IOPscience iopscience.iop.org

Home Search Collections Journals About Contact us My IOPscience

Exact enumeration of Hamiltonian walks on the 4 × 4 × 4 cube and applications to protein folding

This content has been downloaded from IOPscience. Please scroll down to see the full text.

2013 J. Phys. A: Math. Theor. 46 485001

(http://iopscience.iop.org/1751-8121/46/48/485001)

View the table of contents for this issue, or go to the journal homepage for more

Download details:

IP Address: 128.104.46.196
This content was downloaded on 06/01/2016 at 20:45

Please note that terms and conditions apply.

IOP Publishing Journal of Physics A: Mathematical and Theoretical

J. Phys. A: Math. Theor. **46** (2013) 485001 (14pp) doi:10.1088/1751-8113/46/48/485001

Exact enumeration of Hamiltonian walks on the 4 × 4 × 4 cube and applications to protein folding

Raoul D Schram and Helmut Schiessel

Instituut-Lorentz, Leiden University, PO Box 9506, 2300-RA Leiden, The Netherlands

E-mail: schram@lorentz.leidenuniv.nl and schiessel@leidenuniv.nl

Received 9 July 2013, in final form 7 October 2013
Published 12 November 2013
Online at stacks.iop.org/JPhysA/46/485001

Abstract
Hamiltonian walks on lattices are model systems for compact polymers such as proteins. Here we enumerate exactly the number of Hamiltonian walks on the $4 \times 4 \times 4$ cube and give estimates up to the $7 \times 7 \times 7$ cube through Monte Carlo methods. We find that the number of configurations grows faster with chain length than previously anticipated. Finally, we discuss uniqueness of ground states in the HP model for protein folding.

PACS numbers: 87.14.et, 61.25.H−, 36.20.Ey

(Some figures may appear in colour only in the online journal)

1. Introduction

Hamiltonian walks on compact square and cubic sublattices (walks that visit every vertex once and only once) have been studied since the late 1980s [1–4] as idealized conformations of compact proteins. Proteins fold from a denatured state to a compact state through hydrophilic and hydrophobic interactions. One of the main mysteries in biophysics is the question of how proteins fold always to their native state without getting stuck in one of the local minima. By modeling Hamiltonian walks as heteropolymers and thereby achieving a rough representation of a polypeptide chain, one hopes to answer such questions. A popular model of that kind is the HP model [1], in which the monomers of the polymer are either polar (P) or hydrophobic (H). This model suggests that folding is guided toward compact configurations via the hydrophobic interactions leading to a funnel-like free energy landscape [5]. Moreover, it was found that the fraction of HP sequences with a unique ground state stays roughly constant with chain length [5].

Exact enumeration is crucial since otherwise the ground state of a polymer might be missed. What makes enumeration difficult is the exponential increase of the number of configurations with chain length. This is the reason why a lot of effort has been focussed on two-dimensional Hamiltonian walks on square lattices where the number grows slower than for their three-dimensional counterparts. An exact enumeration of Hamiltonian walks for all compact shapes on square lattices has been presented in [2] up to length 30 and also for

 © 2013 IOP Publishing Ltd Printed in the UK & the USA

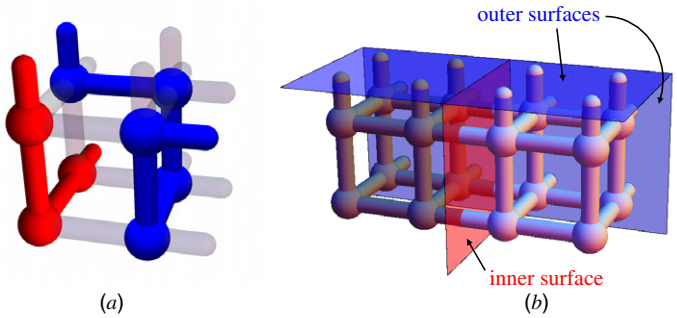

Figure 1. (*a*) Example configuration of a $2 \times 2 \times 2$ cube with two chain pieces (red and blue). (*b*) Definition of the inner and the outer surfaces when combining two $2 \times 2 \times 2$ cubes into a $2 \times 2 \times 4$ cuboid.

length 36; see also [6]. Hamiltonian walks on $L \times L$ lattices up to size $L = 17$ have been enumerated in [7] (see table 2 in that paper).

On the other hand, for the three-dimensional case the exact enumeration for the $3 \times 3 \times 3$ cube was presented in [3, 4]; see also [8–10]. Pande *et al* [11] computed also the numbers for the $3 \times 3 \times 4$ and for the $3 \times 4 \times 4$ cubic sublattices but concluded that 'it seems that the enumeration of the Hamiltonian walks on the $4 \times 4 \times 4$ sublattice is several orders of magnitude out of reach using our current algorithm and supercomputer power'. Exact enumeration of the $4 \times 4 \times 4$ was presented only possible for a geometry with a hole in [12]. The exponential increase of the number of Hamiltonian walks as a function of the number of lattice sites has so far impeded further advance. The algorithms available were either too time consuming [11] or too memory intensive [7] to go beyond that size.

In section 2 we present an algorithm that changes this, and the (much) larger size of $4 \times 4 \times 4$ can be computed in several hours on a single PC. Additionally we present a Monte Carlo algorithm in section 3 both to verify the results of the exact enumeration algorithm, and to extend the size up to $7 \times 7 \times 7$. Inspired by the much faster increase of the number of Hamiltonian walks with length than earlier anticipated, we discuss uniqueness of ground state in the HP model in section 4. Finally, we close with some conclusions in the last section.

