Construction of Stable Mesh Using Self-Assembly Chains

Brodnik, Andrej and Režonja, Sandi

Abstract: ***We study the problem of unique and stable self-assembling mesh nanostructure from short chains of molecules. Although the motivation comes from coiled-coil protein molecules, our results are applicable also to other molecules like DNA. We abstract the problem to a more general problem of periodic stable and non-ambiguous embedding of n k-link chains into a mesh. The interaction between the individual chain links is defined by an interaction matrix M. In the paper we first bound the number of different interaction matrices and then design an algorithm to check whether the set of chains defined by the matrix M uniquely and stably self-assemble into a mesh structure. Our experimental results show that for $k = 2$ and $n = 4$ there is no such embedding.***

Index Terms: ***origami, chains, nanostructures, molecular structures, self-assembly, embedding, mesh***

1. Introduction

Nanostructures have potential use in numerous fields including biomedicine, chemistry, molecular electronics, tissue engineering, material science and others. Therefore a simple and cost effective production of nanustructures is in high demand. A possible way to produce the nanostructure is to use simpler molecules that automatically assemble into a desired nanostructure. Particularly appealing are simple linear chain-like structures that self-assemble. An example of such self-assembly is presented in [1] where the chain structure automatically assembles into a tetrahedron.

Originally the first choice to form chains was to use DNA molecules [2, 3, 4], however the construction of chains is not limited to DNA, but can also involve other molecules such as peptides. Recently additional interest in use of peptides was raised by the study of so called coiled-coil structures (e.g. [5, 6, 7]). Moreover by extending the set of possible chain links in [8] the coiled-coil structures became even more practical.

One can form a number of topologically different nanostructures. For example in [9] authors describe how to form besides a tetrahedron also a four-sided pyramid and triangular prism first theoretically and later also experimentally. Another example is [10], where authors study a possibility of self-assembly of cages. For this reason they describe how to assemble spheres and later hexagonal networks. In this contribution we would like to create a plane, that is a planar quadrilateral mesh. Moreover we want to use simple k-link chains which will self assemble unambiguously into such a mesh.

In the next section we give the basic definitions which are followed by the definition of an interaction matrix and estimation of the number of different matrices. The subsequent sections first present the recursive algorithm that checks if chains defined through an interaction matrix unambiguously form a stable mesh and then we give some practical results.

Manuscript received June 2019.

This work was partially supported by the Slovenian Research Agency project *N2-0053 Graph Optimisation and Big Data* and programme *P2-0359 Pervasive computing*.

The authors are with the University of Ljubljana, Faculty of Computer and Information Science, Slovenia, and the University of Primorska, Faculty of Mathematcis, Natural Sciences and Information Technologies, Slovenia (e-mail: andrej.brodnik@fri.uni-lj.si and sr3182@student.uni-lj.si).

2. Notation and Definitions

We will introduce notation and definitions through simple examples by appropriate abstraction. First, in Figure 1 is presented a part

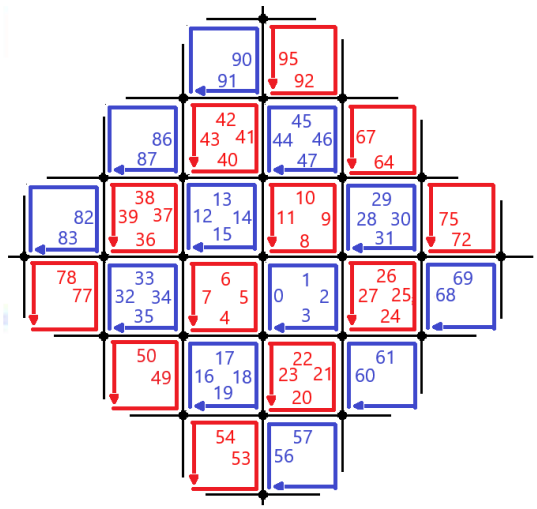

Figure 1: Quadrilateral mesh made of two chains.

of infinite quadrilateral mesh consisting of periodically repeating instances of blue and red chains. In general we consider n chains with links $v_i = v_{i,1}v_{i,2}...v_{i,k}$ $(i = 1..n)$. Besides individual links have also orientation either $v_{i,j}$ or $-v_{i,j}$. In this paper we will limit ourselves to $n = 2$ $k = 4$-link chains, although the results can be generalized to an arbitrary n and k.

