

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta jaderná a fyzikálně inženýrská

VÝZKUMNÝ ÚKOL

Praha, 2014

Jakub Klemsa

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta jaderná a fyzikálně inženýrská

Katedra matematiky



VÝZKUMNÝ ÚKOL

Modely sebeskládajících DNA nanostruktur

Models of self-assembling DNA nanostructures

Vypracoval: Jakub Klemsa

Školitel: Ing. Štěpán Starosta, Ph.D.

Akademický rok: 2013/2014

Na toto místo přijde svázat **zadání mého výzkumného úkolu!**

V jednom z výtisků musí být **originál** zadání, v ostatních kopie.

Čestné prohlášení

Prohlašuji na tomto místě, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze dne February 12, 2014

.....
Jakub Klemsa

Poděkování

Děkuji Ing. Štěpánu Starostovi, Ph.D., za vedení mého výzkumného úkolu a za podnětné návrhy, které ho obohatily.

Jakub Klemsa

Název práce: **Modely sebeskládajících DNA nanostruktur**

Autor: Jakub Klemsa

Obor: Inženýrská informatika

Zaměření: Matematická informatika

Druh práce: Výzkumný úkol

Vedoucí práce: Ing. Štěpán Starosta, Ph.D.,

Konzultant: —

Abstrakt: Abstrakt CZ.

Klíčová slova: Klíčová.

Title: **Models of self-assembling DNA nanostructures**

Author: Jakub Klemsa

Abstract: Abstrakt EN.

Key words: Keywords.

Contents

1	Introduction to DNA computation	1
1.1	Basic DNA principles	1
1.2	Complexity, languages	2
1.2.1	O and other notations	2
1.2.2	Studied complexities	2
1.2.3	Languages	3
1.2.4	P, NP and other classes	3
1.3	Strand models	4
1.3.1	Adleman's experiment	4
1.3.2	Single-stranded molecules	5
1.3.3	Double-stranded molecules	5
1.3.4	Double crossover molecules	6
1.4	Wang-tile models	6
1.4.1	Definition	6
1.4.2	Computational power	6
2	Solutions to NP problems	9
2.1	Abstract model for DAE units	9
2.2	Graph 3-coloring	9
2.3	Graph isomorphism	13
2.4	k -clique	16
2.5	DNA computation feasibility	18
	References	19

Chapter 1

Introduction to DNA computation

1959: Feynman's visionary talk, [6]; pros: extreme parallelism, cons: reliability.

The ground-breaking work was carried out by Adleman, [2], who showed that DNA computation is practically feasible. In his experiment, Adleman used special DNA sequences for solving Hamiltonian Path Problem, one of the most typical NP-complete problems.

... Extreme parallelism! But also possibility of errors.

Work overview

Chapter 1: Intro.

Chapter 2: First of all we will describe models which exploit specific DNA structure.

Chapter 3: Abstract Tile Assembly Model, temperature, 2D vs. 3D.

Positive integers \mathbb{N} .

1.1 Basic DNA principles

DNA (deoxyribonucleic acid) is a large biomolecule carrying living organisms' genetic information. Its most common structure is well known double-helix which consists of two strands connected by hydrogen bonds. These strands are biopolymers built up by *polymerase chain reaction* (PCR) from small units – nucleotides. Each nucleotide consists of two parts: nitrogenous base and backbone molecules.

Nitrogenous bases There are 4 nucleobases in DNA (+1 in RNA): Adenine (**A**), Thymine (**T**), Cytosine (**C**), Guanine (**G**) and Uracil (**U**) in RNA instead of Thymine in DNA. These molecules are responsible for making hydrogen bonds between strands in a manner following Watson-Crick complementarity: only **A** – **T** and **C** – **G** pairs can be formed.

Backbone molecules Backbone of DNA strand is made of alternating deoxyriboses and phosphates. Phosphates only hold adjacent deoxyriboses, each deoxyribose moreover holds one nucleobase. Due to deoxyribose carbon numbering, DNA backbone has so called 5' and 3' ends, default reading order is $5' \rightarrow 3'$. DNA strands must be antiparallel so that nucleobases can connect.

1.2 Complexity, languages

1.2.1 O and other notations

Let us briefly remind O -, o -, Ω -, ω - and Θ -notations for $f, g : \mathbb{N} \rightarrow \mathbb{N}$. We will denote $(\exists n_0 \in \mathbb{N})(\forall n > n_0)$ shortly by $(\forall^* n)$ which can be read “for almost all n ”. Also $(\forall n_0 \in \mathbb{N})(\exists n > n_0)$ will be denoted by $(\exists^\infty n)$ which can be read “there exist infinitely many n ”.

