ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Batsyn M.V., Kalyagin V.A., Tulyakov D.N.

An efficient approach to the protein structure alignment problem

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.КЕЛДЫША
Р о с с и й с к о й а к а д е м и и н а у к

M. V. Batsyn, V. A. Kalyagin, D. N. Tulyakov

An efficient approach to the protein structure alignment problem

Москва — 2015

УДК 519.8+519.7

Бацын М. В., Калягин В. А., Туляков Д. Н.

Эффективный подход для задачи структурного сопоставления протеинов. Препринт Института прикладной математики им. М.В. Келдыша РАН, Москва, 2015

Задача Структурного Сопоставления Протеинов (ЗССП) заключается в поиске наилучшего сопоставления двух протеинов, заданных их первичными структурами. В данной работе представлен эффективный алгоритм для задачи ЗССП. Вычислительные результаты представлены для известного тестового набора Скольника из 40 протеинов и показывают, что предложенный алгоритм более эффективен, чем один из наиболее быстрых подходов для ЗССП - алгоритм ACF (Malod-Dognin et al., 2010).

Ключевые слова: сопоставление протеинов; метод ветвей и границ.

Batsyn M. V., Kalyagin V. A., Tulyakov D. N.

An efficient approach to the protein structure alignment problem. Keldysh Institute of Applied Mathematics RAS, Moscow, 2015

The Protein Structure Alignment Problem (PSAP) consists in finding the best alignment of two proteins defined by their primary structures. In this paper we present an efficient algorithm for the PSAP. The computational results are provided for the popular Skolnick test set of 40 proteins and show that the suggested algorithm is more efficient than one of the fastest PSAP solvers - the ACF algorithm by Malod-Dognin et al. (2010).

Key words: Protein alignment; branch-and-bound algorithm.

The work is supported by RSF grant 14-41-00039.

Contents

1. Introduction

For the majority of the existing proteins only the primary structure is known and the secondary structure could be predicted. But functions of a protein depend also on its tertiary and quaternary structures. However it is known that similar proteins usually have common functions (Marti-Renom et al., 2009 [17]). Similar substructures of amino acids can provide similar functions of proteins. This makes the Protein Structure Alignment Problem (PSAP), which goal is to find the most similar substructure of two proteins, an important problem in bioinformatics. This problem is also used for classification of proteins into families of similar ones (Malod-Dognin et al., 2011 [16]).

The PSAP has received much attention in recent decades. It is a challenging problem of computational biomedicine (Pardalos & Rebennack, 2010 [20]). One of the most widely used approaches to this problem is the CMO (Contact Map Overlap) method introduced by Godzik & Skolnick (1994) [11]. Holm & Sander (1993) [12] proposed a distance matrix alignment (DALI) method to measure proteins similarity and a heuristic algorithm to find the most similar substructure. Gibrat et al. (1996) [10] developed Vector Alignment Search Tool (VAST) which uses reduction to the maximum clique problem (MCP) and applies Bron & Kerbosh (1973) [5] algorithm for the exact solution. Shindyalov & Bourne (1998) [23] suggested to use a notion of aligned fragment pair for measuring similarity and developed a fast algorithm for solving the alignment problem. Integer programming approaches for solving the CMO problem belong to the papers of Carr et al. (2000) [8], Lancia et al. (2001) [14], Caprara & Lancia (2002) [6], Caprara et al. (2004) [7]. These authors reduce the original problem to the MCP (or the maximum independent set problem) and apply their branch-and-cut algorithms. Dukka et al. (2002) [9] use the reduction of the CMO problem to the MCP and solve it with the MCQ algorithm (Tomita & Seki, 2003 [25]). Konc & Janezic (2007) [13] propose MatchProt algorithm to solve the same problem and report slightly better results. Strickland et al. (2005) [24] also consider the CMO problem, present several preprocessing techniques to the resulting MCP and apply their branch-and-bound algorithm. Xie & Sahinidis (2007) [27] suggest a branch-and-bound algorithm to the CMO problem in its original formulation. Pullan (2007) [22] reduces the CMO problem to the MCP and applies the phased local search heuristic (Pullan, 2006 [21]) to obtain high-quality solutions in small time. Andonov et al. (2011) [1] propose an efficient branch-and-bound algorithm for the CMO problem with bounds obtained by a novel Lagrangian relaxation. Malod-Dognin et al. (2010) [15] introduce DAST (Distance Alignment Search Tool) method for the PSAP, reduce it to the MCP and present a fast branch-and-bound algorithm for solving it. In contrast to the CMO the DAST approach always guarantees a small value of the root mean square deviation for the resulting alignment.

