

Thomas Bisig<sup>1</sup>

<sup>1</sup>*Institute for Theoretical Physics, ETH Zürich, CH-8093, Switzerland*

(Dated: October 30, 2005)

We study the properties of triangular lattices based on Josephson junctions due to their possible implementation as topologically protected quantum bits. We first study the problem analytically before we solve it numerically. The aim is to find configurations of the setup, where energy decreases with distance.

## I. INTRODUCTION

Topologically protected quantum bits could be the first implementation of quantum bits which overcomes two conflicting requirements: the qubits should be manipulable through external signals, while not be destroyed by external influence. While quantum optics proposals have only optimal isolation, solid-state implementations exhaust the nanoscale techniques at the time. According to a recently published letter to nature<sup>1</sup> and other papers<sup>2,3</sup>, triangular lattices constructed using Josephson junction arrays could be such an implementation of topologically stable qubits. It has been shown that a strongly correlated system developing an isolated twofold degenerate quantum dimer liquid ground state can be used in the construction of topologically stable qubits. The discussed implementation uses Josephson junction arrays. We want to analyze the energy and its dependency on differently chosen capacities of such a liquid dimer model and therefore are interested in calculating the capacity matrix of a specific setup. Our main interest is to find the configuration of Josephson junctions for which energy decays with distance of two dimers, as this is a requirement<sup>1</sup> on the discussed model. We will find different configurations of capacities for different directions of the energy's decay. For visualization purposes, we developed a method to plot the energy landscape for a given configuration.

The paper is structured as follows: In Sec. II we calculate the energy of the lattice for given charges on the islands and the problem's capacity matrix analytically. To find the energy, we need the inverse of the capacity matrix, calculated in Sec. III. The next step in Sec. IV is to show that if we join six islands together, we will find the capacity matrix of a triangular lattice and we study the limit of vanishing ground capacity. In Sec. V we take help of mathematical tools and calculate the energy numerically. In Sec. VI we will find the capacity configuration for energy decay and an example for random and columnar dimer setup energies can be found in Sec. VII as well as plots of the energy landscape. Conclusions are summarized in Sec. VIII. Most of the code (or at least one version) used to find these results can be found in Appendix A, B, C and D.

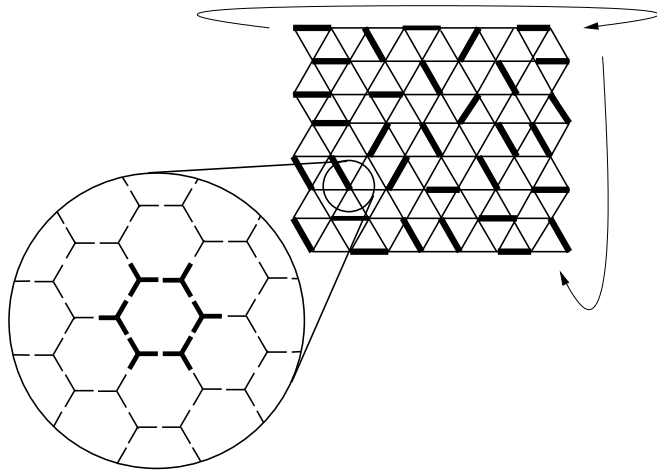


FIG. 1: A triangular lattice with random dimer configuration built up by hexagons (see enlarged region). The thick lines in the lattice mean a dimer, while normal lines just visualize the connection between the different hexagons. A dimer means that there is half a Cooper pair on the two respective neighbor Y-islands, which can be seen in the enlarged region. Six of this Y-islands form a hexagon which is represented as a crosspoint in the lattice. There is only one dimer per hexagon allowed (one half Cooper pair per hexagon). The arrows mean the boundary conditions in both x- and y-direction (toroidal setup).

## II. CALCULATING THE ENERGY

We look at a triangular lattice (Fig. 1) built up by hexagons (Fig. 2) each based on Josephson junctions. Every point in the lattice is an island divided into six sub islands. The sub islands are connected over Josephson junctions to form a hexagon just like every of the six arms of the hexagon is linked with an arm to one of its neighbor hexagons. We take boundary conditions along both x- and y-axis (toroidal setup) and define the following constants (see Fig. 2):  $Q_i$  is the charge on the  $i^{\text{th}}$  Y-island,  $V_i$  is the potential the  $i^{\text{th}}$  Y-island is on,  $V_{i+\mu_k}$  is the potential one of the  $i^{\text{th}}$  islands neighbor is on,  $C_i^k$  is the capacitance between the  $i^{\text{th}}$  Y-island and one of its three respective neighbors (clockwise) and  $C_i^E$  is the ground capacity of the  $i^{\text{th}}$  Y-island. We sum up all parts of an Y-island which contribute to the charge on one island and obtain

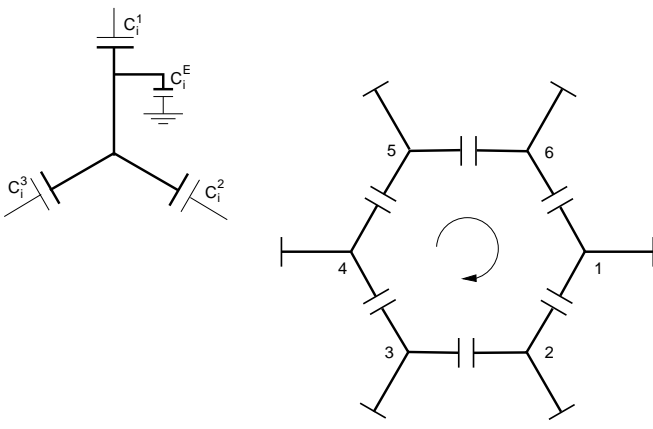


FIG. 2: Every (cross)point in a lattice configuration means a hexagon built up by Y-island (on the left). The convention for the calculation of the capacity matrix can be seen on the upper left. For the numerical calculation the Y-islands are linked over a capacitance  $C_h$  when they belong to the same hexagon and  $C_l$  when they link to one of the Y-islands on the neighbor hexagon. The enumeration convention for the scripts (see Appendix A) can be seen on the right. All the Y-islands whose arm points to the right are 1-islands. All other Y-islands are identified analog.

$$Q_i = C_i^E V_i + \sum_k C_i^k (V_i - V_{i+\mu_k}) \quad (1)$$

and its capacity matrix

$$C_{ij} = \left( C_i^E + \sum_k C_i^k \right) \delta_{i,j} - \sum_k C_i^k \delta_{i,j+\mu_k}. \quad (2)$$

Therefore we can write the Energy  $E = \frac{1}{2}CV^2$  as

$$E = \frac{1}{2} \sum_i \left[ (C_i^E + \sum_k C_i^k) V_i^2 - \sum_k V_i V_{i+\mu_k} \right] \quad (3)$$

and by definition of the capacity matrix [see Eq. (2)]

$$E = \frac{1}{2} \sum_{ij} V_i C_{ij} V_j = \frac{1}{2} \langle V | C | V \rangle. \quad (4)$$

Because the capacity matrix is regular for nonzero ground capacity there exists the inverse and moreover  $C = C^T$ . We now want to calculate the energy of the lattice in dependence of the charges on the islands and not the potential they are on. So, with  $V = C^{-1}Q$  we can rewrite the energy as

$$E = \frac{1}{2} \langle V | C | V \rangle \quad (5)$$

$$= \frac{1}{2} \langle C^{-1}Q | C | C^{-1}Q \rangle \quad (6)$$

$$= \frac{1}{2} \langle Q | C^{-1} | Q \rangle \quad (7)$$

where we used the equality of the capacity matrix and its transposed.

### III. CALCULATING THE CAPACITY MATRIX

In order to find the inverse of the capacity matrix we make two different approaches: In this chapter we calculate the inverse of the capacity matrix analytically, in Sec. V we calculate it numerically. We simplify this analytical task by transforming into Fourier space. Let  $p$  and  $q$  be vectors of the dual lattice. Then we find ( $\hat{X}$  means the Fourier transformed of  $X$ )

$$\hat{V}_q = \sum_j e^{iqj} V_j \quad (8)$$

and by using this to calculate the capacity matrix in the Fourier space with Eq. (2) we get

$$C_{pq} = \sum_{lm} e^{iqm} C_{lm} e^{-ipl} \quad (9)$$

$$= \sum_{lm} e^{iqm} \left( (C_m^E + \sum_k C_m^k) \delta_{ml} - \sum_k C_m^k \delta_{m,l+\mu_k} \right) e^{-ipl} \quad (10)$$

$$= \sum_m e^{il(q-p)} (C_m^E \sum_k C_m^k) - \sum_{lm} e^{iqm} \sum_k C_m^k e^{-ip(m-\mu_k)}. \quad (11)$$