Definition 1.1.

$$\begin{aligned}
g \in O(f) &\Leftrightarrow (\exists C > 0)(\forall^* n)(g(n) < Cf(n)) \\
g \in \omega^{(1)}(f) &\Leftrightarrow (\forall C > 0)(\exists^\infty n)(g(n) > Cf(n)) \\
g \in \omega^{(2)}(f) &\Leftrightarrow (\forall C > 0)(\forall^* n)(g(n) > Cf(n)) \\
g \in \Omega^{(1)}(f) &\Leftrightarrow (\exists C > 0)(\exists^\infty n)(g(n) > Cf(n)) \\
g \in \Omega^{(2)}(f) &\Leftrightarrow (\exists C > 0)(\forall^* n)(g(n) > Cf(n)) \\
g \in o(f) &\Leftrightarrow (\forall C > 0)(\forall^* n)(g(n) < Cf(n)) \\
g \in \Theta(f) &\Leftrightarrow (\exists C_1, C_2 > 0)(\forall^* n)(C_1f(n) \leq g(n) \leq C_2f(n)) \\
g \sim f &\Leftrightarrow \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 1
\end{aligned}$$

Remark 1.1. Note that there are two different definitions for omegas. $\Omega^{(1)}$ is equivalent to the original definition introduced by Hardy [8] which states

$$f \in \Omega(g) \Leftrightarrow \limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0. \quad (1.1)$$

The other, $\Omega^{(2)}$, was introduced by Knuth from good reasons described in [9]. In similar manner there are two definitions for ω .

Note that there are also some relations: the condition for $\Omega^{(1)}$ is negation of the condition for o so these sets are complementary for given function f , similarly for $\omega^{(1)}$ and O . For the second variant one can easily check that $f \in \Omega^{(2)}(g) \Leftrightarrow g \in O(f)$ and $f \in \omega^{(2)}(g) \Leftrightarrow g \in o(f)$. There is also an equivalent condition for Θ :

$$g \in \Theta(f) \Leftrightarrow g \in O(f) \wedge f \in O(g) \Leftrightarrow g \in O(f) \wedge g \in \Omega^{(2)}(f). \quad (1.2)$$

The condition for $g \sim f$ can be easily seen to be equivalent to $|f - g| \in o(g)$. In chapter 2 we will be mostly interested in this relation because it specifies the function better than Θ . For example, $2n \in \Theta(n)$ but $2n \not\sim n$.

1.2.2 Studied complexities

Definition 1.2. All the following complexities are considered as functions of input length: *Biostep complexity* will refer to the number of laboratory steps needed to handle the computation. Winfree [14] describes formally few types of such lab procedures for operations with

Boolean strings (*Separate*[*i*], *Merge*, *Detect*, *Amplify and Append*), Adleman [2] describes couple of them more practically.

Tile complexity will refer to the number of different DNA tiles needed.

Glue complexity will refer to the number of different sticky-end sequences needed (commonly referred to as *glues*). Each sequence with its Watson-Crick complement is considered as only one glue.

1.2.3 Languages

Definition 1.3. Let Σ be an nonempty finite set of *characters* which will be referred to as *alphabet*. Define set of *words* over alphabet Σ as $\Sigma^* = \bigcup_{n \in \mathbb{N}_0} \Sigma^n$ and set of nonempty words as $\Sigma^+ = \bigcup_{n \in \mathbb{N}} \Sigma^n$. Empty word will be denoted by ε . *Language* \mathcal{L} over alphabet Σ is then just a set of words: $\mathcal{L} \subseteq \Sigma^*$.

What is a formal language? Regular, context-free and recursively enumerable languages.

1.2.4 P, NP and other classes

First two definitions are from [3].

Definition 1.4. $\mathcal{L} \subseteq \Sigma^*$. $\mathcal{L} \in NP \Leftrightarrow (\exists p, q \in \mathcal{P}) (\exists \text{ deterministic Turing machine } M) (\forall x \in \Sigma^*) (x \in \mathcal{L} \Leftrightarrow (\exists y \in \Sigma^{p(|x|)}) (M \text{ accepts } (x, y) \text{ in time } \leq q(|x| + |u|)))$.

Such y will be referred to as *certificate* for x (with respect to \mathcal{L} and M).

Definition 1.5. $\mathcal{L} \subseteq \Sigma^*$. $\mathcal{L} \in co-NP \Leftrightarrow (\exists p, q \in \mathcal{P}) (\exists \text{ deterministic Turing machine } M) (\forall x \in \Sigma^*) (x \in \mathcal{L} \Leftrightarrow (\forall y \in \Sigma^{p(|x|)}) (M \text{ accepts } (x, y) \text{ in time } \leq q(|x| + |u|)))$.

Definition 1.6. $DTime(f(n)) = \{\mathcal{L} \text{ language} \mid (\exists \text{ deterministic Turing machine } M) (x \in \mathcal{L} \Leftrightarrow M \text{ accepts } x \text{ in time } \leq f(|x|))\}$.

Other classes are defined in a very similar manner (time \leftrightarrow space, deterministic \leftrightarrow non-deterministic).