In this paper we suggest an efficient approach to the PSAP. We use the DAST method to reduce the alignment problem to the MCP. The suggested algorithm IMCS (Improved MCS) is more efficient than the ACF algorithm (Alignment Clique Finder) which is one of the fastest algorithms for the PSAP to the best of our knowledge. Our approach is based on the MCS algorithm for the MCP by Tomita et al. (2010) [26]. We significantly improve the performance of the MCS algorithm in solving the PSAP. Our main contributions are the following.

- We run the ILS heuristic (Andrade et al., 2012 [2]) and obtain an initial solution close to optimal. This requires relatively small time, but considerably reduces the search tree size of the main branch-and-bound algorithm. This solution is then used for filtering large alignment graphs reducing its size.
- We suggest to calculate for each vertex an upper bound on the size of the maximum clique containing this vertex. For this purpose we run a branch-and-bound algorithm truncated to two levels. The vertices which have the upper bound not greater than the heuristic lower bound (the size of the clique found by the ILS heuristic) could be removed from the alignment graph. This preprocessing is really efficient and removes up to 70% of vertices and edges in large alignment graphs.
- We preallocate dynamic memory enough to store the candidates and their colours on every path from the search tree root to a leaf. We use this memory as a stack. This allows to avoid multiple inefficient allocation/deallocation operations with the heap.
- We use one bit for storing every element of the adjacency matrix and thus reduce its size in memory by 8 times. Since the adjacency matrix is accessed frequently this decreases the CPU cache misses and improves the performance especially for large alignment graphs.

The paper is organized as follows. In the next section we describe the PSAP problem and the DAST approach to reduce it to the MCP. The suggested algorithm is presented in Section 3. Section 4 contains computational results and completes the paper.

2. Protein structure alignment problem

To reduce the protein structure alignment problem to the maximum clique problem we follow the DAST approach. A protein is defined as a sequence of amino acids. Every amino acid has 3D-coordinates of its atoms. We use the coordinates of α-carbon atoms. Amino acid i of protein P_1 is compatible with amino acid k of protein P_2 if i and k belong to the same secondary struc-

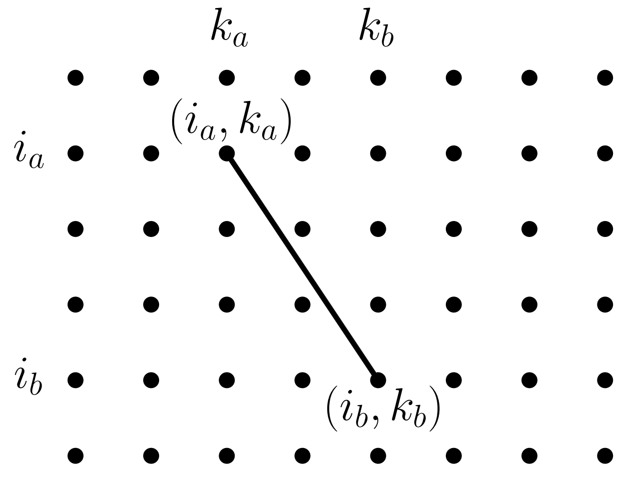

Figure 1. An edge in a protein alignment graph

ture element (SSE): α-helix, β-strand, or coil. We apply KAKSI software by Martin et al. (2005) [18] (http://migale.jouy.inra.fr/?q=kaksi) for SSE prediction. A distance between amino acids i and j is measured as a Euclidean distance: $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$. The objective of the protein structure alignment problem is to find the longest ordered sequence of amino acids $i_1, i_2, ..., i_\omega$ in protein P_1 matching an ordered sequence of amino acids $k_1, k_2, ..., k_\omega$ in protein P_2. Sequence $i_1, i_2, ..., i_\omega$ matches $k_1, k_2, ..., k_\omega$ if for any $a = 1, ..., \omega$ amino acid i_a is compatible with amino acid k_a and for any $a, b = 1, ..., \omega$ $|d_{i_a i_b} - d_{k_a k_b}| < \tau$. Here τ is a distance threshold set to 3Å. This constraint requires that any two amino acids i_a, i_b of the P_1 sequence aligned with two amino acids k_a, k_b of the matching P_2 sequence should have the same distance between them as k_a and k_b with precision τ. This also guarantees that the root square mean deviation of this alignment cannot be more than τ (Malod-Dognin et al., 2010 [15]). Note that the CMO method does not provide such guarantee and potentially can align two close amino acids with two distant ones.

