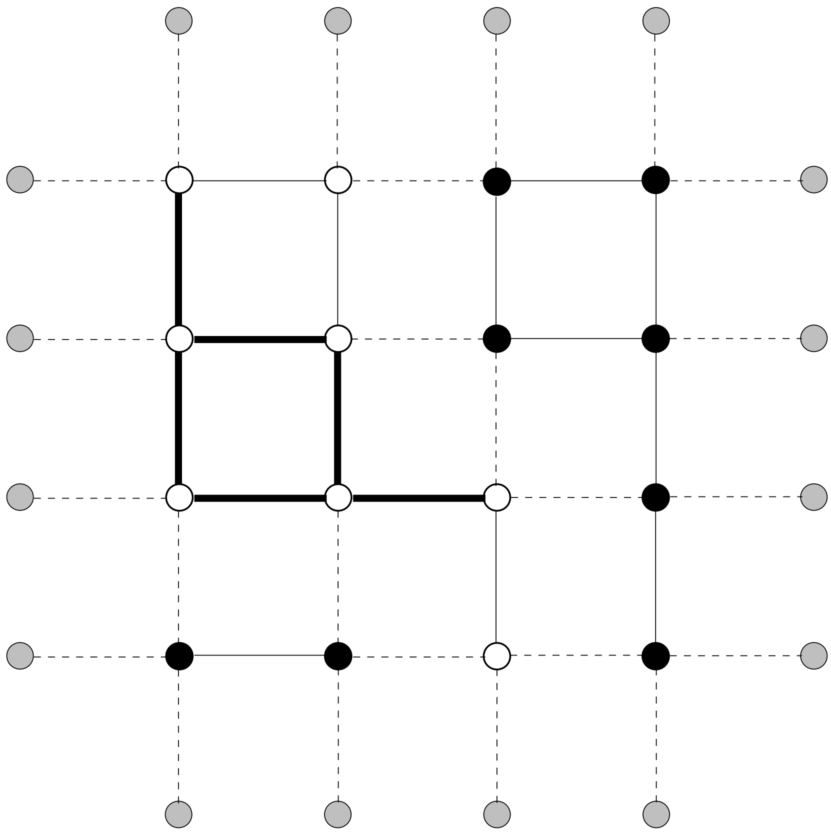

Figure 5: The same fixed-boundary lattice, showing a possible cluster that might have been built in it, with part of it connected to the boundary.

changing the fields at the boundary is not allowed under these circumstances.

One can see without difficulty that one can use the algorithm to build clusters in the usual way in the interior, once the links to the external boundary are marked as non-candidates for activation. Once the cluster is built, there are two possibilities: if the cluster includes no sites that are connected to the boundary sites, then one can go ahead and flip it as usual; if the cluster does include one or more sites connected to the external boundary, then flipping it will change a part of the action which is being ignored by the building algorithm, namely the terms relating to links connecting to the boundary sites. In this last case the situation is similar to the situation discussed in the previous section for the case of external sources, namely there is a part of the action that has to be taken into account separately.

In figure 5 one can see an example of a cluster built on the lattice, including a few sites that are connected to the boundary. The same type of solution used in the previous section applies in this case as well, but in this case the relevant separation of the action is two parts is

$$S[\psi] = S_i[\psi] + S_b[\psi] = -\beta \sum_{\ell_i} \psi_- \psi_+ - \beta \sum_{\ell_b} \psi_- \psi_+$$

where ℓ_i are the links strictly in the interior of the lattice and ℓ_b are those connecting the interior to the boundary. The interior part S_i of the action is taken into consideration by the cluster-building algorithm, and the boundary part S_b can be taken into account by a Metropolis test done afterwards, giving the probability that the cluster will be flipped. Here is the algorithm for this case:

1. Choose a single site of the *interior* the of lattice for starting to build the cluster, in a random way.

2. Consider all the links connected to that initial site, which are *not* connected to the external boundary; the ℓ_- links are never to be activated; activate the ℓ_+ links with the probability $p_+ = 1 - \exp(-2\beta\psi_-\psi_+)$, thus forming a first cluster of sites, that is, updating the cluster from a single site to possibly a few sites.