2. The algorithm

We present here an algorithm that allowed us to exactly enumerate the $4 \times 4 \times 4$ on a PC within a few hours. At a high level the algorithm looks as follows: first it exactly enumerates all $2 \times 2 \times 2$ cubes with bonds sticking out of the cube on three of its faces. Then, two $2 \times 2 \times 2$ cubes are combined into $4 \times 2 \times 2$ cuboids, which themselves are then combined, and so forth until the $4 \times 4 \times 4$ cube is formed.

The advantage of this algorithm compared to more traditional approaches is an exponential speed-up because a particular surface configuration of a cuboid can contain many possible internal configurations. Thus, combining two cuboids that connect through their common surface is done for many configurations at once.

2.1. Creation of the first cube

The $2 \times 2 \times 2$ cubes are the smallest building blocks in our algorithm. We have to generate these building blocks first before we can combine them into bigger cuboids. There is a very limited number of $2 \times 2 \times 2$ elementary cubes (≈ 36 K); see figure 1(*a*) for one example configuration.

Thus the algorithm to find these is not the time limiting factor in the complete computation. The construction of the first cube is done sequentially by chain piece, i.e., every chain segment inside the cube is finished before the construction of the next segment is started. Each position in the cube is numbered and the chain is constructed with the lowest label first and so forth. We describe now how these chain segments are built in a way to ensure uniqueness.

The construction of a chain piece starts at the free position with the lowest label, which we call the seed. We note that this is always possible: since eventually all positions have to be visited, at least one remaining chain segment will always go through the seed. The idea is then to grow the chain in two directions starting from the seed. To prevent double counting a distinction is made between head and tail, where the position in the forward direction has a higher label than the one in the backward direction. We grow this chain piece in every possible direction recursively, until we either let it end in a position inside the cube or it reaches one of the 12 surface points.

After finishing the head of the chain segment, we do the same with its tail part. There is one subtle difference, namely that the segment can end at the seed, provided of course that the head does not end inside the $2 \times 2 \times 2$ cube. This ensures that we also enumerate all configurations of the chain segment that end in the seed. The recursive algorithm checks at each step whether the newly proposed positions are already occupied to ensure self avoidance of the chain. This is done using a simple array look-up which costs $O(1)$ time. After the chain segment is finished a new seed is chosen until all positions are occupied.

The program keeps track of the surface configuration using a simple lookup table. The surface configuration is updated every time after finishing a chain segment. When all the inner lattice points are occupied and the last chain segment is finished, the surface configuration is inserted into a tree-like data structure that is described in section 2.3.

2.2. Definition of surface configuration

The chain segments define a specific surface configuration, which consists of the connections on the surface of the cube. For the $2 \times 2 \times 2$ cube there are 7 surface monomers and 12 possible connections linking it to other cubes, namely 4 connections in each of the $+x$, $+y$ and $+z$ directions, see figure 1(*a*). It is only necessary to know how to connect two cubes; the precise internal configuration of the chain pieces is unimportant. Configurations with the same surface configuration can be processed simultaneously, thereby saving an exponential amount of time.

For a surface point S_i we assign one of the following possible states:

- UNUSED, no connection to the outside,
- S_j, an internal connection to surface point S_j,
- END, chain segment ends inside this cube.

The process of the algorithm first combines two $2 \times 2 \times 2$ cubes into $2 \times 2 \times 4$ cuboids (see figure 1(*b*)), then two of those into $2 \times 4 \times 4$ cuboids, and then finally two of those into $4 \times 4 \times 4$ cubes. At each stage we track the number of configurations for each surface configuration. To create the larger objects, we systematically go through all possible combinations of surface configurations. Whenever the combination is valid we add the product of the multiplicities of the smaller surface configurations to the multiplicity of the newly created larger surface configuration.

Two cubes cannot be combined if their total number of chain ends exceeds two, because obviously a valid Hamiltonian walk only has two chain ends. Two surface configurations are also incompatible if the combination creates a loop. A loop is created for example if both cubes have a connection from surface points j to i. However, this is not the only possibility and it

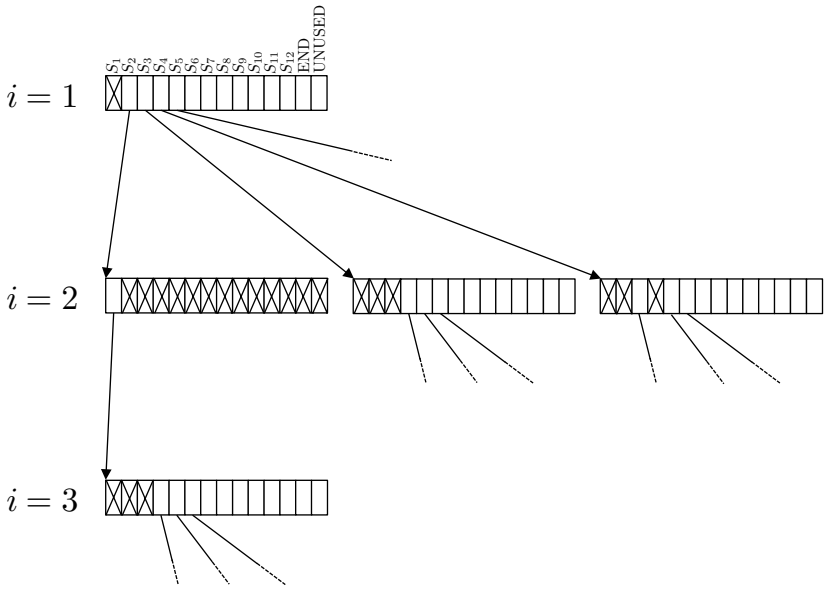

Figure 2. Tree in which surface configurations are stored. Each level i of the tree corresponds to the connection from surface point i. This information is stored in array form, where the index denotes where the link from S_i goes to. For example, if we follow the pointer S_2 from level $i = 1$, it means there is a link between S_1 and S_2 for that particular set of surface configurations. Crossed out elements in the array in the picture mean that these elements are never used (and are null pointers in the array).