Now we can easily define well-known classes P and NP .

Definition 1.7. $P = \bigcup_{k \in \mathbb{N}} DTime(n^k)$ and $NP = \bigcup_{k \in \mathbb{N}} NTime(n^k)$.

Biostep complexity, tile complexity, glue complexity.

P, NP, co-NP, PP, #P, PSpace. Enough? Maybe also polynomial hierarchy: $\Sigma_k P$ and $\Pi_k P$ languages (alternating Turing machine with bounded alternation, [10]).

Regular languages, context-free languages, recursively enumerable languages.

HPP: Adleman uses $O(n)$ biosteps, Winfree one. SAT: Lipton's contribution using m biosteps ($m = \#clauses$), [12], Lipton's set of speedup problems, [11]. Lipton 95 describes basic DNA operations. Energy efficiency (Adleman). NP definition?

1.3 Strand models

Quick overview of considered structures. Winfree's overview (pg 29 – considered molecules, pg 36 – sizes of DAE and a better picture, pg 37 – comparison of DAO/DAE in a lattice, explanation pg 43).

Seeman, Fu and their DAO/DAE in [7], is the picture of DAO strange?

1.3.1 Adleman's experiment

Adleman showed in his ground-breaking work, [2], that DNA molecules are really capable of computation. He exploited that huge parallelism possible in DNA computation for one of the most fundamental NP-complete problems – the Hamiltonian Path Problem (HPP) in directed graph with designated vertices v_{begin} and v_{end} .

Let us remind this type of HPP. Given a directed graph G_n with n vertices and two designated vertices v_{begin} and v_{end} , the problem is to answer whether there exists an oriented path from v_{begin} to v_{end} through the graph such that the path visits every vertex. Note that *path* cannot visit any vertex more than once from definition.

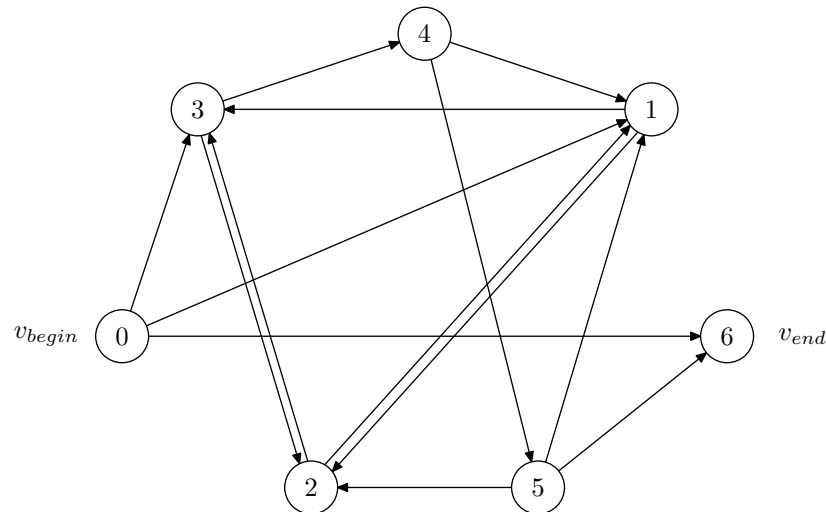


Figure 1.1: Adleman's original graph.

Adleman originally used a graph with seven vertices shown in figure 1.1. It can be seen that the path $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$ is Hamiltonian¹.

Adleman first presents this non-deterministic five-step algorithm, whose steps are then described in terms of DNA manipulations:

Step 1 Generate random paths through the graph.

Step 2 Keep only those paths that begin with v_{begin} and end with v_{end} .

Step 3 If the graph has n vertices, then keep only those paths that enter exactly n vertices.

Step 4 Keep only those paths that enter all of the vertices of the graph at least once.

¹Note that it can be re-labelled such a nice way without loss of generality.

Step 5 If any paths remain, say “Yes”; otherwise, say “No.”²

To see how DNA can compute, let us describe this example more precisely. The computation itself (meaning the inception of the final solution) is hidden in Step 1. Each vertex i is associated with a random³ 20-mer sequence of DNA, let us denote its $5' \rightarrow 3'$ orientation by O_i , its 10-mer prefix by p_i and its 10-mer suffix by q_i . Each edge $i \rightarrow j$ is then associated with $\overline{q_i p_j}$ sequence with reverse backbone orientation ($3' \rightarrow 5'$) where $\overline{q_i}$ stands for Watson-Crick complementary word. There is an exception for $i = \text{begin}$ and $j = \text{end}$: instead of $\overline{q_{\text{begin}} p_j}$ there is $\overline{O_{\text{begin}} p_j}$ and in a similar way for $j = \text{end}$.