The described problem is equivalent to the maximum clique problem (MCP) for the protein alignment graph. The MCP consists in finding the largest clique in the graph – the largest subgraph in which every two vertices are connected with an edge. The alignment graph has vertices $(i.k)$ where i is an amino acid of protein P_1, k is an amino acid of protein P_2, and i is compatible with k (belongs to the same predicted secondary structure element: α-helix, β-strand, or coil). Two vertices $(i_a.k_a)$ and $(i_b.k_b)$ are connected with an edge if $i_a < i_b, k_a < k_b$, and $|d_{i_a i_b} - d_{k_a k_b}| < \tau$ (see Figure 1). The objective is to find the maximum clique $(i_1.k_1), (i_2.k_2), ..., (i_\omega.k_\omega)$ in this graph.

3. Algorithm description

Our IMCS algorithm is based on ILS&MCS approach suggested in Batsyn et al. (2014) [4]. Below we provide a general scheme of our algorithm.

1. Preprocessing of the graph. The current clique is empty: $Q = \varnothing$. The best clique is the clique found by the ILS heuristic: $Q^* = \omega_{ILS}$. The candidate set S contains all the vertices: $S = V$. All vertices in S are coloured with a greedy colouring and sorted in decreasing order of their colours.
2. Branch-and-bound procedure.
   (a) Take the next candidate v. Its colour is c_v. If $|Q| + c_v \leq |Q^*|$ return from the recursion.
   (b) Add v to the current clique: $S = S \setminus \{v\}, Q = Q \cup \{v\}$
   (c) Get new candidate set $S' = S \cap N(v)$, where $N(v)$ is the set of vertex v neighbours.
   (d) If S' is empty (we are in a leaf of the seach tree) then return from recursion. (If the current clique is larger than the largest clique Q^* found so far, then update it before returning: $Q^* = Q$).
   (e) Colour vertices in S' with a greedy colouring.
   (f) Recursively perform the branch-and-bound procedure (2) for the new candidate set S'.
   (g) Perform steps (a) - (f) while the candidate set S is not empty, and then return from recursion.
3. Return the maximum clique Q^*.

We suggest to preallocate dynamic memory of the size large enough to store all candidate sets and colours of candidates on the longest path from the search tree root to its leaf. The length of such a path is equal to the size of the maximum clique and the total number of candidates can be estimated using the following proposition.

Proposition 3.1. *In a branch-and-bound algorithm the total number of candidates on any path from the search tree root to its leaf cannot be more than ωn, where ω is the size of the maximum clique and n is the number of vertices: $|V| = n$.*

Proof. According to the general scheme of a branch-and-bound algorithm in any node on level l of the search tree the current clique Q has l vertices. So the candidate set on level l has not more than $(n - l)$ vertices. The maximum number of levels is ω since it is the size of the largest clique. On level ω the size of the current clique is $|Q| = \omega$ and the number of candidates is zero since it is a leaf node of the search tree. So on every path from the root to a leaf of

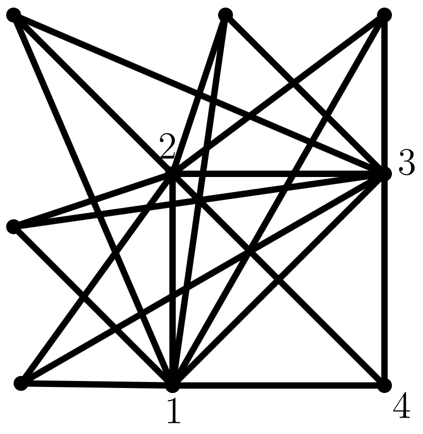

Figure 2. Total number of candidates on branch 1-2-3-4 is $(2n - \omega)(\omega - 1)/2$

the search tree the total number of candidates is not greater than:

$$(n-1) + (n-2) + ... + (n - (\omega - 1)) + 0 = \frac{(2n-\omega)(\omega-1)}{2} \leq \omega n$$