The second term simplifies to

$$\sum_{mk} C_m^k e^{ip\mu_k} e^{im(q-p)}. \quad (12)$$

Using that the capacities are equal for all islands

$$C_m^E = C^E \quad (13)$$

and

$$\sum_k C_m^k = \sum_k C^k \quad (14)$$

we find that Eq. (12) becomes

$$\sum_m e^{im(q-p)} \sum_k C^k e^{ip\mu_k}. \quad (15)$$

The calculated sum

$$\sum_m e^{im(q-p)} = \delta_{pq} \quad (16)$$

simplifies Eq. (15) this even more to

$$\sum_k C^k e^{ip\mu_k} \delta_{pq}. \quad (17)$$

The first term in Eq. (11) becomes with Eq. (13)

$$(C^E + \sum_k C^k) \delta_{pq}. \quad (18)$$

Hence, the capacity matrix is now diagonal

$$C_{pq} = \delta_{pq} \left[ \left( C^E + \sum_k C^k \right) - \sum_k C^k e^{ip\mu_k} \right] \quad (19)$$

and can therefore be inverted. We transform this back and the  $(ml)^{\text{th}}$  element of the inverse matrix becomes

$$C_{ml}^{-1} = \sum_p e^{ip(l-m)} \frac{1}{C^E + \sum_k (C^k - C^k e^{ip\mu_k})} \quad (20)$$

#### IV. LIMIT OF JOINING SIX Y-ISLANDS TO ONE POINT AND THE LIMIT FOR $C^E \ll C^z$

The inverse capacity matrix for a triangular lattice is given by

$$C_{ml}^{-1} = \sum_p e^{ip(l-m)} \frac{1}{C^E + 6C^z - \sum_{s=1}^3 2C^z \cos p_s} \quad (21)$$

where  $C^z$  denotes the link capacity which is the same for all six links. In the limit of no capacities on a hexagon the inverse capacity matrix of the general case (Eq. (20)) has to be equal to the inverse capacity matrix of the triangular lattice (Eq. (21)) in a special sense: The capacity matrix in the general case has dimension  $D_g = 6 \cdot n \cdot m$  where  $n$  and  $m$  are the width and length of the lattice, whereas the dimension in the case with joined islands is just  $D_s = n \cdot m$ . So, to be exact, the energies have to be the same in both cases and not the capacity matrices. We also have to take into account that in the general case each of the six islands are on a potential whereas in the case of the triangular lattice, the whole joined island is on a single potential. So we have to 'take away' five ground potentials when we make the limit. For this task take the inverse of Eq. (20) and let all the  $C^k$  which are not the linking capacities between the different hexagons (Fig. 2) be zero and define the linking capacities  $C^z = C^k$ . So the remaining part of the original capacity matrix is now

$$C_{ml} = \sum_p e^{ip(l-m)} (C^E + C^z - C^z e^{ip\mu_k}). \quad (22)$$

The index  $k$  remains in the exponent because we have to take always the correct neighbor of the Y-island we look at. This will soon be more clear. Now a hexagon is built up by six Y-islands. The whole energy of one hexagon is the sum of the energies of every single Y-island. This means that if we put an electron on a hexagon (with no link capacities) we get the same energy as if we put an electron on each of the Y-islands and sum up. As stated earlier we have to be careful with the ground capacities: we have to count them only once. Following the steps above we obtain for the whole capacity  $C_{ml}$

$$C_{ml} = \sum_u C_{ml}^u = \sum_p e^{ip(l-m)} (C^E + 6C^z - C^z \sum_k e^{ip\mu_k}), \quad (23)$$

where we sum over all the different  $k$ , which means that we have to differ between the different neighbors. Let  $\mu_i = \pm e_i$  where  $e_i$  is the  $i^{\text{th}}$  direction (Fig. 2) but  $\mu_4 = -\mu_1$ ,  $\mu_5 = -\mu_2$  and  $\mu_6 = -\mu_3$ . So we can write the last term in Eq. (23) as

$$\sum_k e^{ip\mu_k} = 2 \sum_{s=1}^3 \cos(pe_s) = 2 \sum_{s=1}^3 \cos(p_s) \quad (24)$$

So we obtain for the inverse of the capacity  $C_n$

$$C_{ml}^{-1} = \sum_p e^{ip(l-m)} \frac{1}{C^E + 6C^z - \sum_{s=1}^3 2C^z \cos p_s}. \quad (25)$$

This is the same as Eq. (21) what we wanted to show.

Let's take a look at the limit  $C^E \ll C^z$ . We are interested in how the energy depends on the distance of two charges placed on the lattice. We look only at the proportionality so in the end the result is correct up to a proportional constant. If we take the limit of infinitely many islands and infinite width and length of the lattice, the sum in Eq. (20) becomes an integral

$$C_{ml}^{-1} \sim \int_{-\pi}^{\pi} d^2 p e^{ip(l-m)} \frac{1}{C^E + 6C^z - \sum_{s=1}^3 2C^z \cos p_s}. \quad (26)$$

For  $C^E \ll C^z$  the main values of the integral are around  $p_i \approx 0$ . We expand the cosine around zero

$$\cos p_i \approx 1 - \frac{(p_i)^2}{2} \quad (27)$$

and obtain

$$C_{ml}^{-1} \sim \int_{-\infty}^{\infty} d^2 p e^{ip(l-m)} \frac{1}{C^E + C^z (p_1^2 + p_2^2 + p_3^2)}. \quad (28)$$

Using that the directions  $e_i$  are normalized and transforming the integral into polar coordinates one obtains

$$C_{ml}^{-1} \sim \frac{1}{C^z} \int_0^\infty dp \frac{p}{\frac{C^E}{C^z} + p^2} \cdot \int_0^{2\pi} d\phi e^{ip|l-m|\cos\phi}. \quad (29)$$

Now we use the following equality

$$\int_0^\infty \frac{J_\eta(bx)x^{\eta-1}}{(x^2+a^2)^{\mu-1}} dx = \frac{a^{\eta-\mu}b^\mu}{2^\mu\Gamma(\mu+1)} K_{\eta-\mu}(ab) \quad (30)$$

where  $K_0(z)$  has the property, that its limit as  $z \rightarrow 0$  is  $K_0(z) = -\ln z$  and  $J_\eta(x)$  is defined as

$$J_\eta(x) = \frac{1}{2\pi} \int_0^{2\pi} e^{i(x\sin\phi - \eta\phi)} d\phi. \quad (31)$$

Using this we obtain for the integral Eq. (29)

$$C_{ml}^{-1} \sim \frac{1}{C^z} \int_0^\infty dp \frac{pJ_0(p|m-l|)}{\frac{C^E}{C^z} + p^2} \quad (32)$$

$$= \frac{1}{C^z} K_0\left(\sqrt{\frac{C^E}{C^z}} |m-l|\right) \quad (33)$$

$$\rightarrow -\frac{\ln|m-l|}{C^z}, \quad (34)$$

where we used the limit as  $C^E \rightarrow 0$ .

## V. NUMERICAL CALCULATION OF THE CAPACITY MATRIX

After knowing what the capacity matrix is analytically, we take some mathematical tools next. For this task a Perl<sup>4</sup> script has been written which calculates the capacity matrix for a given  $n \times m$  Lattice. This script finds all neighbors of a specific Y-island and fills the capacity matrix (with predefined inputs for the capacities). The problem is to find all boundary conditions i.e. to match every single island with its correct neighbors. Inputs are the width and length of the lattice (number of rows has to be even because of the boundary conditions in both directions), the ground capacity, the capacity between islands in the same hexagon and between the ones outside. To calculate the energies and invert capacity matrices a C program has been written using GSL<sup>5</sup>. This program calculates the energy for a given setup (columnar, staggered, liquid or just any predefined setup) by finding first the inverse of the capacity matrix and then calculate Eq. (7). Finally we put twice half a Cooper pair ( $e = 1.602 \cdot 10^{-19}C$ ) onto two neighbors (to form a dimer) and calculated the energy step-by-step when a second dimer is put somewhere onto the lattice. Using Mathematica<sup>6</sup>, this finally gives us the energy landscape for a given capacity matrix (see Sec. VII). The boundary conditions can be seen quite easy this way.

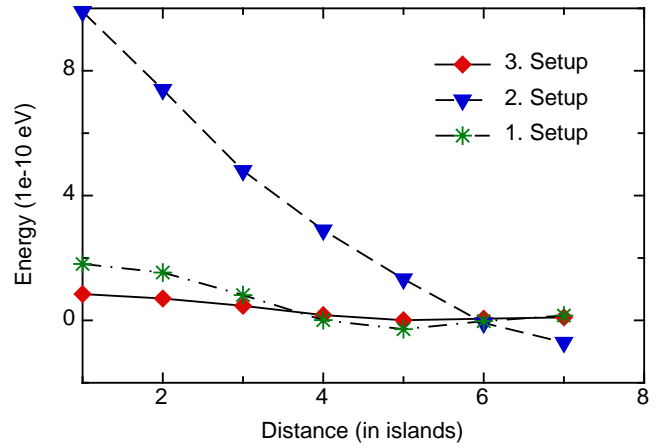


FIG. 3: This plot shows the three fastest decaying energies in horizontal direction in dependence of the distance (in islands). The capacity sets for this setups can be seen in Table I.

