Bioinformation
www.bioinformation.net

open access
Hypothesis

A data-mining approach for multiple structural alignment of proteins

Wing-Yan Siu, Nikos Mamoulis, Siu-Ming Yiu*, Ho-Leung Chan

Department of Computer Science, the University of Hong Kong, Pokfulam Road, Hong Kong, China; Siu-Ming Yiu* - E-mail: smyiu@cs.hku.hk; Phone: 852-2219-8242; Fax: 852-2559-8447; *Corresponding author

Received May 25, 2009; Revised December 19, 2009; Accepted February 10, 2010; Published February 28, 2010

Abstract:
Comparing the 3D structures of proteins is an important but computationally hard problem in bioinformatics. In this paper, we propose studying the problem when much less information or assumptions are available. We model the structural alignment of proteins as a combinatorial problem. In the problem, each protein is simply a set of points in the 3D space, without sequence order information, and the objective is to discover all large enough alignments for any subset of the input. We propose a data-mining approach for this problem. We first perform geometric hashing of the structures such that points with similar locations in the 3D space are hashed into the same bin in the hash table. The novelty is that we consider each bin as a coincidence group and mine for *frequent patterns*, which is a well-studied technique in data mining. We observe that these frequent patterns are already potentially large alignments. Then a simple heuristic is used to extend the alignments if possible. We implemented the algorithm and tested it using real protein structures. The results were compared with existing tools. They showed that the algorithm is capable of finding conserved substructures that do not preserve sequence order, especially those existing in protein interfaces. The algorithm can also identify conserved substructures of functionally similar structures within a mixture with dissimilar ones. The running time of the program was smaller or comparable to that of the existing tools.

Keywords: structural comparisons; proteins; multiple alignment

Background:
Structural comparison of proteins is an important methodology towards the understanding of protein functions. Sequencing proteins and determining the three-dimensional structures (or *structures* for short) of proteins have made rapid progress with the aid of advanced technologies such as X-ray crystallography and NMR spectroscopy. Meanwhile, testing the functions of proteins through experiments remains a time consuming and resource demanding process. Proteins which undergo mutations have been modified gradually during evolution; nevertheless, they are conserved particularly at the substructures responsible for their functioning. This suggests that structural similarity of proteins may infer their functional similarity. Automating the process of structural comparison hence assists our understanding of protein structures with respect to their functions, as well as our prediction of functions of newly discovered proteins. This paper investigates the computational problem of multiple structural alignment that helps discover important functional sites and motifs.

As the structure is more conserved than its sequence during evolution, proteins exhibiting similar functions may have conserved substructures but share no overall sequence similarity **[5,20].** Yet aligning structures in the three-dimensional (3D) space is a computationally hard problem and is NP-hard in many variations **[1].** Note that an alignment of structures involves rotations and translations to superimpose the structures and identify similar substructures. Many existing tools for structural comparisons of proteins try to get around the difficulties with different assumptions and simplifications. In this paper, we investigate the problem when much less information or assumptions are available. In particular, we study the problem with the following properties.

Sequence order independence: We do not require the sequence order information, i.e., the exact sequential arrangement of the amino acids in the proteins. The only information required is the set of 3D locations of the amino acids. This allows aligning structures without sequence order, e.g., protein interfaces which consist of amino acids from two chains which have no particular order. This is different from some existing tools which require sequence order information and assume the amino acids that form the conserved substructures are consecutive segments in the corresponding protein sequences.

Subset alignment: We design our tool to detect conserved substructures that may exist only in a certain subset of the input. This allows discovery of conserved substructures of functionally similar structures within a mixture with dissimilar ones, without prior knowledge of the structural similarity of the input. This is different from some existing tools that align *all* input structures. Note that these existing tools can perform subset alignment by trying all possible subsets. As we will show later, our algorithm can perform it in a much more efficient way.

Bottleneck metric: We use the bottleneck metric to measure the quality of an alignment, which requires that in an alignment, every pair of aligned amino acids must have a small distance. Using this metric helps reduce the chance of aligning substructures that are not structurally similar. Thus our approach is different from some existing tools that measure the similarity with relaxed metrics, e.g., the root mean square deviation (RMSD) of the aligned amino acids. We can see that some aligned amino acids may be far away even if the RMSD is small. To our knowledge, our work is the first to study the problem with all these three properties.

Related work:
Due to the importance of the protein alignment problem, it has attracted significant attention in the last three decades (see, e.g., **[7]** for a survey). Early work mostly considers pairwise alignment, e.g., DALI **[12]** and VAST **[10]**. In particular, [**5, 8]** offer pairwise sequence order independent alignments. They apply the geometric hashing technique **[23]** to allow an efficient database search for query structures. Techniques like tree-progressive approaches **[22]** and center-star **[2]** have been used to derive multiple alignments from results of pairwise ones. Recent work includes **[15].** They inherit the drawback that good pairwise alignments may not result in good multiple alignment.