□

Note that this bound $(2n-\omega)(\omega-1)/2$ on the total number of candidates on any path of the search tree is reachable. In Figure 2 on level 1 vertex 1 has $(n-1)$ candidates, on level 2 vertices 1,2 have $(n-2)$ candidates, on level 3 vertices 1,2,3 have $(n-3)$ candidates, and finally on level 4 we have the maximum clique 1,2,3,4 and zero candidates.

Using proposition 3.1 in our implementation we preallocate dynamic memory for kn candidates, where k is the number of colours in the greedy colouring of the whole graph. This is enough since the number of colours k cannot be less than the size of the maximum clique ω (Balas, 1986 [3]).

We use the preallocated memory as a stack. In every search tree node, when we go to the next level of recursion, we push new candidates and their colours to this stack (and move the pointer to the stack top). When we return one level up the pointer to the top is automatically restored to its previous value. These operations are much faster than allocating/deallocating memory in the heap.

Another idea of improvement of the algorithm performance is a bit implementation of the adjacency matrix. In standard implementation a boolean adjacency matrix needs n^2 bytes or even $4n^2$ bytes in some implementations of the boolean data type. For example, for an alignment graph with 25000 vertices the adjacency matrix occupies 625 megabytes of memory while in bit implementation it will be only 78 megabytes. This approach reduces the CPU cache misses when the adjacency matrix is accessed. Every cache miss increases the access time by 3 - 50 times (depending on the cache level L1, L2 or L3). Since this matrix is accessed many times at every node of the search tree, when the current candidates are coloured, it is reasonable to use the bit implementation.

The next idea is to preprocess large alignment graphs (more than 20 thousand vertices and 25 million edges) and remove vertices which could not be in the maximum clique. For this purpose we first run the ILS heuristic (Andrade et al., 2012 [2]) to find a feasible MCP solution Q^* which is close to the optimal. This solution is then serves as a lower bound and every vertex, for which an upper bound on the size of the maximum clique containing this vertex is not greater than $|Q^*|$, could be removed together with its edges. For better performance a vertex is not removed immediately, but only marked to be removed after the whole procedure. To get an upper bound for a vertex we run a truncated branch-and-bound procedure down to the depth of two levels of the search tree. On the first level for every vertex v we colour the subgraph of its neighbours (candidates) $v_1, v_2, ..., v_k$ with the greedy sequential colouring. Let the number of the used colours be c. The maximum clique in the subgraph of candidates together with vertex v cannot have size greater than $c + 1$. So if $c + 1 \leq |Q^*|$ then vertex v can be removed. Otherwise, we go to the second level and consider every vertex v_i. The subgraph of candidates for v_i contains only vertices from $\{v_1, v_2, ..., v_k\}$ which are neighbours of v_i. Let it be vertices $v_i^1, v_i^2, ..., v_i^l$, and let the greedy colouring of this graph has c_i colours. The maximum clique in this graph together with vertices v and v_i cannot be larger than $c_i + 2$. If $c_i + 2 \leq |Q^*|$ we continue with the next vertex v_{i+1}. If for all $i = 1,2, ..., k$ we get $c_i + 2 \leq |Q^*|$ then vertex v can be removed. Otherwise, if for some i $c_i + 2 > |Q^*|$ we stop this procedure and conclude that v cannot be removed.