3. Given the set of sites added to the cluster in the *previous* update of the cluster (which is *not* the whole cluster), consider all the links that connect those sites to sites that are still outside the cluster, and which are *not* in the external boundary; among these, activate the ℓ_+ links with the probability p_+, thus enlarging (updating) the cluster.

4. Loop back to step 3 until the set of links left for activation trial in the next round of the loop is empty; this means that you stop this loop at the first instance of step 3 that adds no new sites to the cluster.

5. Calculate the variation ΔS_b of the boundary term of the action caused by flipping the cluster just built; this will of course be zero if there are no sites in the cluster connected to sites in the boundary.

6. If it happens that $\Delta S_b \leq 0$, then flip the cluster with probability equal to 1.

7. If $\Delta S_b > 0$, then flip the cluster with probability $\exp(-\Delta S_b)$; if the update is rejected, repeat the current configuration in the stochastic sequence.

8. Loop back to step 1, in order to build and possibly flip a new cluster.

One can see that, just as in the case of external sources, in this case also the algorithm is inevitably somewhat less efficient than the pure Wolff algorithm, in terms of diffusion speed through configuration space. This will happen mostly in the rather unlikely case when one has very large values at the boundary, and also for very small lattices, since the ratio of the number of boundary sites to the number of internal sites is much larger on small lattices, specially in the larger dimensions. Just as in the case of external sources, in this case also we have no control over the rejection rate, although this tends to be a lesser problem here than in the case of the presence of a bulk term in the action, as is the case of the external-sources term. Of course, one can easily combine fixed boundary conditions with the presence of external sources, by simply considering for the Metropolis test all parts of the action which are not taken into consideration by the cluster-building algorithm.

4 Superclusters and Truncated Clusters

One possible problem with the cluster algorithm is the possibility that under some circumstances it will build many very large clusters, taking most of the lattice. In a situation in which there is reflection invariance of the whole action, the flipping of such a "supercluster", combined with a symmetry transformation reflecting the whole lattice, is actually equivalent to flipping the much smaller complementary cluster. In this case one may be using a lot of computational effort building large clusters in order to actually make relatively small changes in the configurations, and hence the algorithm may become inefficient.

This kind of problem appears mostly in low dimensions and for very large values of β (low "temperatures"), away from the critical region, situations which are therefore only of secondary interest for those interested mostly in quantum field theory. However, it might be induced also by the presence of strong, constant external sources, which tend to strongly orient the fields and hence to favor the building of large clusters. In this case not only large clusters are more likely to be built, but they will also most likely be rejected, because flipping them will tend to cause large increases of the S_η term of the action. Once again large amount of computational effort will be spent building clusters that are likely to be rejected, and the algorithm may then diffuse through configuration space only very slowly.

One way to handle this kind of problem is to use truncated clusters. A truncated cluster is one that we just stop building at some point, according to some criterion. For example, one may require that the clusters have no more than a certain given maximum number of sites in them. The way to handle such clusters is suggested by the treatment of fixed boundary conditions described in the previous section. At the point where one arbitrarily stops building the cluster, one has a list of ℓ_+ links that should but no longer will be tested for activation. The corresponding terms of the action have not been taken into account by the building algorithm, so that flipping this truncated cluster would violate the condition of detailed balance.

The way to correct this is quite clear now: one examines the remaining ℓ_+ links and calculates the variation of the corresponding terms of the action due to the flipping of the cluster. Having done this, one performs a Metropolis test with this variation of the action, thus establishing a probability for flipping the cluster. This is exactly the same situation that one has for fixed boundary conditions, when the cluster has part of its sites connected to the boundary sites. The only difference is that in our case here the situation is realized at parts of the border of the truncated cluster, consisting of the links that have not been tested for activation by the building algorithm.

The introduction of a new parameter, such as the maximum number of sites in a cluster, gives us a handle to try to control the rejection rate and thus to try to optimize the efficiency of the algorithm, that is, the speed with which it diffuses through configuration space. Note that one could use also other criteria for truncating the cluster, such as that the cluster should not extend beyond a certain radius from the initial site. This technique may be useful in some rather extreme situations. For example, one might consider using it when, with the use of the usual algorithm, the average size of the clusters becomes larger than one half the total number of sites in the lattice.