## VI. ENERGY DECREASE WITH DISTANCE

In order to find the set of capacities which leads to energy decrease with growing distance we continued as follows. We are interested in lattices built up by  $16 \times 16 \times 6$  Y-islands ( $16 \times 16$  hexagons). We set a dimer on the lattice and calculated the energies for both paths given in (Fig. 5) by dividing the infinitely many possible capacity sets into a discrete list from  $1F$  to  $0.5 \cdot 10^{-9}F$ . Looking at the plots for the first direction (Fig. 5) lead us to our first conclusion: The energy decreases only if we take the hexagon- and ground capacities almost equal and the link capacity more than ten times smaller. As there were more than one possible configuration we've chosen to set  $C_h = 10^{-7}F$  and to look at a range from  $0.7 \cdot 10^{-8}F$  to  $0.7 \cdot 10^{-9}F$  for the link capacity. As we found out, the ground- and hexagon capacity has to be taken almost equal, so we used a range for the ground capacity from  $1.03 \cdot C_h$  to  $0.97 \cdot C_h$ . We define a variable  $W_{optimize} = (E_1 - E_{other})/|E_1|$  where  $E_{other}$  is the average of all energies - except the first ( $E_1$ ). We take the absolute value of  $E_1$  because we want to consider only the cases where  $E_1$  is larger than the average. This has the interpretation of a relative energy between the nearest dimer configuration ( $E_1$ ) and the average of all the other ( $E_{other}$ ). This gives us a tool to determine the fastest decreasing energy: We only look at cases where  $W_{optimize}$  is positive, so we define the configuration with the largest value  $W_{optimize}$  and decreasing energies until the fifth island to be the configuration with the fastest decreasing energy. It is possible to take other optimizations as the one we used. The three highest optimization values we obtained for direction one and the respective capacity sets can be seen in Table I and the plot for the optimized capacity set in Fig. 3.

According to this results we obtain for the second direction (Fig. 5) the capacity set in Table II and the energy

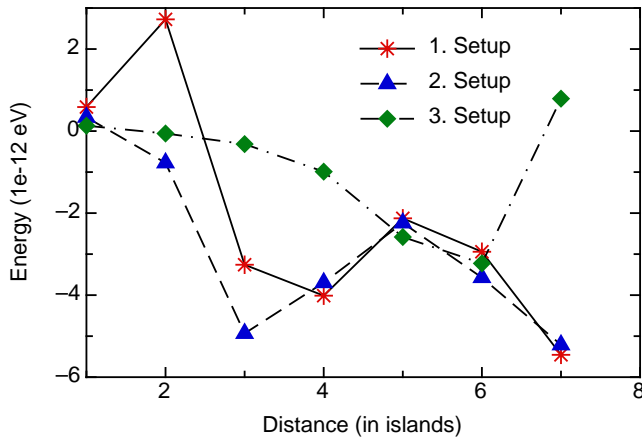


FIG. 4: This plot shows the three fastest decaying energies in diagonal direction in dependence of the distance (in islands). The capacity sets for this setups can be seen in Table II.

TABLE I: The three highest optimization values for the first direction where  $C_h = 10^{-7}F$ , the link capacity  $C_l$  is in a range from  $0.7 \cdot 10^{-8}F$  to  $0.7 \cdot 10^{-9}F$  and the ground capacity between  $1.03 \cdot 10^{-7}F$  and  $0.97 \cdot 10^{-7}F$ . Optimization value is  $W_{optimize} = (E_1 - E_{other})/|E_1|$  where  $E_{other}$  is the average of all energies - except the first ( $E_1$ ). The values of the link and ground capacity are given as percentage of the hexagon capacity.

Nr.	$\frac{C_l}{C_h}$	$\frac{C_g}{C_h}$	$W_{optimize}$
1	$5 \cdot 10^{-2}$	1	0.797265
2	$3 \cdot 10^{-2}$	0.98	0.736900
3	$6 \cdot 10^{-2}$	1	0.705196

to distance plot as in Fig. 4. Here we used a different criteria for the fastest decay. We took the set of capacities which has the highest optimization value and at most one energy  $E_n$  ( $n > 1$ ) that is higher than  $E_1$ .

TABLE II: The three highest optimization values for the second direction where  $C_h = 10^{-7}F$ , the link capacity  $C_l$  is in a range from  $0.7 \cdot 10^{-8}F$  to  $0.7 \cdot 10^{-9}F$  and the ground capacity between  $1.03 \cdot 10^{-7}F$  and  $0.97 \cdot 10^{-7}F$ . Optimization value is  $W_{optimize} = (E_1 - E_{other})/|E_1|$  where  $E_{other}$  is the average of all energies - except the first ( $E_1$ ). The values of the link and ground capacity are given as percentage of the hexagon capacity.

Nr.	$\frac{C_l}{C_h}$	$\frac{C_g}{C_h}$	$W_{optimize}$
1	$5 \cdot 10^{-2}$	0.99	11.2004
2	$4 \cdot 10^{-2}$	0.98	11.1332
3	$2 \cdot 10^{-2}$	1	5.28732

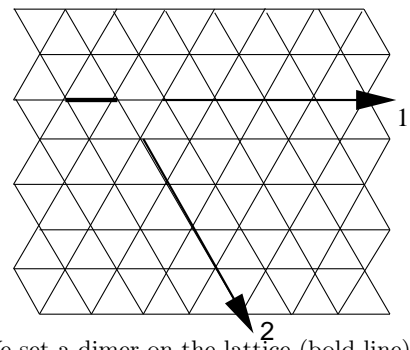


FIG. 5: We set a dimer on the lattice (bold line) and look at the energy curve for direction one (horizontal). This means that we calculate the energy between the bold dimer and each dimer in direction one separately. The same is done for direction two (diagonal).

TABLE III: This table summarizes the calculated energies averaged over  $n$  random configurations ( $n = 4, 8, 16, 32$ ) for a lattice build up by  $16 \times 16$  islands. Additionally the variance  $\sigma$  and the standard deviation  $\eta$  can be found in the table as well.

$n$	$E_{16 \times 16}^{\text{rand}}$ [eV]	$\sigma$	$\eta$
4	$1.2693 \cdot 10^{-9}$	$2.5861 \cdot 10^{-18}$	$1.6081 \cdot 10^{-9}$
8	$1.7216 \cdot 10^{-9}$	$1.0956 \cdot 10^{-17}$	$3.3100 \cdot 10^{-9}$
16	$1.5146 \cdot 10^{-9}$	$6.6207 \cdot 10^{-17}$	$2.5731 \cdot 10^{-9}$
32	$2.5835 \cdot 10^{-9}$	$1.0111 \cdot 10^{-17}$	$3.1797 \cdot 10^{-9}$

## VII. EXAMPLES AND ENERGY LANDSCAPES

As a comparison to the energies and setups above, we would like to calculate the energy of a  $16 \times 16$  lattice with a random and a columnar dimer configuration. The capacities are chosen as set Nr. 1 in Table I.

We first calculate the energies for the columnar setup. We put half a Cooper pair ( $e = 1.602 \cdot 10^{-19}C$ ) on every allowed Y-island and calculate the energy of the lattice and obtain

$$E_{16 \times 16}^{\text{col}} = 1.37846 \cdot 10^{-9} eV. \quad (35)$$

With the same definition of the capacities we calculate the random dimer setup. We average over 4, 8, 16 and 32 randomly chosen configurations and obtain the energies, variances and standard deviations seen in Table III.

To see the boundary conditions of the lattice and the energy decay for direction one, we plot the energy landscape in Fig. 6. By comparing the enlarged region with Fig. 3 one sees that the energy decrease is in both cases the same. The dimer next to the preset dimer (black) is not a valid dimer/energy. So the first energy calculated in Fig. 3 is the second in direction one.

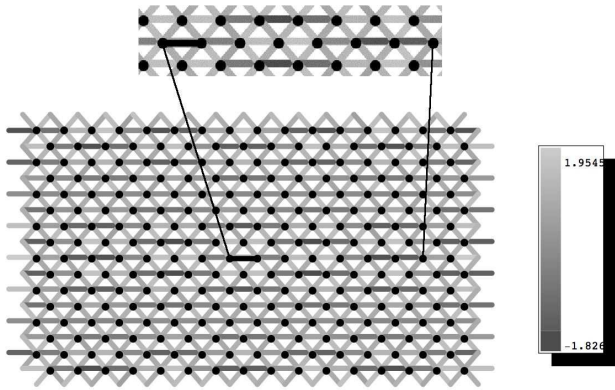


FIG. 6: Energy landscape for the  $16 \times 16$  lattice using the capacity set of the horizontal direction decay. Every black point represents a hexagon. The black line in the middle of the plot is the solid dimer. The gray lines represent the energy of the whole system when additionally two half Cooper pair are put on the respective Y-islands. The energy on the top and on the bottom as well as the one left and right are the same, to see the boundary condition. The dimer next to the black dimer is not a valid dimer/energy, as its not allowed for a hexagon to have two dimers. The first valid dimer is the second in direction one.

In this paper we have shown how to calculate the energy of a triangular lattice based on Josephson junctions analytically and numerically. This dimer model could be a non-optic and non-solid-state implementation of a quantum computer. We have seen that there is energy decrease with distance for some capacity sets and we have explicitly calculated the set for two directions in a  $16 \times 16$  lattice. Through this calculation we have seen that the capacity sets are different for different directions we are looking at. Future approaches could calculate larger systems and find faster and more exactly the optimal configuration. We also developed a method to see boundary conditions and energy landscapes visually as a grey scaled two dimensional plot.

### Acknowledgments

I would like to thank H. G. Katzgraber, my supervisor, for helpful discussion, comments and for his patience. Special thanks to G. Blatter for giving me the possibility of a 'Semesterarbeit'.