We demonstrate the suggested preprocessing on the graph shown in Figure 3a. Let the heuristic solution Q^* found for this graph by the ILS algorithm be a clique of size 3. Consider vertex 6 for example. The subgraph of its neighbours can be coloured in 3 colours as shown in Figure 3b (colours are shown in brackets). So this vertex together with its neighbours can potentially form a clique of size $1 + 3 = 4 > |Q^*| = 3$. We have to go down the search tree and consider every neighbour. For vertex 1 the candidates (common neighbours of vertex 6 and vertex 1) are vertices 2 and 5. We can colour the subgraph of these candidates in 1 colour and so the maximum clique size on this branch is not greater than $2+1 \leq |Q^*| = 3$. The same is true for the branches of vertices 2, 3, 4, and 5. So any branch for vertex 6 will not lead to a clique larger than Q^*. This means that this vertex can be removed together with its 5 edges. In the same way vertices 2, 3, and 4 can be also removed, and the resulting graph

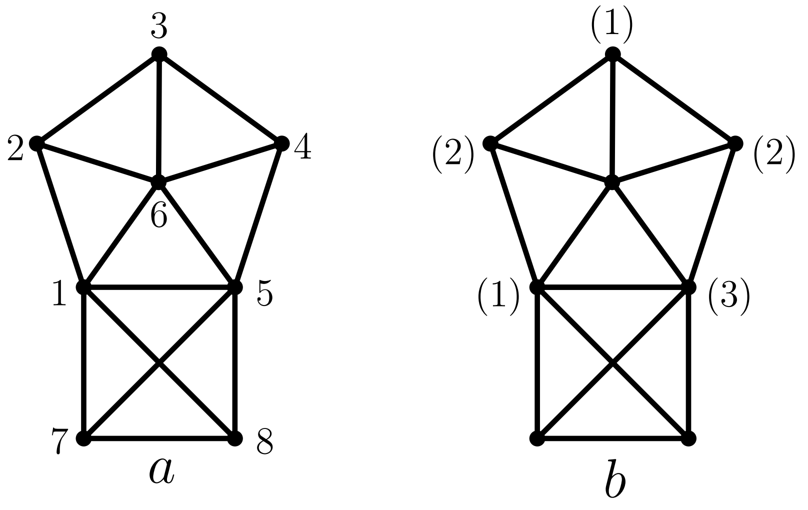

Figure 3. a. Input graph b. Colouring of vertex 6 neighbours

will have only vertices $1, 5, 7, 8$.

4. Computational results

The first computational experiments are made for 7 small alignment graphs having about 8 thousand vertices and 4 million edges. We do not run the ILS heuristic for such graphs because it is not efficient to do it for simple instances. For these graphs in average our IMCS algorithm is more than 11 times faster than the Ostergard's algorithm (2002) [19] and more than 1.25 times faster than the MCS algorithm (see table 1). These results show that the suggested IMCS algorithm is faster than the ACF algorithm by Malod-Dognin et al. (2010) [15] (one of the fastest algorithms for the PSAP to the best of our knowledge). The ACF algorithm is 9 times faster than the Ostergard's algorithm on these graphs. Unfortunately, we have not been able to implement the ACF algorithm and make a direct comparison, because it is not described in detail in Malod-Dognin et al. (2010) [15]. That is why we compare our IMCS algorithm with the state-of-art ACF algorithm indirectly via the Ostergard's algorithm.

For main computational experiments we use the same Skolnick set of 40 proteins as in Malod-Dognin et al. (2010) [15]. We compare our results with the MCS and Ostergard's algorithms on 10 large graphs (20 – 25 thousand vertices and 25 – 30 million edges) and 30 moderate graphs for which the Ostergard's algorithm works not more than 5 hours. For large graphs we first run the suggested preprocessing before the main branch-and-bound algorithm. The results are shown in table 2, where columns $|V|$ and $|E|$ contain a number of vertices and edges before preprocessing, columns "Removed $|V|$ and $|E|$"

Table 1. Running time comparison for small alignment graphs

Instances		Ostergard, sec	MCS, sec	IMCS, sec
d1k32b	d1n6ei	6.20	2.81	2.36
d1k32b	d1n6fb	2.48	2.30	2.05
d1k32b	d1n6ff	2.41	2.27	2.06
d1k32b	d1n6dd	24.01	4.07	2.77
d1n6dd	d1n6ei	9.02	2.82	2.32
d1n6dd	d1n6fb	87.02	3.66	2.67
d1n6dd	d1n6ff	62.11	3.59	2.67

contain a percent of vertices and edges removed by the preprocessing procedure, columns ω_{ILS} and ω contain the size of a clique found by the ILS heuristic and the size of the maximum clique. For smaller graphs the preprocessing does not give a notable speedup and is not performed.