Each of chains appears in several instances that are embedded in a mesh. We will later require additional properties for embedding (uniqueness and non-ambiguity), which permit us to study only periodic embeddings. On the other hand the periodic embedding permits us to consider only limited size mesh of size $2r \times 2r$.

The embedding of chains in Figure 1 is stable, because:

- every horizontal and vertical section is covered by exactly two chain links; and
- at no vertex t there exists a partition of its neighbors into two subsets that are not connected through one of the chains.

If the first property is trivial, the second one guarantees that there will appear no holes in a mesh. For example, consider the very central vertex and its four neighbours: N, E, S, and W. Take any of them and observe that it is connected to the other three through the chains. For example, the red chain connects W and S, while one instance of the blue chain connects N and W and the other S and E. Finally, the N and E are connected via both blue instances and the bottom left red instance of a chain. This connection is particularly important as it binds an otherwise broken direct connection between N and E.

3. Interaction Matrix

Individual chain links can interact or not. The interaction means that they bind and can be jointly embedded in a horizontal or vertical mesh segment. For example in on the top vertical segment in the centre of Figure 1 red link 11 and blue link 14 interact. Furthermore, the chains in Figure 1 have an orientation which is denoted with the arrows. It turns out, that in this case all links interact if they have the same (positive) orientation, which does not need to be the case always.

In general, we can define an interaction matrix M of size $2nk \times 2nk$ describing an interaction between individual chain links (value 1) or a lack of it (value 0). The factor 2 in the size is there because we have to consider both link orientations. The matrix M is symmetric.

It is left to the reader to verify that the following interaction matrix defines the chains in Figure 1:

$$
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

The rows and columns are labeled $v_{1,1}...v_{1,4}$, $v_{2,1}...v_{2,4}$, $-v_{1,1}... - v_{1,4}$, and $-v_{2,1}... - v_{2,4}$. from top to bottom and from left to right respectively.

4. Generating Interaction Matrices

Since an interaction matrix fully describes interactions between chain links it also defines how they will embed in the quadrilateral mesh. Consequently, we will have to generate all *feasible matrices* and for each one check whether it yields a valid embedding. The trivial time complexity of the process is $O(\mu(nk)f(n,k))$, where $\mu(nk)$ denotes the number of feasible interaction matrices and $f(n,k)$ time to verify whether n k-link chains properly embed in a mesh. In this section we will estimate the number of feasible matrices. A matrix is feasible if it generates chain structures that can be *stably* and *unambiguously* embedded in a quadrilateral mesh.

In the most trivial solution we could generate every possible matrix and check if it generates the required chains. However, as we will show it is easy to put additional restrictions on matrix generation to avoid disallowed chain structures. We conclude the section with a brief description of the matrix generation algorithm and estimation on the upper bound number of feasible matrices.

4.1 Necessary Properties

First observe, that the interaction matrix M is a permutation matrix because each row contains exactly one 1. Would it contain none, this link would not interact with any other link, and would there be two ones the link could interact with two other links introducing an ambiguity. Next, M is also symmetric ($M_{i,j} = M_{j,i}$), because the interaction between links is mutual and finally the interaction also remains unchanged if we invert orientation in both links $M_{i,j} = M_{j,i} = M_{\bar{i},\bar{j}} = M_{\bar{j},\bar{i}}$, where $\bar{i}$ is the index of the inverted link. It is easy to verify that $\bar{i} = (i + nk) \bmod 2nk$. Informally, $v_i = -v_{\bar{i}}$.