Note that if one reduces the maximum number of sites in a cluster to just 1, then the cluster algorithm is reduced back to the simple Metropolis algorithm, with a strictly local update. Therefore, we can see that reducing too much the cluster size may bring back the usual critical slowing down problems of the Metropolis algorithm. Note also that in this case there is no fixed separation of the action in two parts, the separation is dynamical, determined only at the point where we decide to stop building the current cluster. This variation of the algorithm is not actually difficult to implement, but finding the optimal maximum cluster size under a given set of circumstances may not be so easy a task.

5 Optimization of the Algorithm

Under some common circumstances it is possible to optimize the extended Wolff algorithms by technical means. This will depend fundamentally on the characteristics of the external source or of the boundary conditions which are present. Specially in the case of external sources, the main cause of performance degradation is the high cluster rejection rate associated to large values of the sources. Let us recall that, after the cluster is built, one has to calculate the variation ΔS_η of the external-source term of the action and then accept the cluster with the probability $\exp(-\Delta S_\eta)$. In order to do this, one extracts a random number $r \in [0,1)$ and only accepts the cluster if

$$r \leq \exp(-\Delta S_\eta),$$

or, equivalently, if

$$\Delta S_\eta \leq -\ln(r),$$

where one should note that $-\ln(r)$ is always a positive number. The best way to calculate ΔS_η is to do so along the construction of the cluster, incrementing ΔS_η as the sites are added to it. The optimizations we are talking about here are usually the following two things: if the variation of the action S_η is guaranteed to be negative, then the cluster is sure to be accepted and it is not necessary to calculate ΔS_η or to perform the final Metropolis test, which saves some computational effort; if the variation of the action is known to always increase as sites are added to the cluster, then at some point one may know that the cluster will be rejected even before completing it, and then one may drop the cluster, thus saving the significant computer effort required to finish building it.

The main idea behind this is that, so long as the random number r used for the final Metropolis test has a flat distribution of probabilities in $[0,1)$, it does not matter whether it is extracted from the generator after the cluster is built or before one starts to build it. Therefore one may extract it beforehand and keep the value $-\ln(r)$ in storage for tests to be performed along the building of the cluster. In order to better exemplify the idea, let us consider a specific case. Let us consider the case in which the external source has a constant and non-vanishing value η over the whole lattice. As we saw in the discussion of the building algorithm for the clusters, it is always true that all the sites of a cluster have the field oriented in the same direction. Therefore, if η is a constant, all the variations of the action S_η associated to the flipping of the field at each site of the cluster will have the same sign. Therefore, given η and the initial site ψ_i of the cluster, the following statements are true:

- If $\eta\psi_i < 0$ then all the variations of S_η due to the inclusion of sites in the cluster will be negative, and hence the cluster is sure to be accepted. In fact, ΔS_η will decrease monotonically from zero as the cluster is built.
- If $\eta\psi_i > 0$ then all the variations of S_η due to the inclusion of sites in the cluster will be positive, and hence the cluster may be rejected. In this case, ΔS_η will increase monotonically from zero as the cluster is built.

Note that, so long as the external source η is not zero, it never happens that $\eta\psi_i = 0$. Note also that, if the random number r extracted beforehand happens to be 0, then one cannot calculate and store $-\ln(r)$, but in this case this is also not necessary, because the condition

$$r = 0 \leq \exp(-\Delta S_\eta)$$

will be satisfied regardless of the value of ΔS_η, and hence the cluster is sure to be accepted. A simple and well-structured way to implement these ideas is to imagine that the code for building the cluster can run in one of two modes: a "check" mode, in which ΔS_η is calculated along the building of the cluster, and a "no-check" mode, in which the variations of S_η are simply ignored and the cluster is built and flipped just as would happen in the absence of any external sources. Here is the algorithm for the optimized code in this case:

1. Choose the first site randomly, starting in check mode.
2. If $\eta\psi_i < 0$ then go to no-check mode, build and flip the cluster, and loop back to step 1, in order to build and possibly flip a new cluster.
3. If $\eta\psi_i > 0$ then extract a random number r for the Metropolis test.
4. If $r = 0$ then go to no-check mode, build and flip the cluster, and loop back to step 1, in order to build and possibly flip a new cluster.
5. Calculate and store $-\ln(r)$, and stay in check mode, building the cluster and calculating ΔS_η along the construction.