Table 2. Preprocessing of large alignment graphs

Instances		$\|V\|$	$\|E\|$	Removed		ω	ω_{ILS}	ILS, sec
				$\|V\|$	$\|E\|$			
d1amk_	d1b9bA	24208	28526291	69.93%	77.43%	76	76	68
d1amk_	d1tri_	23688	27254559	26.84%	24.90%	50	50	23
d1amk_	d3ypiA	23152	26096428	28.21%	28.28%	53	53	81
d1amk_	d8timA	23838	27739393	15.45%	12.20%	42	42	1694
d1aw2A	d3ypiA	23893	27528114	25.49%	25.15%	50	50	866
d1amk_	d1aw2A	25100	30045134	23.24%	22.34%	52	52	404
d1aw2A	d1btmA	25423	30655812	8.28%	5.38%	43	40	288
d1aw2A	d1tmhA	25706	31397640	51.06%	54.91%	65	65	215
d1aw2A	d1treA	25448	30898250	14.76%	12.72%	46	46	551
d1tmhA	d1treA	25262	30270402	11.15%	7.75%	42	42	1230

All experiments are performed on Intel Core i7 machine with 2.2 GHz CPU and 8 Gb of memory. The comparison in computational time (in seconds) is presented in table 3 for large graphs and in table 4 for moderate graphs. On large instances our IMCS algorithm is more than 375 times faster than the Ostergard's algorithm and more than 5 times than the MCS in average. In comparison with the Ostergard's algorithm the speedup varies from 100 to 9450 times. Note that only 3 large instances have been solved by the Ostergard's algorithm, and 7 instances have been stopped after 4000000 seconds (45 days). For such instances the comparison of the ACF with the Ostergard is not reported in Malod-Dognin et al. (2010) [15].

Table 3. Running time comparison for large instances

Instances		$\lvert V\rvert$	$\lvert E\rvert$	ω	ω_{ILS}	ILS + Preprocess, sec	Ostergard, sec	MCS, sec	IMCS, sec
d1amk_	d1b9bA	24208	28526291	76	76	68	4000000	10545	423
d1amk_	d1tri_	23688	27254559	50	50	23	4000000	84608	5260
d1amk_	d3ypiA	23152	26096428	53	53	81	4000000	2393	1257
d1amk_	d8timA	23838	27739393	42	42	1694	1975437	55573	18060
d1aw2A	d3ypiA	23893	27528114	50	50	866	4000000	23421	23331
d1amk_	d1aw2A	25100	30045134	52	52	404	4000000	41793	11320
d1aw2A	d1btmA	25423	30655812	43	40	288	4000000	27002	9386
d1aw2A	d1tmhA	25706	31397640	65	65	215	4000000	112158	8365
d1aw2A	d1treA	25448	30898250	46	46	551	469600	56850	2846
d1tmhA	d1treA	25262	30270402	42	42	1230	3133832	73572	9068

On moderate instances our IMCS algorithm is more than 40 times faster than the Ostergard's algorithm and more than 5 times than the MCS in average. These results prove that in average our algorithm is 2 times faster than the ACF algorithm which is 20 times faster than the Ostergard's algorithm on moderate instances. The best performance in comparison with other algorithms our approach has on large instances which are hard to solve (table 3). It is due to our preprocessing of large alignment graphs. It allows us to reduce the alignment graphs and the search tree size of the branch-and-bound algorithm considerably.