can be harder to detect more complicated loops efficiently. For example, if one combines two surfaces A and B, with connections $A_1 \to A_2, A_3 \to A_4$ for A and $B_2 \to B_3, B_1 \to B_4$ for B a loop is created. We explain how to deal with this issue in more detail in section 2.5. If we are not at the last step, then two ends cannot be connected either, for the same reason that loops are not allowed. The last constraint is that if one surface point facing the other cuboid has a connection, then the same surface point of the other configuration has to have a connection as well to prevent a discontinuous self-avoiding walk ('dangling chains').

These constraints only apply to surface points on the inner surface, which is defined as the surface between the cubes to be combined. For the combination of two $2 \times 2 \times 2$ cubes, the inner surface consists of four points for each cube, while the outer surface contains eight points, see figure 1(*b*).

Then, if all these constraints are satisfied, the new surface has to be computed. This is done progressively: intermediate results are used to more quickly compute the new surface. At the first step of the iteration, this surface is empty, but at each step it is updated according to the partial configuration considered.

2.3. Data structure

As noted before, we store each cuboid only considering its surface and the number of cubes with that particular surface. The cuboid is stored in a tree data structure, shown in figure 2. The surface points are ordered in a way such that the inner surface points have a lower index than the outer surface points. Given that there are p inner surface points and q outer surface points, we number them sequentially: $S_1 \ldots S_p, S_{p+1} \ldots S_{p+q}$. For the data structure the inner and outer surface points are treated the same, but in the computation of the combination step

they are treated differently, which will be explained in more detail later on. A surface element is defined using an array S of length $p + q$. S_i has various meanings depending on its value. Specifically

- S_j: surface point i connects to surface point j,
- END: surface point i ends in the box,
- UNUSED: surface point i is unconnected.

Since every combination is reflection symmetric, the same surface configurations can be used for either cube, i.e., if a surface configuration exist with a certain multiplicity, then the reflected surface has the same (but reflected) configuration with the same multiplicity. Combining two surface configurations A and B, the inner point A_1 is opposite to the inner point B_1.

The first level of the tree consist of an array of $p + q + 2$ elements, where the index gives the value of S_1. The value of each element in this array points to memory addresses of other similar arrays, that determine the value of S_2. This proceeds recursively until finally we are at depth $p + q$ in the tree. Then at the leaves of the tree there are no pointers to other arrays, but instead the value denotes the number of cubes that have that particular surface configuration. As not every surface configuration is possible, the size of this tree is smaller than its maximum possible size. For example, if surface point i connects to surface point j, then obviously surface point j also connects to point i and thus cannot connect to any other point.

Inserting a particular surface configuration C is straightforward and computationally fast. In the case of the $2 \times 2 \times 2$ cube the multiplicity to be added, V_C, is 1, since we add them one by one. In later steps, the configuration C is the result of a combination of two surface configurations A and B. The multiplicity to be added is then $V_C = Z_A \cdot Z_B$, where Z_A is the number of internal configurations that have the same surface configuration A.

The combination algorithm starts at level $i = 1$ and goes recursively through the array following (not null) pointers deeper into the tree. This is done for two pointers simultaneously, representing surface configurations A and B. Both pointers always move simultaneously through the tree and are thus always at the same depth in the tree. If the pointers arrive at the deepest level (i.e. at two leaves in the tree), we add the multiplicity V_C to the counter of the surface configuration C, which we found during this procedure. If at some point during the insertion of the new surface configuration the pointer in the new tree is invalid (null pointer), it means that the surface configuration we found does not exist yet, and we newly add it to the tree. In this case we allocate a new array and have the pointer address the empty array. We keep adding arrays at each depth until we are at level $p + q$, and instead of allocating another array, we set the value of this array element to V_C.

The tree-like data structure has the beneficial property that searching for a particular configuration is fast, namely $O(p + q)$. At every depth the array element can be found in $O(1)$, and there is $O(p + q)$ levels of depth. Creating a fully new surface configuration in the tree is slower at $O((p + q)^2)$ operations, but the number of insertions is expected to be much lower than the number of look-ups, and the average number of operations to insert a new configuration is also expected to be smaller than the worst case scenario, because especially the parts of the tree with low i are heavily reused. The reason is that the number of surface configurations is exponentially smaller than the number of configurations.

2.4. The intermediate steps

There are two intermediate steps in the algorithm, one which combines the $2 \times 2 \times 2$ cubes into $4 \times 2 \times 2$ cuboids and one that combines the latter cuboids into $4 \times 4 \times 2$ cuboids. In

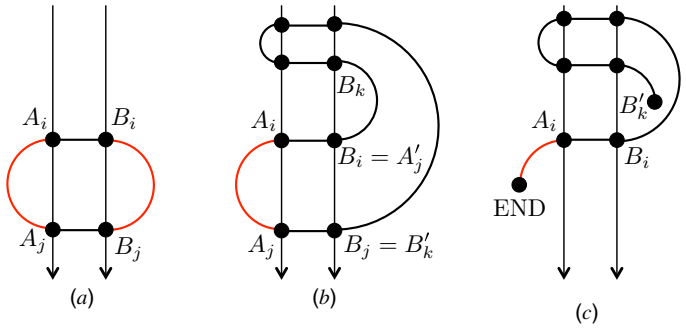

Figure 3. Examples for forbidden configurations created through the new connections $A_i \to A_j$ and $B_i \to B_k$: (*a*) a small loop $A_i \to A_j \to B_j \to B_i \to A_i$, (*b*) a larger loop with $B_j = B'_k$ and (*c*) connecting the two chain ends.