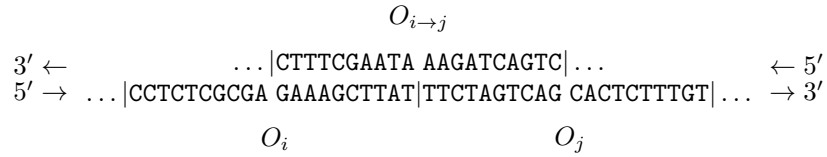


Figure 1.2: Example of assigned sequences.

It can be easily seen that all correctly bonded double-strands correspond with a valid walk through G_n . Moreover, all complete double-strands represent a valid walk from v_{begin} to v_{end} through G_n .

All the other steps are fully described in [2]. The most important thing is that the most time-demanding step is Step 4. In this step one has to purify the product of Step 3 with a biotin-avidin magnetic beads system. This process extracts consequently for every vertex i only those DNA strands which contain a substring representing vertex i . Thus this algorithm has biostep complexity $O(n)$. If we assume that one biostep takes at least tens of minutes and it should be performed repeatedly to avoid errors, we can conclude that $O(n)$ is just too much.

1.3.2 Single-stranded molecules

As we noted before, the only feasible methods are those computing in $O(1)$ biosteps.

SAT in $O(1)$ biosteps etc.

1.3.3 Double-stranded molecules

Linear strands

Equivalent to regular languages.

Dendrimer structures

Equivalent to context-free languages.

²This is the original version, I would rectify the fifth step: If any paths remain, say “Yes”; otherwise, say “*I do not know.*” That is because NP problem gives answer “Yes” iff there *exists* supporting solution. To say “No” one needs to show that *all* solutions do not satisfy. That is exactly the difference between NP and co-NP.

³We will expect those sequences to be different enough.

1.3.4 Double crossover molecules

Equivalent to recursively enumerable languages (Turing universal). Important notes in 3.2.5 Winfree – single side hybridization – how to avoid. Tricky solution of Hamiltonian Path Problem.

1.4 Wang-tile models

1.4.1 Definition

More abstract model where one handles only with “glues” on edges of Wang tiles. Define *temperature*.

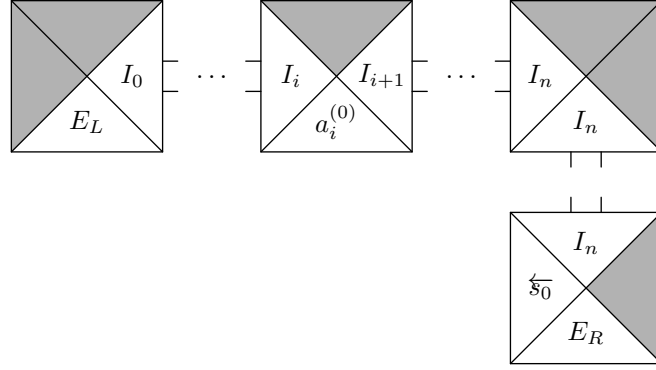
1.4.2 Computational power

Give table of Turing universality [5].

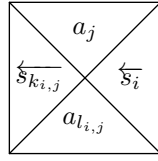
See [14]. Many other results in [5], [4], [13], [1] ...

Turing universality of 2D tiles at $T = 2$

Input tape:



Comes from right, continues left:



Comes from right, continues back to the right:

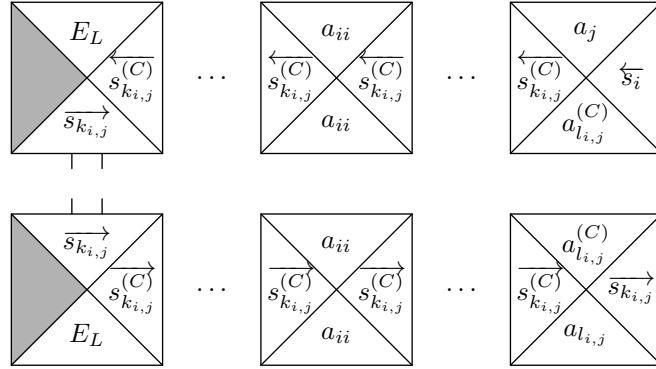
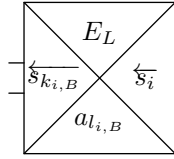
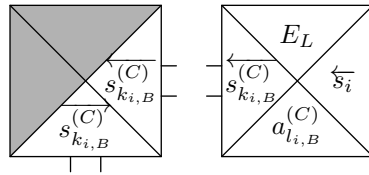


Figure 1.3: Tileset 1/2.

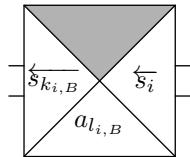
Reads EndLeft, continues left:



Reads EndLeft, continues back to the right:



Comes from right and reads Blank, continues left:



Comes from right and reads Blank, continues back to the right:

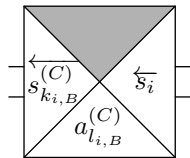


Figure 1.4: Tileset 2/2.

error-tolerant rules – Gacs and Reif, 1988

Chapter 2

Solutions to NP problems

2.1 Abstract model for DAE units

It is better to draw easier-to-understand pictures. Explanation: ... Call those DNA sequences “glues”. Binding, tile and glue complexity.