Table 4. Running time comparison for moderate instances

Instances		$\|V\|$	$\|E\|$	ω	ω_{ILS}	ILS, sec	Ostergard, sec	MCS, sec	IMCS, sec
d1amk_	d2b3iA	6192	2127737	23	23	1.9	50	37	9
d1b00A	d1b9bA	11467	7159288	31	31	20.9	3197	451	111
d1b00A	d1htiA	11221	6637902	41	41	2.4	9750	52	19
d1b71A	d8timA	16717	14576531	34	23	133.6	5667	5534	1015
d1b9bA	d2pcy_	6735	2376861	23	22	17.2	122	67	29
d1b9bA	d3chy_	6340	2353878	29	29	3.4	13698	114	17
d1bawA	d1btmA	6861	2939442	25	25	3.9	2172	236	37
d1bawA	d2plt_	4692	1734838	27	27	0.6	91	37	5
d1bawA	d4tmyA	3156	719655	21	21	0.8	6.7	5.7	1.6
d1byoA	d1treA	6621	2754094	25	25	2.7	104	134	15
d1dbwA	d1ydvA	11723	6964457	35	35	0.8	3609	271	61
d1dpsA	d1nat_	3133	378554	25	21	0.1	219	0.4	0.4
d1dpsA	d1qmpA	3009	387178	25	17	0.0	748	1.8	0.3
d1dpsA	d3chy_	2240	167322	26	14	0.8	362	1.3	0.9
d1dpsA	d4tmyA	2100	144179	26	26	0.0	94	0.3	0.1
d1ier_	d1qmpB	9667	5133743	27	27	13.9	4849	450	65
d1ier_	d3chy_	10189	5544475	26	26	9.2	4632	368	77
d1kdi_	d1tri_	6417	2433204	27	27	0.0	3936	44	10
d1kdi_	d2b3iA	4427	1184704	38	38	0.1	5.5	0.8	0.7
d1nin_	d8timA	6969	2737857	23	23	5.5	667	58	18
d1ntr_	d4tmyB	5078	1706264	29	29	0.8	44	9	3
d1pla_	d1treA	7009	2917919	24	24	30.5	157	86	46
d1qmpA	d1qmpD	5997	2374341	46	46	0.9	6217	9.6	2.3
d1qmpC	d1qmpD	5892	2387556	40	40	0.8	4476	17.0	3.8
d1qmpD	d1treA	12084	7772858	37	37	36.7	8788	538	164
d1qmpD	d3chy_	6119	2523565	46	46	1.0	1291	16.0	3.1
d1rn1A	d2pcy_	4515	1331519	22	22	1.7	68	18.0	5.1
d1ydvA	d2b3iA	6541	2249656	22	22	1.0	116	50	11
d1ydvA	d4tmyA	11063	6236246	42	42	4.9	240	164	18
d3chy_	d4tmyB	5898	2324115	31	31	0.7	751	46.0	5.5

References

[1] R. Andonov, N. Malod-Dognin, and N. Yanev, *Maximum contact map overlap revisited*, Journal of Computational Biology 18 (2011), pp. 27–41.

[2] D.V. Andrade, M.G.C. Resende, and R.F. Werneck, *Fast local search for the maximum independent set problem*, Journal of Heuristics 18 (2012), pp. 525–547.

[3] E. Balas, C.S. Yu, *Finding a maximum clique in an arbitrary graph*, SIAM Journal of Computing 15(4) (1986), pp. 1054–1068.

[4] M. Batsyn, B. Goldengorin, E. Maslov, and P.M. Pardalos, *Improvements*

to mcs algorithm for the maximum clique problem, Journal of Combinatorial Optimization 27(2) (2013), pp. 397–416

[5] C. Bron and J. Kerbosch, *Algorithm 457: finding all cliques of an undirected graph*, Commun. ACM 16 (1973), pp. 575–577.

[6] A. Caprara and G. Lancia, *Structural alignment of large-size proteins via lagrangian relaxation*, in *Proceedigns of the 6th Annual International Conference on Computational Molecular Biology*, RECOMB '02, ACM, Washington D.C., USA, 2002, pp. 100–108.

[7] A. Caprara, R. Carr, S. Israil, G. Lancia, and B. Walenz, *1001 optimal pdb structure alignments: integer programming methods for finding the maximum contact map overlap*, Journal of Computational Biology 11 (2004), pp. 27–52.

[8] R. Carr, G. Lancia, and S. Istrail, *Branch-and-cut algorithms for independent set problems: integrality gap and an application to protein structure alignment*, Tech. Rep. SAND2000-2171, Sandia National Laboratories, 2000.

[9] B. Dukka, T. Akutsu, E. Tomita, T. Seki, and F. A., *Point matching under non-uniform distortions and protein side chain packing based on an efficient maximum clique algorithm*, Genome Informatics 13 (2002), pp. 143–152.

[10] J.F. Gibrat, T. Madej, and S. Bryant, *Surprising similarities in structure comparison*, Current Opinion in Structural Biology 6 (1996), pp. 377–385.

[11] A. Godzik and J. Skolnick, *Flexible algorithm for direct multiple alignment of protein structures and seequences*, CABIOS 10 (1994), pp. 587–596.