6. As each site is added to the cluster, check the relation between the value of ΔS_η and $-\ln(r)$: if the monotonically increasing ΔS_η surpasses $-\ln(r)$, then reject the cluster, which means that you quit building it, repeat the current configuration in the stochastic sequence, and loop back to step 1, in order to build and possibly flip a new cluster.

7. If you get to the end of the construction of the cluster without a rejection, then flip it with probability equal to 1 and loop back to step 1, in order to build and possibly flip a new cluster.

In this way one makes as much economy as possible of computer effort. The idea can be used in essentially the same way for other forms of constant external sources such as, for example, one existing only over the world line of a particle, which is a one-dimensional line of sites within the d-dimensional lattice. Also, it can easily be adapted to fixed boundary conditions with a constant value of the field at the external boundary. In fact, the external source or the value at the boundary do not really have to be constant, so long as they do not change sign, for this idea to work as explained above. In these other cases additional indexing structures may be necessary, in order to single out the sites where the external source exists, or the interior sites which are next to the external boundary, connected to it by the boundary links.

It is much more difficult to optimize the case of a general external source or set of boundary conditions, which can change sign freely. One can use ideas which are similar to the ones described above, but it is much more difficult to obtain a significant amount of optimization. The basic idea goes something like what follows. Let us consider a typical intermediate situation during the construction of the cluster. Assume that one keeps updated along the construction two positive variables, $\Delta S_{\eta,\min}$ and $\Delta S_{\eta,\max}$, which hold the maximum value that can be *subtracted* from the current value of ΔS_η and the maximum value that can be *added* to the current value of ΔS_η, respectively, during the rest of the construction of the cluster.

Under these conditions, after the inclusion of a certain site in the cluster, if one verifies that $\Delta S_\eta + \Delta S_{\eta,\max}$ is less than or equal to $-\ln(r)$, then there is no chance that the action will ever increase enough to cause a rejection, and therefore one can go to no-check mode in order to build the rest of the cluster and flip it. This is so because the sum above is larger than or equal to the maximum possible value that ΔS_η can assume from this point on. Similarly, if one verifies that $\Delta S_\eta - \Delta S_{\eta,\min}$ is greater than $-\ln(r)$, then there is no chance that the action will ever decrease enough to avoid a rejection, and therefore one can quit the current cluster before actually completing its construction. In this case, the difference above is smaller than the minimum possible value that ΔS_η can assume from this point on.

The initial values of the variables $\Delta S_{\eta,\min}$ and $\Delta S_{\eta,\max}$ can be obtained, respectively, as the sum of all negative single-site variations and the sum of all positive single-site variations of the action S_η. Given the orientation of a certain cluster, these sums can be limited to the sites of the lattice where the field has that orientation, or even only to the sites that can actually participate of that particular cluster, because they are connected to the initial site by strings of ℓ_+ links. Along the construction the values of $\Delta S_{\eta,\mathrm{min}}$ and $\Delta S_{\eta,\mathrm{max}}$ have to be modified by the addition or subtraction of the variation of S_η at each site which is added to the cluster, in order to keep the variables up to date. There is, of course, an additional overhead of computer effort, needed in order to initialize and update these variables. Due to this and to the added complexity of the resulting code, sometimes it may not be worth while to implement this kind of optimization.