A series of multiple structural alignment tools have then been developed. POSA **[24]**, MASS **[9]** and Matt **[25]** are tools that require sequence order. In particular, MASS aligns structures based on their secondary structure elements (*SSEs*). This limits its application to proteins whose SSEs information is available. MultiProt **[19]** is partially sequence order dependent, as it first finds matches for short contiguous segments of proteins and then finds multiple alignments

based on these short matches. Tools that are sequence order independent include MUSTA **[14]**, MultiBind **[20]** and MAPPIS [21]. All of them use the geometric hashing technique. They lack the support of subset alignment. [**13]** is also a sequence order independent method, which uses a graph-based technique. However it is targeted for motifs that are usually small in size, and it has been considered to be inefficient **[18].**

Methodology:
We model the alignment problem as a combinatorial problem. We define the *structure* of a protein to be a set of amino acids in the 3D space. Each amino acid is represented by its C_α, N and C atoms, and the 3D coordinates of these three atoms are given. Therefore, a structure consisting of n amino acids is represented by $3n$ coordinates. The *size* of a structure is the number of amino acids in it. A *substructure* S' of S is a subset of amino acids $S' \subseteq S$. The sequence order of the amino acids is unknown.

We want to find similar substructures among different structures with suitable transformations. In particular, a *transformation* T is a Euclidean transformation that can be applied on a structure S such that for all amino acids $s \subseteq S$, the location after transformation is $T(s) = Rs + t$, where R is a 3×3 rotation matrix, t is a 3×1 translation matrix, and the transformation is applied to all the three atoms in s.

To define similarity, let C = $\{c_1, \ldots, c_k\}$ be a set of substructures of same size l , each from a distinct structure. Let $\varepsilon \leq 0$ be a real number and T = $\{T_1, \ldots, T_k\}$ be a set of transformations. Then C is ε-*congruent with respect to* T if: (1) we can transform each substructure c_i by T_i and then order the transformed amino acids of c_i by 1, 2, … , l , and (2) for j = 1, 2, … , l , consider the j-th amino acids of all transformed substructures and let P_j be the set of these amino acids, then the C_α atoms of any two amino acids in P_j have a distance at most ε. Note that we only align the C_α atoms. For any set S' = $\{S_1, \ldots, S_k\}$ of structures, an ε-*congruent alignment* of S' is a tuple (C, T), where C =$\{c_1, \ldots, c_k\}$ is a set of substructures with $c_i \subseteq S_i$ and T = $\{T_1, \ldots, T_k\}$ is a set of transformations, such that C is ε-congruent with respect to T. The *cardinality* of the alignment is the number of structures involved, which is k in this alignment. The *size* of the alignment is the size of each substructure in C.

Finally, let S = $\{S_1, \ldots, S_m\}$ be a set of m structures. Our task is to identify the largest ε-congruent alignment for each subset S' ⊆ S. We allow a parameter *min_cardinality* ≤ 2, which is the minimum cardinality of S' for which an alignment will be considered. Similarly, the parameter *min_size* ≤ 3 is the minimum size of an alignment for S'.

Problem statement. Given a set S = $\{S_1, S_2, \ldots, S_m\}$ of structures, a distance threshold ε, a cardinality threshold *min_cardinality* and a size threshold *min_size*, find, for each subset S' ⊆ S with |S'| ≥ *min_cardinality*, the maximal size ε-congruent alignment whose size is at least *min_size*.

Algorithm SOIL:
Our new algorithm SOIL (Sequence Order Independent aLignment) finds the alignments in 3 steps.

Step 1 -- Geometric hashing:
We first apply the geometric hashing technique [23] to store the structures redundantly in many different transform-ations. Specifically, consider a structure S_k consisting of n amino acids. Let $\{S^k_1, S^k_2, \ldots, S^k_n\}$ be a set of coordinates such that S^k_i is the coordinates for the C_α atom of the i-th amino acid. We reiterate that the order is arbitrary and does not correspond to the sequence order of the protein. The C_α atom S^k_i together with the N and C atoms in the same amino acid form a *basis* for this amino acid. We define a *local reference frame (LRF)* as in [8]. We overload $C_\alpha = S^k_i$, N and C to mean the coordinates of the corresponding atoms. Then, the LRF defined by this basis is centred at S^k_i and specified by the vectors defined using equations 1 and 2 in **supplementary material** (see **[8]** for more details)

Hence, using the C_α atom S^k_i, we form an LRF and transform all other C_α atoms S^k_j, $j \neq i$, using this LRF. We hash the transformed points into a 3D grid-based hash table, where the 3D space is partitioned into hash bins having length ε in each dimension. Precisely, when a point S^k_j is transformed using the LRF of S^k_i, a 5-tuple (k, i, x, y, z) is inserted to the hash table, where the tuple (k, i) correspond to the basis S^k_i and (x, y, z) are the calculated coordinates for S^k_j after transformation. This tuple is inserted to the bin containing (x, y, z). We repeat the procedure by using each amino acid of S_k to form an LRF. We further repeat it for each structure. Note that we only hash the C_α atoms.