these steps, we process the tree data structure of the smaller cuboid, to create the tree of the larger cuboid. This is done in an exhaustive manner, i.e. all possible combinations of surface configurations are attempted, excepting those that create dangling ends, loops or too many ends. By traversing the tree using two separate pointers, each for one of the cuboids to be combined, a new combined surface configuration is found and put in a new tree. Since most of the surface configurations are not expected to fit, the internal surface points are processed first, and the program backtracks if the configuration has become invalid.

The inner surface contacts are the hardest to deal with, because the outer contacts cannot create dangling ends or loops or connect two ends. The only active constraint for the outer surface points concerns limiting the number of ends inside the combined cube to a maximum of 2.

2.5. Inner contacts

During each step we keep track of the current contact map of both the inner and outer surface of the newly combined cube. For example, after $A_1 \to A_2$ and $B_1 \to B_3$ we have the contact map $A_2 \to B_3$. Using this current contact map we track possible loops. Say that given a certain contact map, we have to add a connection $A_i \to A_j$ and $B_i \to B_k$ at depth i. Note that for the connection in configuration A we do not have to modify the contact map at all if $i > j$, because this means that this connection was already added at depth j. The same holds for the connection in configuration B.

On the other hand, if $j > i$ we first check whether the surface point opposite to A_j, B_j, is already connected. If this is true, then we find in our current contact map, the surface point that it connects to, which we call A'_j. If that point is B_i or B'_k (see figure 3(*a*) and (*b*)), then we know that we have created a loop in the new configuration, and we discard it. If both B'_k and A'_j are equal to END, then the configuration is discarded as well, see figure 3(*c*).

If either j or k is on the outer surface or has the value END, there is no possibility of creating a new loop. This follows from the observation that the loop is created from both sides at the same time.

2.6. The final combination

The final step is only slightly different from the intermediate steps. The difference is that in this case the two ends do have to be connected. To this effect, we set a flag if two ends are connected. This has to be the last step (in this part of the trace) in which anything is newly connected, otherwise the program backtracks.

One of the problems that still remained was that the amount of memory used was rather high. To resolve this, we split the problem up into several smaller problems. This is done in the final combination step, where we do only a specific set of combinations. As noted earlier, connections have to be matched on either side of the cuboids, to prevent dangling chains. We define the connection configuration as a simplified surface configuration, where for each surface point a '1' means any type of connection, and 0 means no connection (UNUSED). Surface configurations with a different connection configuration cannot possibly fit, because the combination will create dangling chains. Since there are 16 possible connections in the last steps, we can split the problem into $2^{16} = 65536$ smaller problems. In our case, it was not necessary to split it into so many parts, so we split the problem in 32 parts, depending on the configuration of the first 5 sites. However, the configuration '11111' (all connections made) still took too much memory, so we split that part again in 32 parts to remain within our memory budget, which we set at 8 GB. A further advantage of this is that we could easily parallelize it.

3. Rosenbluth methods

3.1. Octo cube method

In addition to an exact enumeration algorithm, the idea of connecting small blocks into larger ones is also at the basis of a new Monte Carlo method, which is a variant of the Rosenbluth method [13]. Instead of doubling the volume of the cube each time, the cube is now filled one by one with $2 \times 2 \times 2$ cubes. A Rosenbluth method computes the partition sum by iteratively building the object and by finding the probability p_i to take that particular trace at step i:

$$Z = \left\langle \prod_{i=0}^{M} 1/p_i \right\rangle, \tag{1}$$

where M is the number of steps to build the object. The value of M does not have to be constant throughout the simulation. In the case of the octo cube method $M = N/8$. Importantly, the method should guarantee ergodicity, i.e., all configurations (with non-zero contribution to the partition sum) should have a non-zero probability to be reached. In the simplest case we enumerate at each step all possibilities and choosing one of the possibilities randomly with probability $p_i = 1/w_i$, where w_i is the number of possibilities, also called the Rosenbluth weight.

The building blocks of the large cube are not all equivalent, because the smaller cube can have a different number of faces that can be either facing an already placed cube, space outside the large cube or a not yet placed smaller cube. In order to prevent recalculating the basic cubes, we exactly enumerate them and store them in memory. We do not merge symmetry equivalents into one. This means for the $4 \times 4 \times 4$ cube that there are eight different basic cubes that we enumerate. In the case of the $6 \times 6 \times 6$ cube, this number increases to 27. It does not increase further with increasing cube size. The amount of memory to store all these building blocks is sizable, approximately 1.2 GB with our data structure, which is more tailored toward lookup speed than memory efficiency.