In following examples this model will be set up to act similarly like NP: $\exists y R(x, y)$. Although existence is not sure, it is very likely. Predicate R will be “enumerable in polynomial time” for $x \in L$. In this context, *enumerable in polynomial time* will mean number of bindings, not number of biosteps. This can be assumed due to Turing universality of this model in $O(1)$ biosteps – biostep complexity is not restrictive¹ and will be required to be $O(1)$ due to its lab complexity. On the other hand the binding complexity will be very important, we will be interested even in constants. This is because the less binding complexity, the less probability of error.

Describe complexities:

Binding complexity ... total number of bindings. The higher binding complexity the higher probability of error.

Tile complexity ... total number of different tile types. The higher tile complexity the more demanding it is to prepare the tiles.

Glue complexity ... total number of used glue types. The higher glue complexity the higher number of DNA letters in the sticky end sequence.

Define (slightly more correctly) binding complexity of this model as the number of bindings. Only the biggest term will be considered but even with constant.

2.2 Graph 3-coloring

Remind original Knuth’s algorithm at <http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/oetsen.htm>! And prove that everything goes fine! Robusticity?

First idea: Generate all bonds with colored atoms and check the entire system (haha, complexity like $O(n^4)$ because $|E| \in O(n^2)$). Second solution: Generate a reverse-order sequence of vertices and let it order in the correct order. All pairs should meet each other,

¹From Turing’s thesis, Turing machine is the most universal model.

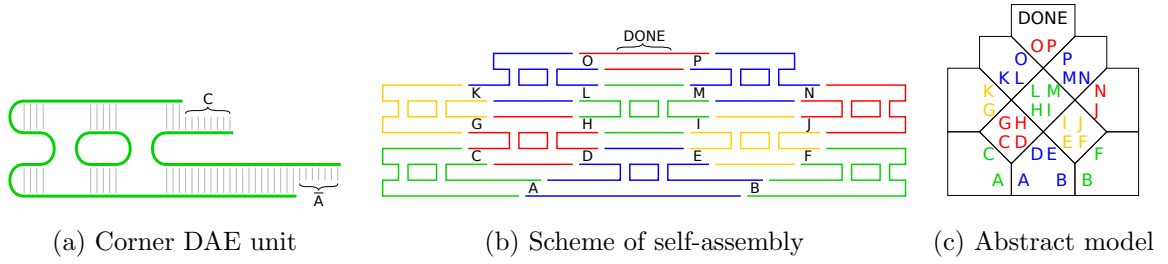


Figure 2.1: Evolution of abstract model from DAE units to tiles.

the problem to solve is whether all pairs really meet each other. After that check that the area is full like Winfree – from one side to the other. Improvement: the check can be triggered from both sides simultaneously.

Set of tiles

First of all the graph needs to have even number of vertices, thus one separated non-colored vertex has to be added if applicable. Then follow these rules which are showed in an example, see figure 2.3.

Bottom line For every pair $(2k, 2k + 1)$ there will be a bottom-type tile with non-colored numbers $(2k + 2, 2k)$ on the bottom and with all feasible² color combinations of $(2k + 1, 2k)$ on the top. From practical decoding reasons (see Winfree [14]) the sequences encoding colored numbers on the top must be physically present also wherever on the bottom DNA strand, see figure 2.2. $\frac{9n}{2}$ tile types were required.

Bottom corner tiles Both corner tiles are connected on the bottom by the highest and the lowest non-colored number, respectively, and have their special glue ($\#$ for $-\infty$ and $*$ for $+\infty$, respectively) on the top. These first two sets of tiles generate all colorings of given graph (without those omitted in previous step) with reverse order of numbers. 2 tile types were required.

Inner tiles These tiles are responsible for ordering³. There exist all color combinations for all different numbers with two important exceptions. There *do not* exist tiles with numbers of connected vertices with the same color. Thus, as soon as there appears such forbidden combination, the self assembly cannot continue and reach “DONE”. Because the numbers are generated in reverse order they must meet each other – note that they simply cannot “jump” and every number has to exchange with all the higher ones as well as with the lower ones. This implies that every forbidden combination would be revealed, thus it answers correctly if and only if the coloring is correct. The second exception are those described in the following paragraph. $9n^2$ tile types were required.

Border tiles There are two tile types on the borders, one with sharp, one with asterisk. They keep the structure growing up.

²If $(2k + 1, 2k)$ are connected, same-colored numbers are omitted.

³Principally they are the same as in Winfree [14].