Furthermore, the diagonal elements have to be 0, as if $M_{i,i} = 1$ this would lead us into an inherently ambiguous embedding. An example is in Figure 2 where blue and red are instances of

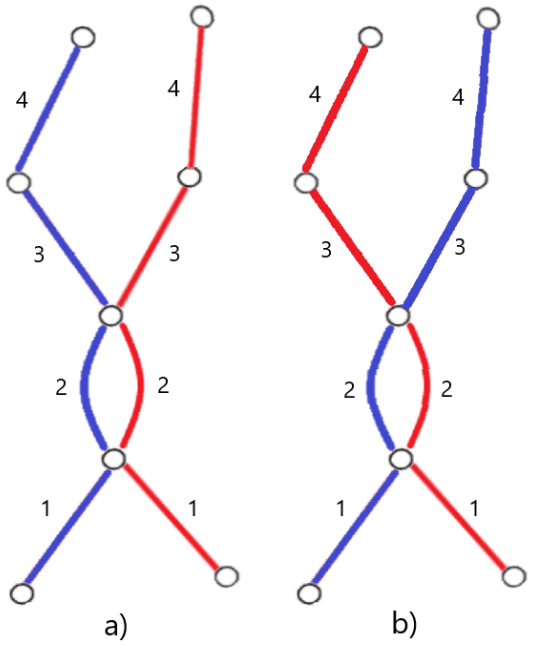

Figure 2: Since $M_{2,2} = 1$ we can generate either left or right embedding.

the same chain, with $M_{2,2} = 1$. Consequently we can generate either a left or right embedding, which leads us to ambiguity.

We state the last two properties for case $n = 2$, although they can be generalized to general n. In the previous section we defined the stability property which ensures that the embedding does not permit holes in the mesh or even breaking into pieces – i.e. if there is no interaction between links of chains v_1 and v_2, the embedding will fall apart. Finally, one can always invert the orientation of chains and appropriately change entries in M.

4.2 An Algorithm

The generation algorithm is based on the permutation matrix generation algorithm with an additional property to respect the limitations mentioned in the previous subsection. The simplified pseudo code is presented in Algorithm 1. The function initially takes as an argument a $2nk \times 2nk$ matrix filled with zeros. It then systematically generates all feasible matrices respecting limitations from subsection 4.1. Obviously, to output a single matrix it takes $\Theta(nk)$

Algorithm 1 Generation of interaction matrices.

```
1: function GENERATEMATRICES(InteractMatrix M)
2:   if each row contains 1 then
3:     if chains are connected with each other then
4:       if no matrix equivalent to M is saved then
5:         save M
6:     return
7:   i ← first row that does not contain 1
8:   for all columns j ≠ i that do not contain 1 do
9:     changes:
10:      M[i][j] = 1; M[j][i] = 1
11:      M[ī][j̄] = 1; M[j̄][ī] = 1
12:    GENERATEMATRICES(M)
13:    revert changes
```

time. However, we can show that the change of one matrix into another takes $O(1)$ amortized time. It remains to limit the number of feasible matrices based on properties described in the previous subsection.

Let the dimension of the matrix be $x \times x$. Because of requirement $M_{i,j} = M_{i,j} = M_{\bar{j},\bar{i}} = M_{\bar{j},\bar{i}}$, we can put 1 on $x-2$ positions and each time reduce the size of the matrix to $(x-4) \times (x-4)$. On the other hand, putting 1 on a position with $\bar{i} = j$ reduces the size to $(x-2) \times (x-2)$. This brings us to the following recurrent estimate on the number of matrices:

$$F(x) = \begin{cases} 1 & x = 0 \\ 1 & x = 2 \\ F(x-2)+ & \\ (x-2)F(x-4) & \text{otherwise.} \end{cases}$$

Unfortunately we could not find a closed form of the equation, but for our case, when $n = 2$ and $k = 4$ $F(16) = 5937$.

The generated matrices also include those with no interaction between the chains. For two chains there are $F^2(\frac{x}{2})$ such matrices and in our case this is 625. Finally, we exclude matrices with inverted links, which brings us to the final number of feasible matrices that is 767.