6 Fortran Routines

There are Fortran routines available implementing the cluster-update routines as described in this document. There are also a few test programs that may be useful as examples of how to use the routines. As they currently stand, these routines are written for use in either 32-bit or 64-bit processors. The files described in what follows, containing the source code, are freely available in a compressed tar file at the URL:

`http://latt.if.usp.br/technical-pages/twawesab/`

There are files with the definition of the necessary data structures, as well as the initialization routines and the cluster-update routines themselves. These modules are meant to be integrated into larger programs at a source-code level. The files containing the data structures are include files, meant to be included in the modules that need access to the data structures they contain. All interchange of data among modules is made by means of common blocks. Each initialization routine defines the data in the common block in the corresponding include file. These initialization routines should be called once at the beginning of the program, one for each common block that is used in the program.

Read the `Makefile`, the `README` file and the source code in the `src/` subdirectory in order to see which modules depend on which data structures. All the code is fairly well documented with internal comments. Here are short explanations of the nature of each source-code file. First the files with the data structures:

`fp_def.f`: an include file with the basic parameters defining the run; this is needed by all modules.

`fp_cal.f`: an include file with other parameters, which are calculated using the basic ones; this is needed by most modules.

`cb_param.f`: an include file with a common block containing additional initial and calculated parameters.

`cb_flags.f`: an include file with a common block containing the dimensionality flags.

cb_index.f: an include file with a common block containing the data structures related to the indexing of neighbors.

cb_exsrc.f: an include file with a common block containing the data structures related to the external sources.

cb_field.f: an include file with a common block containing the data structures related to the field.

cb_fdbck.f: an include file with a common block containing variables related to feedback and monitoring functions.

The files containing the main routines, including the initialization routines and the cluster-update routines themselves:

init_param.f: a subroutine that initializes the common block in cb_param.f.

init_flags.f: a subroutine that initializes the common block in cb_flags.f.

init_index.f: a subroutine that initializes the common block in cb_index.f.

init_exsrc.f: a subroutine that initializes the common block in cb_exsrc.f.

init_field.f: a subroutine that initializes the common block in cb_field.f.

cluster_no_source.f: a cluster routine for a periodical lattice without any external sources.

cluster_general_bas.f: a cluster routine for a periodical lattice with an arbitrary external source over the whole lattice, without any specific optimizations.

cluster_general_opt.f: an optimized cluster routine for a periodical lattice with an arbitrary external source over the whole lattice.

cluster_constant.f: an optimized cluster routine for a periodical lattice with a constant external source over the whole lattice.

cluster_localized.f: an optimized cluster routine for a periodical lattice with a constant external source over a localized subset of the lattice, such as a single site or a straight line of sites.

Files containing the test programs and some auxiliary routines, including a couple of C interfaces to system facilities:

test_cluster_no_source.f: a program that executes in an infinite loop the cluster routine cluster_no_source, and measures the average cluster size and the speed of the routine.

`test_cluster_general_bas.f`: a program that executes in an infinite loop the cluster routine `cluster_general_bas`, and measures the average cluster size, the cluster rejection rate and the speed of the routine.

`test_cluster_general_opt.f`: a program that executes in an infinite loop the cluster routine `cluster_general_opt`, and measures the average cluster size, the cluster rejection rate and the speed of the routine.

`test_cluster_constant.f`: a program that executes in an infinite loop the cluster routine `cluster_constant`, and measures the average cluster size, the cluster rejection rate and the speed of the routine.

`test_cluster_localized.f`: a program that executes in an infinite loop the cluster routine `cluster_localized`, and measures the average cluster size, the cluster rejection rate and the speed of the routine.

`urandom_ctof.c`: a Fortran-callable C interface to the system libraries, to get a random seed from the Linux kernel.

`clock_ctof.c`: a Fortran-callable C interface to the system libraries, to get the CPU time of the current process from the system.

`dranr-1.f`: the first random number generator from the authors of the book "Numerical Recipes".

`dranr-2.f`: the second random number generator from the authors of the book "Numerical Recipes".

References

[1] Wolff, Ulli, "Collective Monte Carlo Updating for Spin Systems", *Phys. Rev. Lett.*, v.62, n.º 4, 1989.

[2] Wang, Jian-Sheng, and Swendsen, Robert H., "Cluster Monte Carlo Algorithms", *Physica* A, 167, 565-567, 1990.