Checking tiles As soon as the biggest and the smallest number reach * and #, respectively, there are two special tiles which start checking whether nothing is missing. Note that all tiles had time enough to get into correct order. In this setup checking tiles do not need to check correct order thus there can exist only two types of checking sequences “C” and “D” with all color-number combinations of middle numbers – “D” with the smaller half, “C” with the higher half. $3n$ tile types were required.

DONE tile If everything is checked and checking sequences meet each other, “DONE” tile will be connected to signalize correct solution. 1 tile type was required.

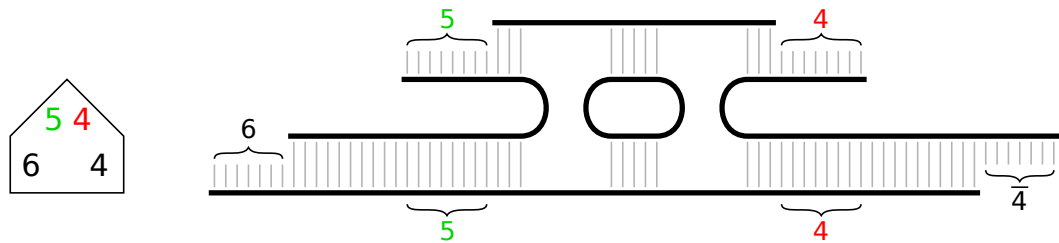


Figure 2.2: Bottom tile with desired sequences in the bottom strand.

Summed up, this DNA algorithm requires $9n^2$ tile types. Glue complexity is ...

The first idea's binding complexity was like $O(n^4)$, the second is already $O(n^2)$, the binding complexity is $1^{1/2} n^2$. The improvement decreases it to $1^{1/4} n^2$.

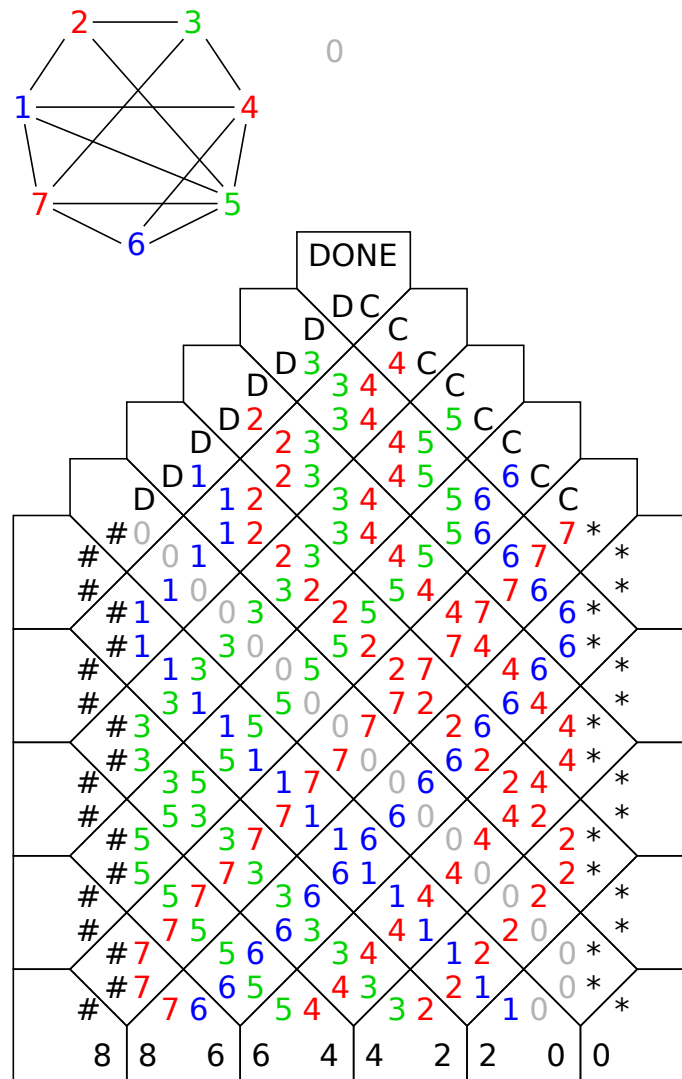


Figure 2.3: 3-color computation.

2.3 Graph isomorphism

Graph isomorphism problem clearly belongs to NP but it does not seem to be NP-hard⁴ [?]. Thus it seems that it is neither P nor NP-complete. From this reason it seems to belong to a special class and thus we will describe a DNA system which solves this very special problem.

Surprisingly it is very similar to 3-coloring if we consider n colors instead. The problem can be stated: “For a non-colored graph G and a graph H where every vertex is colored with different color, find a coloring of G with all of those n colors used exactly once (1) such that these colored graphs are isomorphic (2).” Now one has to check that:

1. every “color” was used exactly once so that it is a bijection,
2. edges and non-edges are preserved.

Set of tiles

Like before, the graph needs to have even number of vertices, thus one separated self-bijective vertex has to be added if applicable. An example is given, see figure 2.5.