5. Embedding of Chains

In the previous sections we introduced the notion of chains and defined the possible mutual interactions of chain links through the interaction matrix M. Moreover, we presented how we generate feasible matrices, while in this section we show how we check whether chains defined through M can self-assemble in a mesh. At the end of the section we present empirical results that answer negatively for $n = 4$, $k = 2$ and a quadrilateral mesh.

5.1 Required Embedding Properties

The self-assembly of chain instances in a mesh form is equivalent to a problem of embedding chain instances into a mesh under three constraints we already mentioned. The first one is *stable embedding* (see section 2), which disallows the appearance of holes in a mesh or even breaking it into pieces. Obviously one can always in $O(1)$ per link check whether the embedding is stable.

The second constraint we need to respect is *uniqueness* of embedding, which means that an interaction matrix can not produce two different embeddings up to mirroring and rotation. This constraints comes from the requirement that embedding self-assembles in one single way.

The last requirement introduces the third constraint about *periodicity*. The periodicity comes as a natural simplification as we want our embedding to be the same anywhere in a mesh. This requirement permits us to study an embedding on a smaller finite part of the quadrilateral mesh of size $r \times r$. We name this piece of mesh a tile, using with which we can tile the whole plane. However, for technical reasons we need to extend a tile for e units on all sides to properly treat the boundaries.

5.2 Embedding and Matrix Evaluation

In the previous section we described the process of interaction matrix generation. Each matrix is then used to check whether chains it defines properly embed in a quadrilateral mesh. The process of embedding can end in one of the following situations:

Impossible: the chains can not embed in an infinite quadrilateral mesh.

Unstable: the formed embedding is not stable as described in Section 2.

Ambiguous: there are two possible embeddings of the same set of chains.

Valid: the chains properly embed in a quadrilateral mesh.

The embedding algorithm (see Algorithm 2) performs an exhaustive search by checking for

Algorithm 2 Evaluation of interaction matrices.

```
 1: function EVALUATE(IntMatrix M, ChainStruct C)
 2:   c ← C.findFreeLink() //current link to connect
    //recursion stop condition:
 3:   if c does not exist then
 4:     if !C.checkStability() then
 5:       return UNSTABLE
 6:     if no valid structure exists then
 7:       save C as valid structure
 8:       return VALID
 9:     if valid structure is equivalent to C then
10:       return VALID
11:     return AMBIGUOUS
    //branching loop and recursive call:
12:   candidates ← C.findCandidates(c, M)
13:   candidates += appropriate link from
        C.newOppositeChain(c, M)
14:   validFound ← false
15:   for all candidate in candidates do
16:     if C.connectEmbed(c, candidate, M) then
17:       rval ← EVALUATE(M, C, S)
18:       VS.revertConnectEmbed()
19:     if rval == VALID then
20:       solutionFound ← true
21:       if C.distanceToCentre(c) >= r then
22:         return VALID
23:     else if rval == AMBIGUOUS
          or rval == UNSTABLE then
24:       return rval

25:   if validFound == true then
26:     return VALID
27:   return IMPOSSIBLE
```

a given interaction matrix M what kind of embedding it permits. It is a recursive algorithm that takes as an argument an interaction matrix M and structure describing the current embedding C, extends the C for one element, and recursively calls itself. The extension is performed until the area of one $r \times r$ tile is completely embedded or further embedding is not possible. In the later case the embedding is *impossible*. In the former case the algorithm checks whether the embedding is *valid*, *unstable* or *ambiguous*. The last one occurs when the same interaction matrix M can produce two different stable embeddings.

The structure C consists of instances of chains, which links are either free (not connected to other links), connected but not embedded, or connected and embedded. In the extension step we take one unconnected link (see line 2 in Algorithm 2) at a distance of at most $r + e$ from the centre, and try to connect it with another unconnected link in C (`for` loop in line 15). Finally, the embedding is done in a lazy fashion which means that we check if connecting two links fixes the structure sufficiently that we need to embed some more of its links. Note, that in the extension step for each free link we also try to add a new chain.