<sup>1</sup> L. B. Ioffe, M. V. Feigel'man, A. Ioselevitch, D. Ivanov, M. Troyer, and G. Blatter, *Topologically protected quantum bits using josephson junction arrays*, Letter to Nature **415**, 503 (2002).

<sup>2</sup> M. V. Feigel'man, L. B. Ioffe, V. B. Geshkenbein, P. Dayal, and G. Blatter, *Superconducting tetrahedral quantum bits*, cond-mat p. 04077663 (2004).

<sup>3</sup> D. Ivanov, L. Ioffe, V. Geshkenbein, and G. Blatter, *Interference effect in isolated josephson junction arrays with geometric symmetries*, Physical Review B **65**, 024509 (2001).

<sup>4</sup> <http://www.cpan.org>.

<sup>5</sup> <http://www.gnu.org/software/gsl/>.

<sup>6</sup> <http://www.wolfram.com>.

## APPENDIX A: PERLCODE

### 1. Lattice.pl - Capacity matrix generator

```
#!/usr/bin/perl -w

#####
#                               Lattice.pl                               #
# This script calculates the capacity matrix of a                       #
# triangular lattice. Inputs are the width and height                   #
# of the lattice (as number of hexagons), the ground                   #
# capacity the capacity between the islands on a                       #
# hexagon and between hexagons                                         #
# (h_link and b_link capacity). The output is                           #
# written in a file "capmatrix" and includes the                       #
# capacity matrix (space for rows, "\cr" for columns).                 #
# For numbering the branches see sketch below.                          #
#                               #
#       Thomas Bisig (02.2005), thomas@macapp.net                       #
#####
```



```

print("Groundcapacity: ");
chomp ( $ground = <STDIN> );

# Lattice is defined only if the rownumber
# is even, so we have to check this.

exit 0 if $height%2 != 0;

# This is the loop over all islands.

for ($k=1;$k<=6*$height*$width;++$k)
{
    $num = $k;

}

#####
#####
### We first look, whether the island is one on the border and if not ###
### look at the cases in the inner. Otherwise, we had to check everytime ###
### that we exclude the boundary (and this in every if loop!) ###
#####
#####

#####
# First we look at the upper boundary and the Y-branch numbered as five #
#####

# The first (and the last) island has to be treated specially.

if (($num)%6==5 && $num<=6*$width && $flag==0 && $num != 5)
{
    $temp=$num+1;
    $vect[$temp-1]=$h_link;
    $temp=$num-1;
    $vect[$temp-1]=$h_link;

# What island (hexagon) am I on?
# We need this to find the boundary partner

    $isle=ceil($num/6);

# Whats the number of the respective partner hexagon-island?
# We have to count all the rows before the one the partner island
# is on and add the position of the island we are looking from.

    $pisle = $width * ($height-1) + $isle - 1;

# Now we need to find the number of the corresponding Y-branch. This is exactly
# (n-th hexagon minus one) times six plus the corresponding branch (here 2)

    $temp=($pisle)*6-4;
    $vect[$temp-1]=$b_link;
    print "We are on the upper boundary on a 5-island.\n";
    $flag = 1;

```

```
}
```

```
#####  
# Next we look at the upper boundary and the Y-branch numbered as six #  
#####
```

```
if (($num)%6==0 && $num<=6*$width && $flag==0)
```

```
{  
  $temp=$num-5;  
  $vect[$temp-1]=$h_link;  
  $temp=$num-1;  
  $vect[$temp-1]=$h_link;
```

```
  $isle=ceil($num/6);
```

```
# Whats the number of the respective partner hexagon-island?  
# We have to count all the rows before the one the partner island  
# is on and add the position of the island we are looking from.  
# This is the island one BEFORE the real partner island!
```

```
  $pisle = $width * ($height-1) + $isle;
```

```
# Now we need to find the number of the corresponding Y-branch. This is exactly  
# (n-th hexagon minus one) times six plus the corresponding branch (here 3)
```

```
  $temp=($pisle)*6-3;  
  $vect[$temp-1]=$b_link;  
  print "We are on the upper boundary on a 6-island.\n";  
  $flag = 1;  
}
```

```
#####  
# Next we look at the lower boundary and the Y-branch numbered as two #  
#####
```

```
# The last island on the lattice has to be treated specially.
```

```
if (($num)%6==2 && $num>=$width*$height*6-6*$width && $flag==0 && $num != (6 * $width * $height) - 4)
```

```
{  
  $temp=$num+1;  
  $vect[$temp-1]=$h_link;  
  $temp=$num-1;  
  $vect[$temp-1]=$h_link;
```

```
  $isle=ceil($num/6);
```

```
# Whats the number of the respective partner hexagon-island?  
# Because the respective partner is in the first row, we can  
# subtract the number of rows before our island or just  
# take $isle modulo the $width of the lattice.
```

```
  $pisle = $isle % $width + 1;
```

```
# Now we need to find the number of the corresponding Y-branch. This is exactly  
# (n-th hexagon minus one) times six plus the corresponding branch (here 5)
```

```
  $temp=($pisle)*6 - 1;
```

```

    $vect[$temp-1]=$b_link;
    print "We are on the lower boundary on a 2-island.\n";
    $flag = 1;
}

```

```

#####
# Next we look at the lower boundary and the Y-branch numbered as three #
#####

```

```

if (($num)%6==3 && $num>=$width*$height*6-6*$width && $flag==0)
{
    $temp=$num+1;
    $vect[$temp-1]=$h_link;
    $temp=$num-1;
    $vect[$temp-1]=$h_link;

    $isle=ceil($num/6);

```

```

# Whats the number of the respective partner hexagon-island?
# Because the respective partner is in the first row, we can
# subtract the number of rows before our island or just
# take $isle modulo the $width of the lattice.

```

```

    $pisle = $isle - ($width * ($height - 1));

```

```

# Now we need to find the number of the corresponding Y-branch. This is exactly
# (n-th hexagon minus one) times six plus the corresponding branch (here 6)

```

```

    $temp=($pisle)*6;
    $vect[$temp-1]=$b_link;
    print "We are on the lower boundary on a 3-island.\n";
    $flag = 1;
}

```

```

# We have to look at two most special cases namely the Y-branches Nr. 5 (the
# 5. island in the first row) and Nr. ($width * $height - 4) the two island on the last row.

```

```

# First case:

```

```

if ($num == 5 && $flag == 0) {

    $temp=$num+1;
    $vect[$temp-1]=$h_link;
    $temp=$num-1;
    $vect[$temp-1]=$h_link;
    $vect[$width * $height * 6 - 4 - 1] = $b_link;
    print "Special Case!!!\n";

    $flag = 1;
}

```

```

# Second case:

```

```

if ($num == (6 * $width * $height) - 4 && $flag == 0) {

    $temp=$num+1;

```

```

    $vect[$temp-1]=$h_link;
    $temp=$num-1;
    $vect[$temp-1]=$h_link;
    $vect[5 - 1] = $b_link;
    print "Special Case!!!\n";

    $flag = 1;
}

#####
#   Next we look at the left side and the Y-branch numbered as four   #
#####

# We have to get the Y's on the left side which have the number 4

if (($num)%6==4 && ceil($num/6)%$width==1 && $flag==0)
{
    $temp=$num+1;
    $vect[$temp-1]=$h_link;
    $temp=$num-1;
    $vect[$temp-1]=$h_link;

    $isle=ceil($num/6);

# Whats the number of the respective partner hexagon-island?
# We just have to add (width - 1) to find the partner island.
# This is the island one BEFORE the real partner island!

    $pisle = $isle + $width - 2;

# Now we need to find the number of the corresponding Y-branch. This is exactly
# (n-th hexagon minus one) times six plus the corresponding branch (here 1)

    $temp=($pisle)*6+1;
    $vect[$temp-1]=$b_link;
    print "We are on the left boundary on a 4-island.\n";
    $flag = 1;
}

#####
#   Next we look at the left side and the Y-branch numbered as three   #
#####

# We have to get the Y's on the left side which have the number 3

if (($num)%6==3 && ceil($num/6)%$width==1 && $flag==0 && ceil($num/(6*$width))%2 != 0)
{
    $temp=$num+1;
    $vect[$temp-1]=$h_link;
    $temp=$num-1;
    $vect[$temp-1]=$h_link;

# What island (hexagon) am I on?
# We need this to find the boundary partner
# This is different for even/odd rownumber
# so we have to distinguish between those two options
# Here the even rows don't have to be done, they are