After selecting the appropriate building block at each step, we have to connect it to the already present surface of previously inserted blocks. If there are already placed building blocks, it reduces the number of blocks that fit. Apart from selecting one of the possibilities, it is also necessary to count the number of possibilities to compute the Rosenbluth weight. Thus, it is almost mandatory to build a list of possibilities, and then randomly select from them. It is computationally intensive to recompute this list every time. Therefore, the list is stored

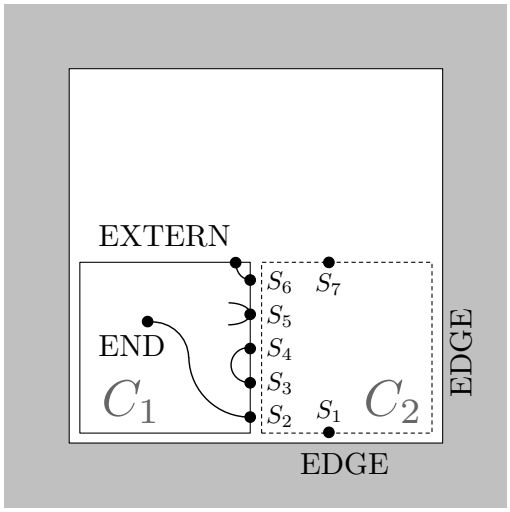

Figure 4. An example of a surrounding surface for cube C_2. The cube C_1 has already been placed in an earlier step and influences the surroundings of the new cube. Surface point S_1 is on the edge of the larger cube, and consequently it has the value EDGE. The other surface points have the following values: $S_2 \rightarrow$ END, $S_3 \rightarrow S_4$, $S_4 \rightarrow S_3$, $S_5 \rightarrow$ UNUSED, $S_6 \rightarrow$ EXTERN and $S_7 \rightarrow$ EMPTY.

in memory, and reused if the exact same local surrounding surface is encountered again. It is important to explain what we mean by the exact same local surface, because it is slightly different from the surface as explained in the context of the exact enumeration algorithm.

At each step we compute how the surface surrounding the new cube looks like. These are the possibilities for each of the 24 elements of the surface S_i:

- EDGE: element is on the edge of the larger cube,
- S_j: link to other element S_j on the same cube,
- END: link to end in any cube not yet placed,
- EXTERN: link to an element l on another cube.
- UNUSED: link cannot be used.
- EMPTY: link can be freely used.

Examples of these possibilities are shown in figure 4. It is important that the external link does not have to be specified, because otherwise the number of possible configurations grows to infinity with the size of the final cube. In the present case, even though it is an extremely large set, it will be finite independent of the total size of the cube, and we can reuse the most common ones. To save time and memory, the list of possibilities are only created on demand. Creating this list is similar to the exhaustive search for combinations in the exact enumerations algorithm. To find all of them, we try and put in all basic cubes without creating loops, dangling ends or connected ends.

After the list is created either from scratch or found in the memory, one of the update rules in the list is selected. It is the most natural to select one of the configurations with equal probability and multiply the current Rosenbluth method by the total number of possibilities at this step. However, this choice is neither the only nor the optimal one. The Rosenbluth method converges the fastest if the final weights are close to each other with a small variance. That way, the largest weights dominate the average less. The algorithm without modifications produces correct results, but most weights will be relatively small. The reason for this is that

the chance of selecting a smaller cube with one or two ends inside the cube is much larger than selecting one with no ends in the cube. Since the total number of loose ends in the cubes is two for the final configuration, the algorithm depletes the number of loose ends very quickly, i.e. the average number of loose ends does not grow linearly with cube number. Thus, most of the attempts will have one or two larger weights multiplied by the rest which is composed of lower weights. With a small probability, the algorithm selects cubes that have a low amount of loose ends (1 or 0) in the first few steps. This configuration then has a much higher weight, because the number of choices along the way is much higher (it is still able to choose from cubes with and without loose ends). Random selection leads thus to a relatively wide distribution of weights.

In order to improve on this, we select the cubes such that the probability of an end ending in cube i is roughly equal for all cubes. In the case of no loose ends already in the cube we select a cube with one end with the following probability (independent of the number of cubes with one end):

$$p_1 = 2(n_r - 1)/n_r^2, \tag{2}$$

where n_r is the number of remaining cubes: $n_r = M - i$. A cube with two ends is selected with probability:

$$p_2 = 1/n_r^2. \tag{3}$$

In the case that there is already one loose end in the cube, the probability of selecting a cube with one loose end is simply $1/n_r$. The result of this adjustment is a very significant improvement in the rate of convergence of the partition sum Z_N.

At each step, the program keeps track of which external contact connects to which position, such that at the end the new surface configuration can be created easily.

3.2. Single cube method

Instead of concatenating cubes of size $2 \times 2 \times 2$, we can also use single monomers as building blocks. An advantage compared to the octo cube method lies in the fact that the final cube does not have to be composed of an even number of monomers; the final structure can even have an arbitrary shape. The downside is that the range of possible weights of the Rosenbluth method increases, which results in a larger error bar for the same number of simulated polymers. Even though the method creates more cubes per second than the octo cube method, it takes longer to obtain the same error bars for small sizes due to the wide weight distribution. The main advantage of this method is that it needs much less memory than the octo cube method, because there is no need to store configuration lists.

We use the observation that the order in which the blocks are placed does not matter for the partition sum or any other quantity. However, it does matter for the distribution of the weights, and also for the chance of success of the creation of the full polymer. Consider for example the worst case example, in which the first half of the blocks are placed in a checkerboard fashion. This does not put any constraint on any of the blocks, because no surface is shared between them. However, the chances that each of the empty places have either one or two incoming connections is very small. To improve on this we introduce a measure for the urgency to insert a monomer. This measure is defined as the ratio of the number of connections to be made still (i.e. two minus the number of connections already set) divided by the number of remaining possible connections. The monomer with the largest ratio goes first. In case of a tie, the lowest number of possible connections goes first. If there is still a tie, the choice is arbitrary.

We also added pruning and enrichment to the Rosenbluth method, making it a pruned enriched Rosenbluth method (PERM) [14]. The advantage of the PERM method over the basic

Table 1. Exact enumeration and simulation results of the partition sum Z_N.