There are some possible simple improvements of the algorithm that do not change its asymptotic complexity, but do improve practical execution time. For example, since we are considering a quadrilateral mesh there can be 4, 12, 20, 28, 36, 44 link pairs on distances 0, 1, 2, 3, 4, 5 respectively. Consequently, we can not add new chains in the extension step if too many free links are already in C.

To estimate an asymptotic upper bound on the running time of algorithm observe, that we are working on the area of a single tile only, indeed an extended for e. This gives us an area with a radius $R = r + e$ containing $4R^2$ links, which is also the depth of the recursion. Moreover, in each extension step the number of candidate links is bounded by $\frac{4R^2}{8} = \frac{R^2}{2}$.

In the extension step we first search for the candidate links which takes $O(R^2)$ time. On the other hand, because we deploy a lazy embedding it takes $O(1)$ amortized time per link, which is subsumed by the search time for the candidate. Finally, on one hand the extension step time depends on n and k, and on the other hand so does R. However, since we fixed them to $n = 2$ and $k = 4$ in this analysis we consider only R – the area to perform an exhaustive search.

This brings us to the final upper bound on the running time of our algorithm

$$O\left(R^2\left(\frac{R^2}{2}\right)^{4R^2}\right) .$$

5.3 Empirical Results

We ran the algorithm on all 767 feasible interaction matrices generated for $n = 2$ and $k = 4$.

	IMPOSSIBLE	UNSTABLE	AMBIGUOUS	VALID
N	466	223	78	0
$\bar{t}[ms]$	4	180	$1,1 \times 10^6$	-
$t_{max}[ms]$	10	11000	$1,9 \times 10^7$	-
σ	2	850	3.5×10^6	-

Table 1: Results for $n = 2$, $k = 4$, $r = 3$, and $e = 1$.

Table 1 gives the summary of results for the 3×3 tiles with $e = 1$. The first row reports that we found no matrix producing valid embedding. Moreover, we found that 466 matrices do not define a possible embedding, 223 define unstable embedding, and 78 define ambiguous embeddings.

Table also presents the maximum time (t_{max}) and average time ($\bar{t}$) with standard deviation (σ) to realize what kind of embedding defines interaction matrix M. The figures show that realizing that an embedding is not possible is computed in the shortest time, while the time to realize that the embedding is ambiguous takes a few orders of magnitude more time. This is somehow understandable as in this case we find two embeddings and we need to compute that they are different.

6. Conclusion

The paper presents a problem of unambiguous self-assembling of chains into a quadrilateral mesh. We transformed the problem into a problem of embedding of chains into the mesh. To define the interaction between chain links we introduced an interaction matrix. Besides being unambiguous the embedding must also be stable and periodic.

To solve the problem we designed two algorithms, the first one generates feasible interaction matrices and the second one tries to embed chains defined by the matrix into a quadrilateral mesh. The later algorithm performs a lazy exhaustive search which has inherently an exponential running time in the number of chains and their length.

We empirically evaluated algorithms on two chains each consisting of four links. Such problem admits 767 feasible interaction matrices. Our search was limited to tiles of radius 3. The search found no matrix defining a valid embedding. Moreover, the algorithm found for the interaction matrix presented in Section 3 the embedding shown in Figure 1, but we knew also of embedding in Figure 3 from the same interaction matrix. Due to exponential time complexity also the running time of the algorithm becomes unpractical for longer chains.

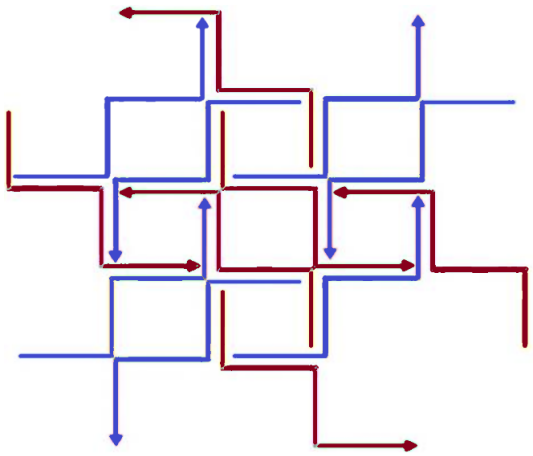

Figure 3: Known embedding of two chains with an interaction matrix from Section 3.