[12] L. Holm and C. Sander, *Protein structure comparison by alignment of distance matrices*, Journal of Molecular Biology 233 (1993), pp. 123–138.

[13] J. Konc and D. Janezic, *A Branch and Bound Algorithm for Matching Protein Structures*, in *Adaptive and Natural Computing Algorithms*, Lecture Notes in Computer Science, 4432, Springer, Warsaw, Poland, 2007, pp. 399–406.

[14] G. Lancia, R. Carr, B. Walenz, and S. Istrail, *101 optimal PDB structure alignment: a branch-and-cut algorithm for the maximum contact map overlap problem*, in *Proceedigns of the 5th Annual International Conference*

on Computational Biology, RECOMB '01, ACM, Montreal, QC, Canada, 2001, pp. 193–202.

[15] N. Malod-Dognin, R. Andonov, and N. Yanev, *Maximum Cliques in Protein Structure Comparison*, in *Lecture Notes in Computer Science, 6049*, SEA 2010, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 106–117.

[16] N. Malod-Dognin, M. Le Boudic-Jamin, K. Pritish, and R. Andonov, *Using Dominances for Solving the Protein Family Identification Problem*, in *Lecture Notes in Bioinformatics, 6833*, WABI 2011, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 201–212.

[17] M.A. Marti-Renom, E. Capriotti, I.N. Shindyalov, and P.E. Bourne, in *Structural Bioinformatics, 2nd Edition*, J. Gu and P.E. Bourne, eds., chap. Structure Comparison and Alignment, John Wiley & Sons, Inc, 2009, pp. 397–417.

[18] J. Martin, G. Letellier, A. Marin, J.F. Taly, de Brevern A.G., and J.F. Gibrat, *Protein secondary structure assignment revisited: a detailed analysis of different assignment methods*, BMC Structural Biology 5 (2005), pp. 1–17.

[19] P. Ostergard, *A fast algorithm for the maximum clique problem*, Discrete Applied Mathematics 120 (2002), pp. 197–207.

[20] P.M. Pardalos and S. Rebennack, *Computational Challenges with Cliques, Quasi-cliques and Clique Partitions in Graphs*, in *Lecture Notes in Computer Science, Vol. 6049*, SEA 2010, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 13–22.

[21] W. Pullan, *Phased local search for the maximum clique problem*, Journal of Combinatorial Optimization 12 (2006), pp. 303–323.

[22] W. Pullan, *Protein Structure Alignment Using Maximum Cliques and Local Search*, in *AI 2007: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, 4830, Springer, Gold Coast, Australia, 2007, pp. 776–780.

[23] I.N. Shindyalov, P.E. Bourne, *Protein structure alignment by incremental combinatorial extension (CE) of the optimal path*, Protein Engineering 11(9) (1998), pp. 739–747.

[24] D. Strickland, E. Barnes, and J. Sokol, *Optimal protein structure alignment using maximum cliques*, Operations Research 53 (2005), pp. 389–402.

[25] E. Tomita and T. Seki, *An efficient branch-and-bound algorithm for finding a maximum clique*, in *Proceedings of the 4th international conference on Discrete mathematics and theoretical computer science*, DMTCS'03, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 278–289.

[26] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki, *A simple and faster branch-and-bound algorithm for finding a maximum clique*, in *Proceedings of the 4th international conference on Algorithms and Computation*, WALCOM'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 191–203.

[27] W. Xie and N. Sahinidis, *A reduction-based exact algorithm for the contact map overlap problem*, Journal of Computational Biology 14 (2007), pp. 637–654.

Mikhail V. Batsyn *(mbatsyn@hse.ru)*
Laboratory of Algorithms and Technologies for Networks Analysis,
National Research University Higher School of Economics,
Niznhy Novgorod, Russian Federation

Valery A. Kalyagin*(vkalyagin@hse.ru)*
Laboratory of Algorithms and Technologies for Networks Analysis,
National Research University Higher School of Economics,
Niznhy Novgorod, Russian Federation

Dmitry N. Tulyakov *(dntulyakov@gmail.com)*
Keldysh Institute of Applied Mathematics,
Russian Academy of Science, Moscow, Russian Federation