Size	N	Exact	Single cube	Octo cube
$3 \times 3 \times 3$	27	2480 304	$2.477(4) \times 10^6$	
$4 \times 4 \times 4$	64	27 747 833 510 015 886	$2.77(4) \times 10^{16}$	$2.777(3) \times 10^{16}$
$5 \times 5 \times 5$	125	–	$1.91(5) \times 10^{33}$	
$6 \times 6 \times 6$	216	–	$9.14(1) \times 10^{59}$	$2(?) \times 10^{59}$
$7 \times 7 \times 7$	343	–	$3.7(3) \times 10^{97}$	

Rosenbluth method is that the weight distribution is more narrow, because configurations with more weight are branched several times, creating a better estimate of the weight configurations of that type. Also less time is wasted on configurations that have a low weight compared to the average. Even though the weight distribution is significantly less wide than with the basic Rosenbluth method, it still shows the limitations of Rosenbluth based methods for cubes larger than $7 \times 7 \times 7$.

We use dynamically updated weight criteria for pruning and enrichment. The first 10^4 runs are done without any pruning or enrichment. After these first 10^4 runs, the polymer is pruned if the weight is less than 1/10th of the average weight, and enriched if the current weight is more than 10 times the average weight at that particular step. In the case of enrichment, the state of the cube configuration is copied to keep the two instances the same, but with different starting RNG seeds.

4. Results

4.1. Exact enumeration

Using our exact enumeration algorithm we find that the number of Hamiltonian walks in a $4 \times 4 \times 4$ cube equals 27 747 833 510 015 886; see also table 1.

We compare now this number to the prediction by Pande *et al* [11]. Based on the exact enumeration of Hamiltonian walks in cubic sublattices up to $3 \times 4 \times 4$ they extrapolated that there should be about 2×10^{15} walks inside a $4 \times 4 \times 4$ cube. Pande *et al* counted only walks that are not related by symmetry (rotations and reflections) but distinguished between head and tail for each walk. Our number is larger by exactly a factor $3!2^3/2 = 24$. Based on our exact enumeration we estimate that the number of walks not related by symmetry is equal to 1156159729583995. This is about 2 times smaller than the estimate in [11]. Note, however, that the authors of that paper expected their number to be an over-estimate as they argued that maximally symmetric shapes feature less conformations, an effect also observed in two dimensions [1].

4.2. Monte Carlo results

An important point to check first with PERM algorithms is the width of the weight distribution. It is well known that the width of this distribution often becomes very wide for large N, and that the results become dominated by a single or a few large weights. To check this, we have plotted this distribution in figure 5 for the largest system considered, the $7 \times 7 \times 7$ cube. Importantly, we define the weight as the total contribution of each run starting from scratch. Thus, in the process of enriching it is a sum over multiple individual weights. Comparing the shape of the distribution with the one of the $6 \times 6 \times 6$ system (which is very smooth; not shown here), we conclude that it is unlikely that we have severe problems here, since the result is not dominated by a single weight. In any case, any (severe) problems in this regard will more

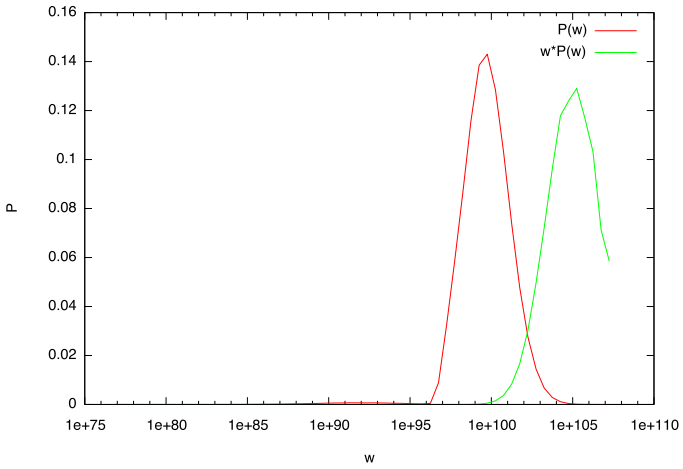

Figure 5. The weight distribution of the single cube PERM algorithm for the $7 \times 7 \times 7$ system. The red line denotes the probability P that a weight is found in the interval $[w, \sqrt{10}w)$. The green line shows $P \times w$, which gives the contribution of a each logarithmic bin to the partition sum. Ideally the two curves are close together.

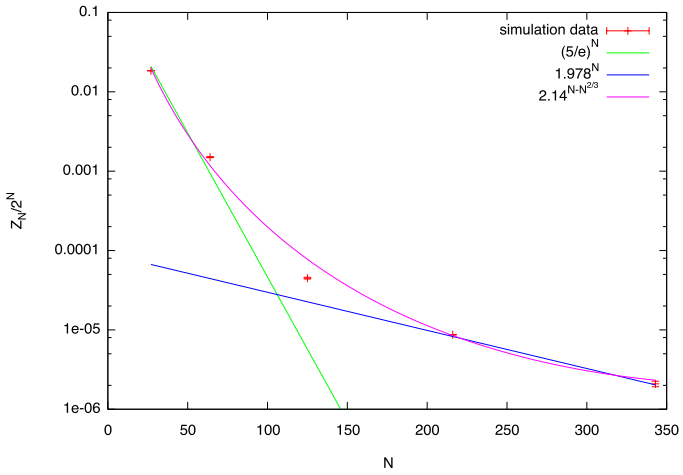

Figure 6. The partition sum Z_N (divided by 2^N) as a function of the number of monomers N. The red crosses give the numbers of Hamiltonian walks for the $3 \times 3 \times 3$ and the $4 \times 4 \times 4$ cube (exact) and for the $5 \times 5 \times 5$ up to the $7 \times 7 \times 7$ cube (approximate). The green line corresponds to the meanfield result $(5/e)^N$, as suggested in [11], while the blue line proportional to 1.978^N gives the best fit to just the large cubes $6 \times 6 \times 6$ and $7 \times 7 \times 7$. The pink curve fits reasonably well and is of the form $b\mu^{N-N^{2/3}}$ (with $b = 3$ and $\mu = 2.14$), which takes into account surface effects (see text for details).

likely have the effect of underestimation of the partition sum. In that case, our conclusions would not change much.