In the future we would like to empirically study longer chains which require bigger tiles. Consequently we will probably need to replace deterministic exponential algorithm with a faster heuristic algorithm. On the other hand a better theoretical modeling could also lead to a better understanding of the possible solution. In particular as tiling problem is a well studied problem.

Acknowledgements

The authors want to thank Prof. Roman Jerala, National Institute of Chemistry, Department of Synthetic Biology and Immunology for inspiring and motivating this research, and valuable discussions on the problem exact definition and partial results.

References

[1] Gradišar H., Božič S., Doles T., Vengust D., Hafner-Bratkovič I., Mertelj A., Webb B., Šali A., Klavžar S., and Jerala R. Design of a single-chain polypeptide tetrahedron assembled from coiled-coil segments. *Nature Chemical Biology*, 9(6):362–366, 2013.

[2] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional dna crystals. *Nature*, 394(6693):539–544, 1998.

[3] Shawn M. Douglas, Hendrik Dietz, Tim Liedl, Björn Högberg, Franziska Graf, and William M. Shih. Self-assembly of dna into nanoscale three-dimensional shapes. *Nature*, 459:414 EP –, May 2009.

[4] Paul W. K. Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

[5] Meital Reches and Ehud Gazit. Molecular self-assembly of peptide nanostructures: Mechanism of association and potential uses. *Current Nanoscience*, 200:105–111, 05 2006.

[6] Vladimir Potapov, Jenifer B. Kaplan, and Amy E. Keating. Data-driven prediction and design of bzip coiled-coil interactions. Article, PLoS Comput Biol, 2015.

[7] Greg L. Hura and John A. Tainer. Coiled coils unspring protein origami. *Nature Biotechnology*, 35:1044–1045, 2017.

[8] Andrej Brodnik, Vladan Jovicic, Marko Palangetic, and Daniel Siladi. Construction of orthogonal cc-sets. *Informatica (Slovenia)*, 43(1), 2019.

[9] Ajasja Ljubetič, Fabio Lapenta, Helena Gradišar, Igor Drobnak, Jana Aupič, Žiga Strmšek, Duško Lainšček, Iva Hafner-Bratkovič, Andreja Majerle, Nuša Krivec, Mojca Benčina, Tomaž Pisanski, Tanja Ćirković Veličković, Adam Round, José María Carazo, Roberto Melero, and Roman Jerala. Design of coiled-coil protein-origami cages that self-assemble in vitro and in vivo. *Nature Biotechnology*, 35:1094–1101, 2017.

[10] Jordan M. Fletcher, Robert L. Harniman, Frederick R. H. Barnes, Aimee L. Boyle, Andrew Collins, Judith Mantell, Thomas H. Sharp, Massimo Antognozzi, Paula J. Booth, Noah Linden, Mervyn J. Miles, Richard B. Sessions, Paul Verkade, and Derek N. Woolfson. Self-assembling cages from coiled-coil peptide modules. *Science*, 340(6132):595–599, 2013.

Andrej Brodnik is a professor of Computer Science at the University of Primorska and also lectures at the University of Ljubljana, where he also heads LUSY. The research area of dr. Brodnik are data structures and algorithms with a special attention to practical algorithms based on the theoretical modeling. He publishes in the area of combinatorial optimization and in relation to bioinformatics. Besides he is also active in Computer Science Education Research for primary and secondary schools.

Sandi Režonja received his bachelor's degree in Computer Science and mathematics from the University of Ljubljana in 2019. Currently, he is an external member of the Laboratory for Ubiquitous Systems (LUSY) at the University of Ljubljana, Faculty of Computer and Infomration Science. His research interests include optimization, heuristics, algorithms engineering and data structures.