```

```
# the same as the general in the inner
```

```
    $isle=ceil($num/6);  
    $pisle = $isle + 2 * $width - 2;  
    $temp=($pisle)*6+6;  
    $vect[$temp-1]=$b_link;  
    print "We are on the left boundary on a 3-island.\n";  
    $flag = 1;  
}
```

```
#####  
# Next we look at the left side and the Y-branch numbered as five #  
#####
```

```
# We have to get the Y's on the left side which have the number 3  
# This is different for even/odd rownumber  
# so we have to distinguish between those two options  
# Here the even rows don't have to be done, they are  
# the same as the general in the inner
```

```
if (($num)%6==5 && ceil($num/6)%$width==1 && $flag==0 && ceil($num/(6*$width))%2 != 0)  
{  
    $temp=$num+1;  
    $vect[$temp-1]=$h_link;  
    $temp=$num-1;  
    $vect[$temp-1]=$h_link;
```

```
# What island (hexagon) am I on?  
# We need this to find the boundary partner
```

```
    $isle=ceil($num/6);  
    $pisle = $isle - 2;  
    $temp=($pisle)*6+2;  
    $vect[$temp-1]=$b_link;  
    print "We are on the left boundary on a 5-island.\n";  
    $flag = 1;  
}
```

```
#####  
# Next we look at the right side and the Y-branch numbered as one #  
#####
```

```
# We have to get the Y's on the left side which have the number 1
```

```
if (($num)%6==1 && ceil($num/6)%$width==0 && $flag==0)  
{  
    $temp=$num+1;  
    $vect[$temp-1]=$h_link;  
    $temp=$num+5;  
    $vect[$temp-1]=$h_link;  
  
    $isle=ceil($num/6);
```

```
# Whats the number of the respective partner hexagon-island?
```

```

# We just have to subtract (width - 1) to get the partner island
# This is the island one BEFORE the real partner island!

    $pisle = $isle - $width;

# Now we need to find the number of the corresponding Y-branch. This is exactly
# (n-th hexagon minus one) times six plus the corresponding branch (here 4)

    $temp=($pisle)*6+4;

    $vect[$temp-1]=$b_link;
    print "We are on the right boundary on a 1-island.\n";
    $flag = 1;
}

#####
#   Next we look at the right side and the Y-branch numbered as two   #
#####

# We have to get the Y's on the left side which have the number 2

if (($num)%6==2 && ceil($num/6)%$width==0 && $flag==0 && ceil($num/(6*$width))%2 == 0)
{
    $temp=$num+1;
    $vect[$temp-1]=$h_link;
    $temp=$num-1;
    $vect[$temp-1]=$h_link;

    $temp=$num + 9;
    $vect[$temp-1]=$b_link;
    print "We are on the right boundary on a 2-island.\n";
    $flag = 1;
}

#####
#   Next we look at the right side and the Y-branch numbered as six   #
#####

# We have to get the Y's on the left side which have the number 6

if (($num)%6==0 && ceil($num/6)%$width==0 && $flag==0 && ceil($num/(6*$width))%2 == 0)
{
    $temp=$num-5;
    $vect[$temp-1]=$h_link;
    $temp=$num-1;
    $vect[$temp-1]=$h_link;

# What island (hexagon) am I on?
# We need this to find the boundary partner

    $isle=ceil($num/6);
    $pisle = $isle - 2 * $width;
    $temp=($pisle)*6+3;
    $vect[$temp-1]=$b_link;
    print "We are on the right boundary on a 6-island.\n";
    $flag = 1;
}

```

```
}
```

```
#####  
#       Next we connect the islands in the inner of the lattice       #  
#####
```

```
$row=ceil($num/(6*$width));
```

```
if ($num%6 == 0 && $flag==0)
```

```
{
```

```
  if ($row%2 == 0)
```

```
    {
```

```
      $temp=$num-1;
```

```
      $vect[$temp-1]=$h_link;
```

```
      $temp=$num-5;
```

```
      $vect[$temp-1]=$h_link;
```

```
      $temp=$num-6*( $width-1 ) - 3;
```

```
      $vect[$temp-1]=$b_link;
```

```
      print "We are in the inner on a 6-island.\n";
```

```
      $flag = 1;
```

```
    }
```

```
  else
```

```
    {
```

```
      $temp=$num-1;
```

```
      $vect[$temp-1]=$h_link;
```

```
      $temp=$num-5;
```

```
      $vect[$temp-1]=$h_link;
```

```
      $temp=$num - 6*$width - 3;
```

```
      $vect[$temp-1]=$b_link;
```

```
      print "We are in the inner on a 6-island.\n";
```

```
      $flag = 1;
```

```
    }
```

```
}
```

```
if ($num%6 == 5 && $flag==0)
```

```
{
```

```
  if ($row%2 == 0)
```

```
    {
```

```
      $temp=$num-1;
```

```
      $vect[$temp-1]=$h_link;
```

```
      $temp=$num+1;
```

```
      $vect[$temp-1]=$h_link;
```

```
      $temp=$num - 6*$width - 3;
```

```
      $vect[$temp-1]=$b_link;
```

```
      print "We are in the inner on a 5-island.\n";
```

```
      $flag = 1;
```

```
    }
```

```
  else
```

```
    {
```

```
      $temp=$num-1;
```

```
      $vect[$temp-1]=$h_link;
```

```
      $temp=$num+1;
```

```
      $vect[$temp-1]=$h_link;
```

```
      $temp=$num - 6*( $width+1 ) - 3;
```

```
      $vect[$temp-1]=$b_link;
```

```
      print "We are in the inner on a 5-island.\n";
```

```
      $flag = 1;
```

```
    }
```

```

    }
}

if ($num%6 == 4 && $flag==0)
{
    $temp=$num-1;
    $vect[$temp-1]=$h_link;
    $temp=$num+1;
    $vect[$temp-1]=$h_link;
    $temp=$num - 9;
    $vect[$temp-1]=$b_link;
    print "We are in the inner on a 4-island.\n";
    $flag = 1;
}

if ($num%6 == 3 && $flag==0)
{
    if ($row%2 == 0)
    {
        $temp=$num-1;
        $vect[$temp-1]=$h_link;
        $temp=$num+1;
        $vect[$temp-1]=$h_link;
        $temp=$num + 6*$width + 3;
        $vect[$temp-1]=$b_link;
        print "We are in the inner on a 3-island.\n";
        $flag = 1;
    }
    else
    {
        $temp=$num-1;
        $vect[$temp-1]=$h_link;
        $temp=$num+1;
        $vect[$temp-1]=$h_link;
        $temp=$num + 6*( $width - 1 ) + 3;
        $vect[$temp-1]=$b_link;
        print "We are in the inner on a 3-island.\n";
        $flag = 1;
    }
}

if ($num%6 == 2 && $flag==0)
{
    if ($row%2 == 0)
    {
        $temp=$num-1;
        $vect[$temp-1]=$h_link;
        $temp=$num+1;
        $vect[$temp-1]=$h_link;
        $temp=$num + 6*( $width + 1 ) + 3;
        $vect[$temp-1]=$b_link;
        print "We are in the inner on a 2-island.\n";
        $flag = 1;
    }
    else
    {
        $temp=$num-1;
        $vect[$temp-1]=$h_link;
    }
}

```

```

        $temp=$num+1;
        $vect[$temp-1]=$h_link;
        $temp=$num + 6*( $width) + 3;
        $vect[$temp-1]=$b_link;
        print "We are in the inner on a 2-island.\n";
        $flag = 1;
    }

}

if ($num%6 == 1 && $flag==0)
{
    $temp=$num+5;
    $vect[$temp-1]=$h_link;
    $temp=$num+1;
    $vect[$temp-1]=$h_link;
    $temp=$num + 9;
    $vect[$temp-1]=$b_link;
    print "We are in the inner on a 1-island.\n";
    $flag = 1;
}

$vect[$num-1]=$ground;

# Print the capacity matrix into a file called capmatrix

open (APPEND, ">>capmatrix") or die "$! error trying to append";
foreach (@vect) {
    print APPEND "$_ ";
}
print APPEND "\n";

# We have to set the flag back to zero
# otherwise no loop would be executed anymore

$flag = 0;
}

```

## 2. Dimers.pl - Finds all pairs of Y-islands

```

#!/usr/bin/perl -w

#####
#          Dimer.pl          #
# Calculates a list of Y-Islands which form a dimer #
# Output is a list to read into Mathematica.      #
#          #          #
#      Thomas Bisig (02.2005), thomas@macapp.net  #
#####

#####
#          Comments can be found in Lattice.pl    #
#####

use POSIX qw(ceil);
use POSIX qw(floor);

```

```

$width = 0;
$height = 0;
$flag=0;
$string = "";

$num = 0;

print("Width of lattice: ");
chomp ( $width = <STDIN> );
print("Height of lattice: ");
chomp ( $height = <STDIN> );

exit 0 if $height%2 != 0;

for ($k=1;$k<=6*$height*$width;++$k)
{
    $string = "";
    $num = $k;
    $string = "$num";

#####
# First we look at the upper boundary and the Y-branch numbered as five #
#####

if (($num)%6==5 && $num<=6*$width && $flag==0 && $num != 5)
{
    $isle=ceil($num/6);
    $pisle = $width * ($height-1) + $isle - 1;
    $temp=($pisle)*6-4;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

#####
# Next we look at the upper boundary and the Y-branch numbered as six #
#####

if (($num)%6==0 && $num<=6*$width && $flag==0)
{
    $isle=ceil($num/6);
    $pisle = $width * ($height-1) + $isle;
    $temp=($pisle)*6-3;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

#####
# Next we look at the lower boundary and the Y-branch numbered as two #
#####

if (($num)%6==2 && $num>=$width*$height*6-6*$width && $flag==0 && $num != (6 * $width * $height) - 4)
{
    $isle=ceil($num/6);
    $pisle = $isle % $width + 1;
    $temp=($pisle)*6 - 1;
    $string = $string . " " . "$temp" . "\n";
}