Bottom line These tiles have almost exactly the same rules as in graph 3-coloring, the difference is that *all* of the same-colored combinations are omitted. $\frac{n^3}{2}$ tile types were required.

Bottom corner tiles Are exactly the same. 2 tile types were required.

Inner tiles There are three types of inner tiles:

Number-ordering tiles These are similar to previous ones, the difference is which do exist and which do not. Let us assume a tile with numbers k and l with colors a and b , respectively. Note that numbers k and l correspond with vertices in graph G and colors a and b correspond with vertices in graph H . This tile must check the isomorphism property – existence or non-existence of edge between appropriate vertices. Thus the tile exists if and only if

$$(\{k, l\} \in E(G) \wedge \{a, b\} \in E(H)) \vee (\{k, l\} \notin E(G) \wedge \{a, b\} \notin E(H)).$$

From similar reasons all pairs of vertices from graph G meet each other, thus every edge is checked so condition number 2 would be done. n^4 tile types were required.

Color extracting tiles Now we have to extract colors (forget numbers) and order them in given order so that we can check that every color is used exactly once. This process will be triggered by a special inner tile with the highest number of arbitrary color and a non-colored asterisk on the bottom. On the top it will have an asterisk of that number’s color and a non-colored asterisk. For every other number with arbitrary color there exists a tile with it and an asterisk of an arbitrary but different color on the bottom. On the top it will have two asterisks of these colors in correct order. n^3 tile types were required.

Color-ordering tiles These are similar to those with numbers. Similarly there do not exist tiles with one color. n^2 tile types were required.

⁴Clearly, if $P = NP$ it would even belong to P.

Border tiles These tiles are exactly the same like for 3-coloring.

Checking tiles As soon as there appears a combination of sharp and most-left-colored asterisk, a checking tile comes having “C” of the second color on the right top. After this initialization there are tiles with colored “C” and same-colored asterisk on the bottom and next-colored “C” on the right top. This ensures that every color was used exactly once. The last color is followed by non-colored “C”. n tile types were required.

DONE tile Finally if non-colored “C” meets non-colored asterisk, a “DONE” tile is connected signaling correct solution. 1 tile type was required.

Summed up, this DNA algorithm requires n^4 tile types and its binding complexity is $2^{1/2} n^2$. Glue complexity is ...

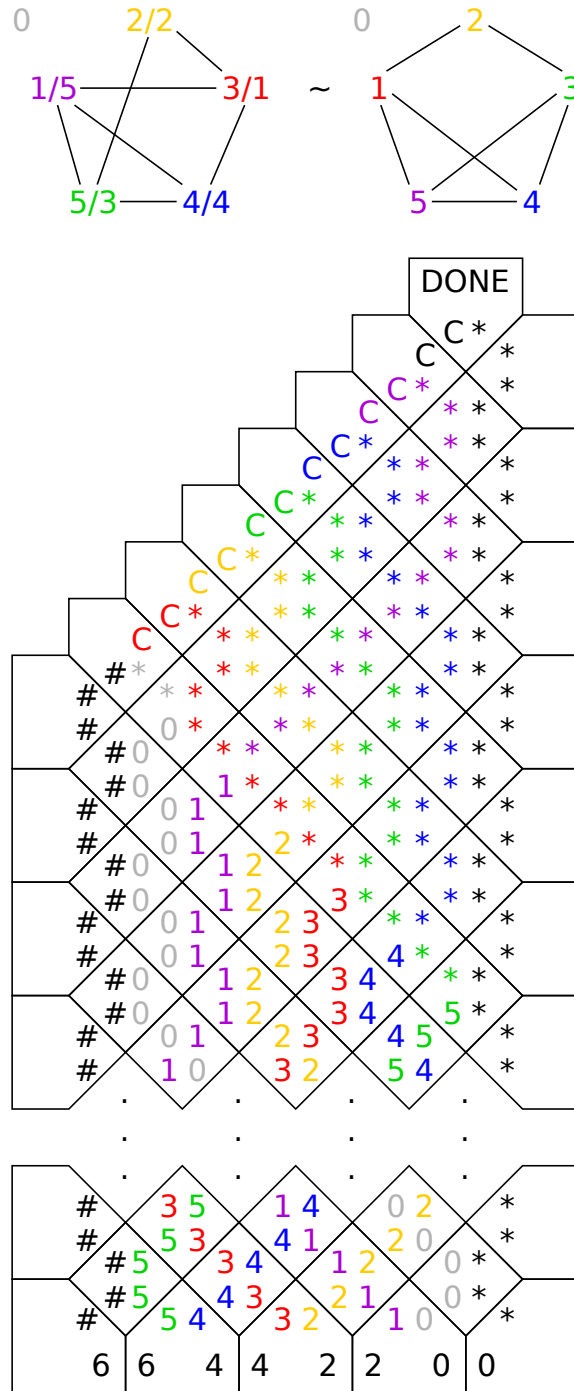


Figure 2.4: Graph isomorphism computation. Color order is defined by their wavelength.