Based on exact enumeration of smaller cubic sublattices it was suggested in [11] that the number of Hamiltonian walks scales as $Z_N \sim \mu^N$, with only small (logarithmic) corrections. The value of μ was found to be close to $5/e$, suggesting that the meanfield calculation of Flory [15] might be applicable in the $N \to \infty$ limit. However, using the data obtained through the Monte Carlo simulations (see table 1) we find the plot depicted in figure 6, showing strong

corrections manifesting themselves by the fact that the numbers do not lie on a straight line. The discrepancy grows quickly with N, e.g. one finds that the number of walks in a $6 \times 6 \times 6$ cube is about a factor 1000 larger than estimated in [11]. Only the first two points in figure 6, the ones for the $3 \times 3 \times 3$ and the $4 \times 4 \times 4$ cubes are reasonably close to the $5/e$ line. On the other hand, the two points corresponding to $6 \times 6 \times 6$ and $7 \times 7 \times 7$ lie on a line with a larger value of μ, namely $\mu \approx 1.978$, see figure 6.

We attribute these severe corrections to surface effects. Monomers at the surface have less possible bonds than monomers on the inside of the cube. Therefore for small cubes with their large surface to volume ratio the number of possibilities is smaller than one would expect from an argument based on an infinite lattice. Instead of using the number of monomers, we suggest using the number of bonds N_b divided by the coordination number z:

$$Z_N \sim \mu^{N_b/z}. \tag{4}$$

For a cubic sublattice ($z = 6$) we find:

$$N_b/6 = N - N^{2/3}. \tag{5}$$

This expression is exact as can be easily checked by counting the number of vertices, edges and faces of the cube, and the number of monomers therein. For sufficiently large N most of the monomers belong to the interior and with $N_b \to zN$ we retrieve $Z_N \sim \mu^N$. Thus our modified expression that takes into account the finite size effects is given by

$$Z_N \approx b\mu^{N-N^{2/3}}. \tag{6}$$

In figure 6 we have plotted this expression as the green curve. It coincides reasonably well with the data, using $b = 3$ and $\mu = 2.14$. An important point to note is that the number of free parameters is exactly the same for both models, which leads us to conclude that using N_b/z instead of N gives the better answer.

The value for μ is much larger than the one predicted in [11]. This can be explained by the fact that even for the largest sublattice, $3 \times 4 \times 4$, surface effects still play the dominant role since 11 out of 12 monomers reside at the surface.

4.3. Protein folding and the HP model

The HP model was introduced by Lau and Dill [1] as a simplified model for interactions between the amino acids that constitute a polypeptide chain. In this model the amino acids are divided into two kinds, polar (P) and hydrophobic (H) ones. Hydrophilic interactions are also believed to be the main driving interactions for protein folding. One of the main open questions is how the protein finds the proper fold starting from an open (non-folded) conformation. Lattice models suggested that the probability of a degenerate ground state becomes smaller and smaller, and that the polypeptide chain finds its eventual conformation through a funnel-like energy landscape to the ground state.

Exact enumeration techniques have been performed for Hamiltonian walks on a square lattice [1], and it was found that indeed the degeneracy of random HP sequences seemed to decrease with chain length (see figures 12–15 in [1]). Enumeration of longer chains then indicated that the fraction becomes almost constant (slightly above 2%) [5]. Note that the number of configurations on a square lattice grows like 1.4^N [2] which is slower than the increase in the number of HP sequences, 2^N. Our results suggest that the situation is different for a cubic lattice where the configurations might grow like 2.14^N, i.e. faster than the number of sequences. Using a simple exact argument we show now an important consequence when using the HP model in a case when $\mu > 2$, namely that the average number of conformations in the lowest energy state is asymptotically larger than exponential in N.

Theorem 1. *Let $D(s, N, M)$ be the number of ground state conformations of an HP sequence s of length N with M P-monomers. Then averaging over all possible HP sequences $s \in S_M$, the following holds:*

$$\langle D(s, N, M)\rangle_{s\in S_M} > Z_N 2^{-N}. \tag{7}$$

where S_M denotes the set of all sequences with M P-monomers and Z_N denotes the total number of polymer configurations.