```

```

    $flag = 1;
}

#####
# Next we look at the lower boundary and the Y-branch numbered as three #
#####

if (($num)%6==3 && $num>=$width*$height*6-6*$width && $flag==0)
{
    $isle=ceil($num/6);
    $pisle = $isle - ($width * ($height -1));
    $temp=($pisle)*6;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

# We have to look at two most special cases namely the Y-branches Nr. 5 (the
# 5. island in the first row) and Nr. ($width * $height -4) the two island on the last row.

# First case:
if ($num == 5 && $flag == 0) {
    $temp=$width * $height * 6 - 4;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

# Second case:
if ($num == (6 * $width * $height) - 4 && $flag == 0) {
    $temp = 5;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

#####
# Next we look at the left side and the Y-branch numbered as four #
#####

if (($num)%6==4 && ceil($num/6)%$width==1 && $flag==0)
{
    $isle=ceil($num/6);
    $pisle = $isle + $width - 2;
    $temp=($pisle)*6+1;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

#####
# Next we look at the left side and the Y-branch numbered as three #
#####

if (($num)%6==3 && ceil($num/6)%$width==1 && $flag==0 && ceil($num/(6*$width))%2 != 0)
{
    $isle=ceil($num/6);
    $pisle = $isle + 2 * $width - 2;
    $temp=($pisle)*6+6;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

```

```
}
```

```
#####  
# Next we look at the left side and the Y-branch numbered as five #  
#####
```

```
if (($num)%6==5 && ceil($num/6)%$width==1 && $flag==0 && ceil($num/(6*$width))%2 != 0)  
{  
  $isle=ceil($num/6);  
  $pisle = $isle - 2;  
  $temp=($pisle)*6+2;  
  $string = $string . " " . "$temp" . "\n";  
  $flag = 1;  
}
```

```
#####  
# Next we look at the right side and the Y-branch numbered as one #  
#####
```

```
if (($num)%6==1 && ceil($num/6)%$width==0 && $flag==0)  
{  
  $isle=ceil($num/6);  
  $pisle = $isle - $width;  
  $temp=($pisle)*6+4;  
  $string = $string . " " . "$temp" . "\n";  
  $flag = 1;  
}
```

```
#####  
# Next we look at the right side and the Y-branch numbered as two #  
#####
```

```
if (($num)%6==2 && ceil($num/6)%$width==0 && $flag==0 && ceil($num/(6*$width))%2 == 0)  
{  
  $temp=$num + 9;  
  $string = $string . " " . "$temp" . "\n";  
  $flag = 1;  
}
```

```
#####  
# Next we look at the right side and the Y-branch numbered as six #  
#####
```

```
if (($num)%6==0 && ceil($num/6)%$width==0 && $flag==0 && ceil($num/(6*$width))%2 == 0)  
{  
  $isle=ceil($num/6);  
  $pisle = $isle - 2 * $width;  
  $temp=($pisle)*6+3;  
  $string = $string . " " . "$temp" . "\n";  
  $flag = 1;  
}
```

```
#####  
# The inner islands #  
#####
```

```

$row=ceil($num/(6*$width));

if ($num%6 == 0 && $flag==0)
{
  if ($row%2 == 0)
  {
    $temp=$num-6*( $width-1 ) - 3;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
  }
  else
  {
    $temp=$num - 6*$width - 3;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
  }
}

if ($num%6 == 5 && $flag==0)
{
  if ($row%2 == 0)
  {
    $temp=$num - 6*$width - 3;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
  }
  else
  {
    $temp=$num - 6*( $width+1 ) - 3;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
  }
}

if ($num%6 == 4 && $flag==0)
{
  $temp=$num - 9;
  $string = $string . " " . "$temp" . "\n";
  $flag = 1;
}

if ($num%6 == 3 && $flag==0)
{
  if ($row%2 == 0)
  {
    $temp=$num + 6*$width + 3;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
  }
  else
  {
    $temp=$num + 6*( $width - 1 ) + 3;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
  }
}

if ($num%6 == 2 && $flag==0)

```

```

    {
    if ($row%2 == 0)
        {
        $temp=$num + 6* ( $width + 1 ) + 3;
        $string = $string . " " . "$temp" . "\n";
        $flag = 1;
        }
    else
        {
        $temp=$num + 6*( $width) + 3;
        $string = $string . " " . "$temp" . "\n";
        $flag = 1;
        }
    }

if ($num%6 == 1 && $flag==0)
    {
    $temp=$num + 9;
    $string = $string . " " . "$temp" . "\n";
    $flag = 1;
    }

open (APPEND, ">>dimerlist") or die "$! error trying to append";
print APPEND $string;
$flag = 0;
}

```

### 3. coord.pl - Coordinates to draw the energy landscape

```

#!/usr/bin/perl -w

#####
#                               coord.pl                               #
# Calculates the position of the middle of the                       #
# dimers. This is for plotting purpose only.                         #
#                               #                                       #
#   Thomas Bisig (02.2005), thomas@macapp.net                       #
#####

$width = 4;
$height = 4;
$dist = 2;
$Pi = 3.1415926;

$x = 1;
$y = 1;

# Loops over all positions. Easy to see with a little
# bit of geometry. No comments.

for( $h = 0; $h < $height; $h++ )
{
    for( $k = 0; $k < $width; $k++ )
    {
        $x_coord = $x + $k*2*$dist;
        $y_coord = $y - $h * $dist*sqrt(3);
    }
}

```

```

if ( $h % 2 != 0 )
{
    $x_coord += $dist;
}

for( $p = 0; $p < 6; $p++ )
{
    $x_temp=$x_coord;
    $y_temp=$y_coord;

    $x_temp += $dist * cos((6-$p)*2*$Pi/6);
    $y_temp += $dist * sin((6-$p)*2*$Pi/6);

    $string = $x_temp . " " . $y_temp . " " . 0 . "\n";
    open (APPEND, ">>coord") or die "$! error trying to append";
    print APPEND $string;
}
}
}

```

#### 4. Columnar.pl - Defining columnar setup

```

#!/usr/bin/perl -w

#####
#                               #
#      Columnar.pl              #
# Calculates the positions of the dimers for a columnar setup #
# This script takes as inputs the width and height as inputs #
# and outputs the columnar dimer setup                        #
#                               #
#      Thomas Bisig (02.2005), thomas@macapp.net             #
#####

#####
#      For comments read Lattice.pl or Dimers.pl            #
#####

use POSIX qw(ceil);
use POSIX qw(floor);

$width = 0;
$height = 0;
$flag=0;
$string = "";

$num = 0;

print("Width of lattice: ");
chomp ( $width = <STDIN> );

```

```

print("Height of lattice: ");
chomp ( $height = <STDIN> );

# Fill the vector with zeros
@vect[$width*$height*6-1];
for ($i=0;$i<$width*$height*6;$i++)
{
    $vect[$i]=0;
}

exit 0 if $height%2 != 0;

# Loops over all 1-islands

for ($k=1;$k<=6*$height*$width; $k += 12)
{
    if ( $k % ( 2 * $width * 6 ) == 1 && $k % ( 2 * 2 * $width * 6 ) != 1 )
    {
        $k += 6;
    }

    if ( $k % ( 2 * $width * 6 ) == 7 && $k % ( 2 * 2 * $width * 6 ) == 7 )
    {
        $k -= 6;
    }

    $string = "";
    $num = $k;
    $string = "$num";

# We only have to check the linking of the 1-islands (easy)

if (($num)%6==1 && ceil($num/6)%$width==0 && $flag==0)
{
    $isle=ceil($num/6);
    $pisle = $isle - $width;
    $temp=($pisle)*6+4;
    $vect[$temp-1]=1;
    $vect[$num-1]=1;
    #$string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

if ($num%6 == 1 && $flag==0)
{
    $temp=$num + 9;
    $vect[$temp-1]=1;
    $vect[$num-1]=1;
    #$string = $string . " " . "$temp" . "\n";
    $flag = 1;
}

$flag = 0;
}

#open (APPEND, ">>columnarlist") or die "$! error trying to append";
#print APPEND $string;

```

```

open (APPEND, ">>columnarlist") or die "$! error trying to append";
foreach (@vect) {
    print APPEND "$_";
}

```

## 5. RandomLattice.pl - Generates a random lattice

```

#!/usr/bin/perl -w

#####
#                               RandomLattice.pl                               #
# Random lattice generator. We only look at the connections #
# from an island with number 1,2 or 3 (random).           #
# This script takes the width and height of a lattice as  #
# input and outputs the random lattice configuration       #
#                                                         #
#           Thomas Bisig (02.2005), thomas@macapp.net      #
#####

use POSIX qw(ceil);
use POSIX qw(floor);

# Lattice Properties:
# width and height are the number of _hexagons_
# not the number of Y-branches!

$width = 0;
$height = 0;
$flag=0;

# Which number we look at

$num = 0;
$NeighbourIsOk = 0;

# Define the properties of the lattice

print("Width of lattice: ");
chomp ( $width = <STDIN> );
print("Height of lattice: ");
chomp ( $height = <STDIN> );

# Lattice is defined only if the rownumber
# is even, so we have to check this.

exit 0 if $height%2 != 0;

# We need to check in every loop if the corresponding island allows us (1) to make connection
# or whether it already has one (0)

for ($i=0;$i<$width*$height;$i++)