2.4 *k*-clique

k-clique problem belongs to NP-complete problems. Note that *k*-clique problem in G is equivalent to *k*-independent set in \overline{G} and that is equivalent to $n - k$ -vertex cover of \overline{G} so we can assume $k \leq \frac{n}{2}$. Like before we add an unchecked vertex if *k* is odd so we will assume *k* to be even.

Set of tiles

Bottom line For now there are tiles with $2l - 2$ and $2l$ ($0 < l \leq \frac{k}{2}$) on the bottom and with an arbitrary ordered⁵ pair of different numbers from 1 to n with $k - 2l + 2$ -th and $k - 2l + 1$ -th colors, respectively, on the top. Note that now the order of colors is given and do not forget that they should also contain those upper sequences once more on the bottom strand. $\frac{kn^2}{4}$ tile types were required.

Bottom corner tiles Are exactly the same. 2 tile types were required.

Inner tiles These tiles are now responsible for ordering by color during which they check existence of *every* edge in similar manner to previous problems. And because the first line contains them in reverse order there will meet each other. k^2n^2 tile types were required.

Border tiles These tiles are exactly the same like for 3-coloring.

Checking tiles As soon as the most left color reaches sharp and the most right color reaches asterisk, checking is triggered in similar manner to 3-coloring. kn tile types were required.

DONE tile This is exactly the same like 3-coloring. 1 tile type was required.

Summed up, this DNA algorithm requires k^2n^2 tile types. Binding complexity is $1^{1/4} \min\{k^2, (n-k)^2\}$ Glue complexity is ...

⁵Note that this restriction does not reduce the set of possible *k*-member subsets.

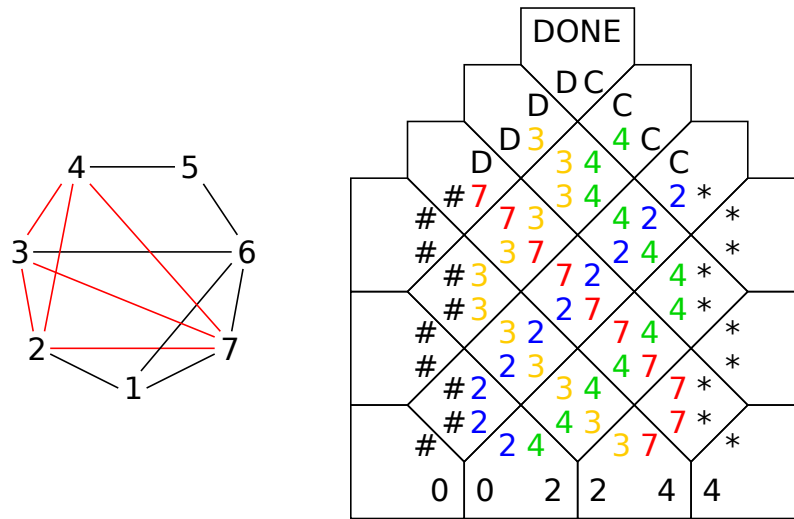


Figure 2.5: k -clique computation. Color order is defined by their wavelength.

2.5 DNA computation feasibility

Bibliography

- [1] Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 740–748. ACM, 2001.
- [2] Leonard M Adleman. Molecular computation of solutions to combinatorial problems. *Science - New York then Washington*, pages 1021–1024, 1994.
- [3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- [4] Bahar Behsaz, Ján Maňuch, and Ladislav Stacho. Turing universality of step-wise and stage assembly at temperature 1. In *DNA Computing and Molecular Programming*, pages 1–11. Springer, 2012.
- [5] Matthew Cook, Yunhui Fu, and Robert Schweller. Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 570–589. SIAM, 2011.
- [6] Richard P Feynman. There’s plenty of room at the bottom. *Engineering and Science*, 23(5):22–36, 1960.
- [7] Tsu Ju Fu and Nadrian C Seeman. Dna double-crossover molecules. *Biochemistry*, 32(13):3211–3220, 1993.
- [8] Godfrey Harold Hardy and John Edensor Littlewood. Some problems of diophantine approximation. *Acta mathematica*, 37(1):155–191, 1914.
- [9] Donald E Knuth. Big omicron and big omega and big theta. *ACM Sigact News*, 8(2):18–24, 1976.
- [10] Dexter Kozen. *Theory of computation*. Springer, 2006.
- [11] Richard J Lipton. Speeding up computations via molecular biology. *DNA Based Computers*, 27:67–74, 1996.
- [12] Richard J Lipton et al. Dna solution of hard computational problems. *Science*, 268(5210):542–545, 1995.
- [13] Paul WK Rothmund and Erik Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468. ACM, 2000.

-
- [14] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.