Proof. Let $T(S_M)$ be the number of polymer configurations that are in the ground state for a given HP sequence summed over all sequences $s \in S_M$. By definition one has $T(S_M)/|S_M| = \langle D(s, N, M)\rangle_{s\in S_M}$. Combining this with $|S_M| < 2^N$ we obtain:

$$\langle D(s, N, M)\rangle_{s\in S_M} > T(S_M)2^{-N}. \tag{8}$$

We now use the fact that the HP model can be mapped onto the Ising model of ferromagnetism (with a trivial shift in the energy) with the additional condition that the total magnetic moment is fixed [8]. Consider the lowest energy state(s) E_0 of the spin system. The contribution of the sequences that have this lowest energy as their ground state to $T(S_M)$ is defined as $T(S_M, E_0)$. Obviously, because it is a sum of positive elements, we have $T(S_M) > T(S_M, E_0)$. First construct the spin lattice of the energy state E_0, then put on top of this the polymer to retrieve all combinations of polymer conformations and HP sequences that give this particular ground state. We observe this way that exactly:

$$T(S_M, E_0) = Z_N. \tag{9}$$

Using this, we obtain equation (7). □

Thus, asymptotically the ground state is exponentially degenerate if $\mu > 2$. Using our simulation results, where we found $\mu \approx 2.14$, we see that in fact $\log\langle D(s, N, M)\rangle > 0$, for large enough N. Thus, we conclude that the average degeneracy of the ground state is exponential in N for large N. This, however, does not automatically imply that the probability of the ground state of an arbitrary HP sequence being degenerate goes to 1: theoretically there remains the possibility that only a small fraction of sequences lead to the huge number of E_0 ground states whereas the majority of sequences might feature unique ground states.

However, given that aligning neighboring spins reduces the energy, the ground state is even more likely to have large blobs of aligned spins. If we draw an imaginary box around one such aligned spin blob, the ground state is only unique if we cannot change the internal configuration of the blob. For a random sequence, there will be a (smooth) distribution of blob sizes. Intuitively it seems clear that for an infinitely long chains all sizes will be present (even if with exponentially small probability). Thus, for the ground state to be unique, all these blobs (of any size) have to have a unique internal configuration. This seems very unlikely. This suggests that ground states get less unique with growing chain length, even in the case of $\mu < 2$. From the theorem presented here it seems likely that for smaller μ this becomes visible at larger length, which might be the reason why it has not been detected in earlier studies [5].

5. Conclusions

We have performed exact enumeration and Monte Carlo simulations on the problem of counting the number of Hamiltonian walks on cubic sublattices. Surprisingly, we are able to discard all currently available values for μ, including $5/e$ and $6/e$. A new ansatz in which we account for surface effects leads us to believe that its most probable value lies around 2.135. Should this ansatz be incorrect, we can at least conclude that $\mu \geqslant 1.98$. We then showed that for

$\mu > 2$ the absolute ground state of the system (over all conformations and sequences) grows exponentially degenerate with chain length. We furthermore argued that—independent of the value of μ—the probability that a random HP sequence is degenerate goes to one.

The latter observations led us to question how far the HP model with random sequences is useful to understand protein folding. When going to realistic chain lengths it might become increasingly hard to find sequences with unique ground states. It seems to us that properly folding proteins are most likely not the result of a random search through all possible sequences. One could imagine instead that the combination of two viable proteins has a much higher chance of being a new viable protein than a random protein. In addition, a viable protein should not only be identified by having a unique ground state, but it should also quickly find it dynamically. One could even ask whether a viable protein could fold into its 'proper' state that is not the ground state, but dynamically stable nonetheless. Since the algorithms presented here only deal with equilibrium statistics of compact shapes, we cannot answer these questions here.

Acknowledgments

This work is part of the research programme of the Foundation for Fundamental Research on Matter (FOM), which is financially supported by the Netherlands Organisation for Scientific Research (NWO).

References

[1] Lau K F and Dill K A 1989 A lattice statistical mechanics model of the conformational and sequence spaces of proteins *Macromolecules* **22** 3986–97
[2] Chan H S and Dill K A 1989 Compact polymers *Macromolecules* **22** 4559–73
[3] Chan H S and Dill K A 1990 The effects of internal constraints on the configurations of chain molecules *J. Chem. Phys.* **92** 3118–35
[4] Shakhnovich E and Gutin A 1990 Enumeration of all compact conformation of copolymers with random sequence of links *J. Chem. Phys.* **93** 5967–71
[5] Dill K A, Bromberg S, Yue K, Fiebig K M, Yee D P, Thomas P D and Chan H S 1995 Principles of protein folding—a perspective from simple exact models *Protein Sci.* **4** 561–602
[6] Camacho C J and Thirumalai D 1993 Minimum energy compact structures of random sequences of heteropolymers *Phys. Rev. Lett.* **71** 2505–8
[7] Jacobsen J L 2007 Exact enumeration of Hamiltonian circuits, walks and chains in two and three dimensions *J. Phys. A: Math. Theor.* **40** 14667–78
[8] Shakhnovich E I and Gutin A M 1993 Engineering of stable and fast-folding sequences of model proteins *Proc. Natl Acad. Sci. USA* **90** 7195–9
[9] Socci N D and Onuchic J N 1994 Folding kinetics of proteinlike heteropolymers *J. Chem. Phys.* **101** 1519–28
[10] Sali E, Shakhnovich E and Karplus M 1994 Kinetics of protein folding: a lattice model study of the requirements of folding to the native state *J. Mol. Biol.* **235** 1614–36
[11] Pande V S, Joerg C, Grosberg A Y and Tanaka T 1994 Enumerations of the Hamiltonian walks on a cubic sublattice *J. Phys. A: Math. Gen.* **27** 6231–6
[12] Mai J, Sokolov I M and Blumen A 1997 Exact enumeration of all conformations of a heteropolymer chain in a prescribed, non-compact volume *J. Chem. Phys.* **106** 7829–33
[13] Rosenbluth M N and Rosenbluth A W 1955 Monte Carlo calculation of the average extension of the molecular chains *J. Chem. Phys.* **23** 356–9
[14] Grassberger P 1997 Pruned-enriched Rosenbluth method: simulations of θ polymers of chain length up to 1 000 000 *Phys. Rev.* E **56** 3682–93
[15] Flory P J 1953 *Principles of Polymer Chemistry* (New York: Cornell University Press)