```

```

    {
        $check[$i]=1;
    }

for ($i=0;$i<$width*$height*6;$i++)
{
    $vect[$i]=0;
}

# Loop over all islands
for ($k=0;$k<$height*$width;++$k)
{
# Setting the NeighbourIsOk value to zero again
    $NeighbourIsOk = 0;

# Check whether we can use this island
    if ($check[$k]==1 && &HasAvailableNeighbour($k) == 1)
        {
            while ( $NeighbourIsOk == 0 )
                {

                    $num = $k*6+int( rand(3)) + 1;

#####
#                               For comments read Lattice.pl                               #
#####

#####
# We look at the lower boundary and the Y-branch numbered as two #
#####

if (($num)%6==2 && $num>=$width*$height*6-6*$width && $flag==0 && $num != (6 * $width * $height) - 4)
    {
        $isle=ceil($num/6);
        $pisle = $isle % $width + 1;

        if ($check[$pisle-1]==1)
            {
                $temp=($pisle)*6 - 1;
                $vect[$temp-1]=1;
                $vect[$num-1]=1;
                $check[$pisle-1]=0;
                $check[ceil($num/6)-1]=0;
                $NeighbourIsOk = 1;
            }

        $flag = 1;
    }

#####
# We look at the lower boundary and the Y-branch numbered as three #
#####

if (($num)%6==3 && $num>=$width*$height*6-6*$width && $flag==0)
    {

```

```

$isle=ceil($num/6);
$pisle = $isle - ($width * ($height -1));

if ($check[$pisle-1]==1)
{
    $temp=($pisle)*6;
    $vect[$temp-1]=1;
    $vect[$num-1]=1;
    $check[$pisle-1]=0;
    $check[ceil($num/6)-1]=0;
    $NeighbourIsOk = 1;
}

$flag = 1;
}

if ($num == (6 * $width * $height) - 4 && $flag == 0) {

    if ($check[0]==1)
    {
        $vect[5-1]=1;
        $vect[5-1]=1;
        $check[0]=0;
        $check[ceil($num/6)-1]=0;
        print "Special Case!!! This should NEVER SHOW UP!!\n";
        $NeighbourIsOk = 1;
    }

    $flag = 1;
}

#####
#   We look at the left side and the Y-branch numbered as three   #
#####

if (($num)%6==3 && ceil($num/6)%$width==1 && $flag==0 && ceil($num/(6*$width))%2 != 0)
{
    $isle=ceil($num/6);
    $pisle = $isle + 2 * $width - 2 + 1;

    if ($check[$pisle-1]==1)
    {
        $temp=($pisle)*6;
        $vect[$temp-1]=1;
        $vect[$num-1]=1;
        $check[$pisle-1]=0;
        $check[ceil($num/6)-1]=0;
        $NeighbourIsOk = 1;
    }
    $flag = 1;
}

#####
#   We look at the right side and the Y-branch numbered as one   #
#####

```

```

if (($num)%6==1 && ceil($num/6)%$width==0 && $flag==0)
{
    $isle=ceil($num/6);
    $pisle = $isle - $width;

    if ($check[$pisle + 1 - 1]==1)
    {
        $temp=($pisle)*6+4;
        $vect[$temp-1]=1;
        $vect[$num-1]=1;
        $check[$pisle + 1 - 1]=0;
        $check[ceil($num/6)-1]=0;
        $NeighbourIsOk = 1;
    }
    $flag = 1;
}

#####
# We look at the right side and the Y-branch numbered as two #
#####

if (($num)%6==2 && ceil($num/6)%$width==0 && $flag==0 && ceil($num/(6*$width))%2 == 0)
{
    $pisle = ceil($num/6) + 1;

    if ($check[$pisle-1]==1)
    {
        $temp=$num + 9;
        $vect[$temp-1]=1;
        $vect[$num-1]=1;
        $check[$pisle-1]=0;
        $check[ceil($num/6)-1]=0;
        $NeighbourIsOk = 1;
    }
    $flag = 1;
}

#####
# The next six if-loops are for the linking of the "inner" islands #
#####

$row=ceil($num/(6*$width));

if ($num%6 == 3 && $flag==0)
{
    if ($row%2 == 0)
    {
        $temp=$num + 6*$width + 3;
        $pisle = ceil($temp/6);

        if ($check[$pisle-1]==1)
        {
            $vect[$temp-1]=1;
            $vect[$num-1]=1;
            $check[$pisle-1]=0;
        }
    }
}

```

```

        $check[ceil($num/6)-1]=0;
        $NeighbourIsOk = 1;
    }

    $flag = 1;
}

else
{

    $temp=$num + 6*( $width - 1 ) + 3;
    $pisle = ceil($temp/6);

        if ($check[$pisle-1]==1)
        {
            $vect[$temp-1]=1;
            $vect[$num-1]=1;
            $check[$pisle-1]=0;
            $check[ceil($num/6)-1]=0;
            $NeighbourIsOk = 1;
        }

    $flag = 1;
}

}

if ($num%6 == 2 && $flag==0)
{
    if ($row%2 == 0)
    {
        $temp=$num + 6* ( $width + 1 ) + 3;
        $pisle = ceil($temp/6);

            if ($check[$pisle-1]==1)
            {
                $vect[$temp-1]=1;
                $vect[$num-1]=1;
                $check[$pisle-1]=0;
                $check[ceil($num/6)-1]=0;
                $NeighbourIsOk = 1;
            }

        $flag = 1;
    }
    else
    {
        $temp=$num + 6*( $width) + 3;
        $pisle = ceil($temp/6);

            if ($check[$pisle-1]==1)
            {
                $vect[$temp-1]=1;
                $vect[$num-1]=1;
                $check[$pisle-1]=0;
                $check[ceil($num/6)-1]=0;
                $NeighbourIsOk = 1;
            }

        $flag = 1;
    }
}

```

```

    }
}

if ($num%6 == 1 && $flag==0)
{
    $temp=$num + 9;
    $pisle = ceil($temp/6);

    if ($check[$pisle-1]==1)
    {
        $vect[$temp-1]=1;
        $vect[$num-1]=1;
        $check[$pisle-1]=0;
        $check[ceil($num/6)-1]=0;
        $NeighbourIsOk = 1;
    }

    $flag = 1;
}

$flag = 0;
}
}

}

# Prints the charge vector into randomlist

open (APPEND, ">>randomlist") or die "$! error trying to append";

foreach (@vect) {
    print APPEND"$_";
}

#####
# Checks whether there is at least one free neighbour      #
# Returns 0 if no neighbour is free, in the other          #
# case returns 1                                           #
#####

sub HasAvailableNeighbour {
    my ($n) = @_;
    my ($o) = 0;
    my ($count) = 0;
    my ($nu) = 0;
    my ($flag2) = 0;
    my ($pisle2) = 0;
    my ($temp2) = 0;
    my ($isle2) = 0;

    for ($o=1; $o < 4; $o++) {

        $nu = $n * 6 + $o;

        if (($nu)%6==2 && $nu>=$width*$height*6-6*$width && $flag2==0 && $nu != (6 * $width * $height) - 4)
        {
            $isle2=ceil($nu/6);

```

```

        $pisle2 = $isle2 % $width + 1;
        $flag2 = 1;
    }

    if (($nu)%6==3 && $nu>=$width*$height*6-6*$width && $flag2==0)
    {
        $isle2=ceil($nu/6);
        $pisle2 = $isle2 - ($width * ($height -1));
        $flag2 = 1;
    }

    if ($nu == (6 * $width * $height) - 4 && $flag2 == 0)
    {
        $pisle2 = 1;
        $flag2 = 1;
    }

    if (($nu)%6==3 && ceil($nu/6)%$width==1 && $flag2==0 && ceil($nu/(6*$width))%2 != 0)
    {
        $isle2=ceil($nu/6);
        $pisle2 = $isle2 + 2 * $width - 2 + 1;
        $flag2 = 1;
    }

    if (($nu)%6==1 && ceil($nu/6)%$width==0 && $flag2==0)
    {
        $isle2=ceil($nu/6);
        $pisle2 = $isle2 - $width + 1;
        $flag2 = 1;
    }

    if (($nu)%6==2 && ceil($nu/6)%$width==0 && $flag2==0 && ceil($nu/(6*$width))%2 == 0)
    {
        $pisle2 = ceil($nu/6) + 1;
        $flag2 = 1;
    }

    $row=ceil($nu/(6*$width));

    if ($nu%6 == 3 && $flag2==0)
    {
        if ($row%2 == 0)
        {
            $temp2=$nu + 6*$width + 3;
            $pisle2 = ceil($temp2/6);
            $flag2 = 1;
        }
        else
        {
            $temp2=$nu + 6*( $width - 1 ) + 3;
            $pisle2 = ceil($temp2/6);
            $flag2 = 1;
        }
    }

```

```

    }
}

if ($nu%6 == 2 && $flag2==0)
{
    if ($row%2 == 0)
    {
        $temp2=$nu + 6* ( $width + 1 ) + 3;
        $pisle2 = ceil($temp2/6);
        $flag2 = 1;
    }
    else
    {
        $temp2=$nu + 6*( $width) + 3;
        $pisle2 = ceil($temp2/6);
        $flag2 = 1;
    }
}

if ($nu%6 == 1 && $flag2==0)
{
    $temp2=$nu + 9;
    $pisle2 = ceil($temp2/6);
    $flag2 = 1;
}

$count += $check[$pisle2-1];

$flag2 = 0;
}

if ($count==0) {
return 0;
}
else {
return 1;
}
}
}

```

## APPENDIX B: MATHEMATICA SHEET

### 1. ELandscape.m - Plotting the energy landscape

(\* Calculates the Energy at a specific position (x,y) and puts everything into a file MathOutSurface. Look at the file with gnuplot splot 'MathOutSurface'. The electric charges (the set dimer) is on the island 55-64 \*)

```

Clear[data];
Clear[data2];
Clear[dimerlist];
Clear[l];

```

```

data = ReadList["/home/bisigt/Semesterarbeit/Scaling/4-4/tempfiles/coord",
  Number, RecordLists \[Rule] True];
dimerlist =
  ReadList["/home/bisigt/Semesterarbeit/Scaling/4-4/tempfiles/dimerlist",
    Number, RecordLists \[Rule] True];

inv =ReadList["/home/bisigt/Semesterarbeit/Scaling/4-4/tempfiles/invmat",
  Number, RecordLists \[Rule] True];

o=Table[0,{96}];
f[i_]:= (
  Clear[l];
  l=Join[Table[0.,{54}],{1},Table[0.,{8}],{1},Table[0.,{32}]];
  l[[dimerlist[[i,1]]]]=1;
  l[[dimerlist[[i,2]]]]=1;

  p=0.5(1 . inv . l);
  data[[i,3]]=p;
  o[[i]]=p;
)

For[i=1,i\[LessEqual]96,f[i];i++];

maximum=Max[o];
minimum =Min[o];

r = Abs[minimum] + maximum;

u[i_]:= (
  Clear[temp];
  x=data[[i,1]];
  y=data[[i,2]];
  z=data[[i,3]]+Abs[minimum];
  s=Mod[i,6];
  temp=Which[s==1,{{x-2,y},{x+2,y}},s==2,{{x-1,y+1.75},{x+1,y-1.75}},
    s\[Equal]3,{{x+1,y+1.75},{x-1,y-1.75}},s\[Equal]4,{{x-2,y},{x+2,y}},
    s\[Equal]5,{{x+1,y-1.75},{x-1,y+1.75}},
    s\[Equal]0,{{x-1,y-1.75},{x+1,y+1.75}}];
  temp={Hue[z/r *0.75],Line[temp]};
  q=Join[q,temp];
)

For[i=1,i\[LessEqual]96,u[i];i++];

p={Thickness[0.035],Line[{{5,-6},{9,-6}}]};

s1={PointSize[0.04],Table[Table[Point[{{i*4+1, 1-k*7}],{i,0,3}], {k,0,1}]}];
s2={PointSize[0.04],
  Table[Table[Point[{{i*4+3, -2.5 - k*7}],{i,0,3}], {k,0,1}]}];

<<Graphics'Legend'
ShowLegend[
  Show[Graphics[q],Graphics[s1],Graphics[s2],Graphics[p],
    PlotRange \[Rule] {{-2,18},{-13,5}},{Hue[0.75-0.75#]&,10,
    ToString[maximum],ToString[minimum],LegendPosition\[Rule]{1.1,-.4}]]

```

## 1. matrix.c - Calculating energy

```

#include <stdio.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_cblas.h>
#include <gsl/gsl_const_mkssa.h>
#include <math.h>

int
main (void)
{
    // defining constants and variables
    int s, i, j;
    int n = 1536;
    double e = GSL_CONST_MKSA_ELECTRON_CHARGE;
    double result[7];

    // define all the used matrices
    gsl_matrix * m = gsl_matrix_alloc (n, n);
    gsl_matrix * inverse = gsl_matrix_alloc (n, n);
    gsl_matrix * dotproduct = gsl_matrix_alloc (n, n);
    gsl_matrix * temp = gsl_matrix_alloc (n, n);
    gsl_permutation * perm = gsl_permutation_alloc (n);

    // define vectors
    gsl_vector * q[7];
    gsl_vector * x[7];

    int o=0;
    for (o;o<8;++o)
    {
        q[o]=gsl_vector_calloc (n);
        x[o]=gsl_vector_calloc (n);
    }

    // setting the 'solid' dimer
    int r=1;
    for (r; r< 8; ++r)
    {
        gsl_vector_set (q[r], 1, e);
        gsl_vector_set (q[r], 10, e);
    }

    // setting the first dimer
    gsl_vector_set (q[1], 13, e);
    gsl_vector_set (q[1], 22, e);

    // setting the second dimer
    gsl_vector_set (q[2], 19, e);
    gsl_vector_set (q[2], 28, e);

    // setting the third dimer

```

```

gsl_vector_set (q[3], 25, e);
gsl_vector_set (q[3], 34, e);

// setting the fourth dimer
gsl_vector_set (q[4], 31, e);
gsl_vector_set (q[4], 40, e);

// setting the fifth dimer
gsl_vector_set (q[5], 37, e);
gsl_vector_set (q[5], 46, e);

// setting the sixth dimer
gsl_vector_set (q[6], 43, e);
gsl_vector_set (q[6], 52, e);

// setting the seventh dimer
gsl_vector_set (q[7], 49, e);
gsl_vector_set (q[7], 58, e);

// open file with matrix stored
FILE * f = fopen ("capmatrix","r");
gsl_matrix_fscanf (f, m);
fclose (f);

// make a copy of the initial matrix
gsl_matrix_memcpy (temp, m);

// make LU decomposition of m (saved in m!?)
gsl_linalg_LU_decomp (m, perm, &s);

// invert the matrix m
gsl_linalg_LU_invert (m, perm, inverse);

int k=1;
for ( k; k <8; ++k )
{
// matrix vector product
gsl_blas_dgemv (CblasNoTrans, 1.0, inverse, q[k], 0.0, x[k]);

// vector vector product
gsl_blas_ddot ( q[k], x[k], &result[k]);
}

int u=1;
for ( u; u <8; ++u )
{
result[u] /= 2.0;
result[u] /= GSL_CONST_MKSA_ELECTRON_VOLT;
printf ("----- The energy is %g eV\n", result[u]);
}

double optimize_var;
double average_var;

int p=2;

for ( p ; p<8 ; ++p )

```

```

{
    average_var += result[p];
}

average_var /= 6;

// Somehow, abs() does not work
double a;

if (result[1]<0)
{
a=-result[1];
}
else
{
a=result[1];
}

optimize_var = (result[1]-average_var)/a;

FILE * g = fopen ("plotdata","w+");
fprintf (g, "1\t%.10g\n", result[1]);
fprintf (g, "2\t%.10g\n", result[2]);
fprintf (g, "3\t%.10g\n", result[3]);
fprintf (g, "4\t%.10g\n", result[4]);
fprintf (g, "5\t%.10g\n", result[5]);
fprintf (g, "6\t%.10g\n", result[6]);
fprintf (g, "7\t%.10g", result[7]);
fclose (g);

printf( "Optimization variable: %g\n\n", optimize_var);

FILE * h = fopen ("log","aw");

fprintf (h, "%g\n", optimize_var);

fclose(h);
}

```

## APPENDIX D: EXECUTION FILE (BASH)

### 1. makeworking.sh - Bash file

```

#!/bin/bash

declare count="0"
declare count2="0"
declare count3="0"
VAR="rename.ps"
DIRNAME="qq-rr"

list=(1e-4 0.5e-4 1e-5 0.5e-5 1e-6 0.5e-6 1e-7 0.5e-7 1e-8 0.5e-8 1e-9 0.5e-9)

gcc matrix2.c -lgs1 -lm -lgs1cblas
echo "x Compiling complete"

```

```
echo
```

```
while [ $count3 -lt 12 ]
```

```
do
```

```
h=${list[$count3]}
```

```
while [ $count2 -lt 12 ]
```

```
do
```

```
STR2=${DIRNAME/qq/$count3}
```

```
STR2=${STR2/rr/$count2}
```

```
mkdir $STR2
```

```
g=${list[$count2]}
```

```
while [ $count -lt 12 ]
```

```
do
```

```
l=${list[$count]}
```

```
echo "Hexagon Capacity is now: " $h
```

```
echo "Link Capacity is now: " $l
```

```
echo "Ground Capacity is now:" $g
```

```
perl Lattice.pl $h $l $g
```

```
echo "x Matrix created"
```

```
./a.out
```

```
echo "x Energy calculated"
```

```
gnuplot < gn
```

```
echo "x Plotted"
```

```
STR=${VAR/rename/$count}
```

```
mv "plot.ps" $STR
```

```
cp $STR $STR2
```

```
rm $STR
```

```
rm capmatrix
```

```
cd $STR2
```

```
ps2pdf $STR
```

```
rm $STR
```

```
cd ../
```

```
echo "x Cleaned up"
```

```
count=$((count+1))
```

```
echo "x Finished this loop"
```

```
echo
```

```
done
```

```
count2=$((count2+1))
```

```
count=$count3
```

```
done
```

```
count3=$((count3+1))
```

```
count2=$count3
```

```
done
```

```
echo "-----"
```

```
echo "Finished!"
```