Step 2 - Frequent pattern mining:
Given the hash table created previously, we consider each bin as a *coincidence group*, which is simply a set containing all the bases in the bin. Our main observation is the following.

Observation:
Assume that a pair of bases $\{(k_1, i_1), (k_2, i_2)\}$ is a subset to x coincidence groups. Then, if structures S_{k1} and S_{k2} are transformed using the LRFs formed by S^{k1}_{i1} and S^{k2}_{i2}, respectively, there are at least (x+1) pairs of points locating closely with each other (specifically with distance at most $\sqrt{3}\varepsilon$, i.e., diagonal of the 3D box).

In other words, we can potentially form an alignment for S_{k1} and S_{k2} with (x+1) pairs of points aligned, where the extra one pair of points corresponds to S^{k1}_{i1} and S^{k2}_{i2}. Thus, finding a large alignment of points can be facilitated by finding a subset of bases with a large number of co-occurrences in the coincidence groups. This observation is the main novelty of our work. It allows the tool to consider all subsets simultaneously and identify the potentially good alignments directly.
We consider each basis as an *item* and each coincidence group as a set of items. Then to find the set of bases that frequently appear in the coincidence groups, it becomes the problem *frequent itemset mining*, which is well-studied in data mining. For any set of items, the number of coincidence groups that contain it is known as the *support* for this set of items. We apply an efficient frequent itemset mining algorithm, FP-growth **[11]**, to identify set of items with large support. The algorithm is exact, that is, it reports exactly all itemsets satisfying this requirement. We set the required support to 5% of *min_size*, since SOIL will extend the alignment and the final size obtained may be big enough. The current implementation will also try 3% and then 1% if FP-growth does not return results involving all structures using the previous support threshold.

Step 3 -- Alignment generation:
This step generates alignments from the frequent itemsets. Consider any set $\{S_{k1}, S_{k2}, \ldots, S_{kw}\}$ of w structures. Let $I = \{b_1, b_2, \ldots, b_w\}$ be the itemset involving these w structures and having the largest support. For i = 1, … , w, the item b_i is some basis (k_i, j_i) and we transform all amino acids of S_{ki} using the LRF formed by S^{ki}_{ji}. Then we construct a graph G so that each vertex in G corresponds to a transformed point of S_{ki} for some i and two vertices are connected by an edge if and only if they are from different structures and within ε distance from each other. G is a w-partite graph and we aim at finding a large w-partite matching. Note that finding the largest w-partite matching is NP-hard. Instead, we use a greedy heuristic that if w vertices in G form a clique and none of them has been included in the matching, we include them into the w-partite matching for G. We continue until no more vertices can be included.

For each subset of structures, we repeat the above for the K itemsets with the largest supports, where K is empirically set to be 10 times the number of input structures. The largest alignment found is then reported as the alignment for this subset of structures. SOIL then repeats the procedure for other subsets of structures which satisfy that some itemset involving them has a large support.

Results and discussion:
We implemented SOIL in C++. The FP-growth procedure was adopted from the open-source package provided by Borgelt **[6**] and modified for our particular purpose. Experiments were run on a PC with a dual core 2.66GHz CPU and 4GB memory. In the experiments, we applied the default settings as follows: the distance threshold ε is 3.0 Å, the cardinality threshold *min_cardinality* is 2, and the size threshold *min_size* is 3.
We compared SOIL with C-alpha match **[5]** (pairwise only), MultiProt [**19]** and MultiBind **[20].** C-alpha match has a web server and we ran the tests on it. We downloaded the MultiProt software and ran the tests on our machine. As MultiBind only aligns labeled pseudocenters and small datasets, and does not support subset alignment, we adopted the algorithm and re-implemented it so that it aligns the C_α atoms. For subset alignment, we let MultiBind iterate through every subset of structures, and let the first structure be a pivot, which is required by MultiBind. We denote the re-implemented software as MultiBind*.

Note that C-alpha match and MultiProt use the RMSD metric. MultiBind* uses the bottleneck metric. But it is only for the alignment between the pivot and another structure, and has no guarantee for any two non-pivot structures. When making comparison, we first show the sizes of the alignments under the bottleneck metric, i.e., without counting those aligned points that have distance greater than ε. Then we also show the sizes of the alignments under the tool's own distance metric.

Pairwise alignment:
We performed pairwise alignment on 15 pairs of protein structures. The first 10 pairs have been used as testing data, e.g., in **[19**]. The 11-th pair shares the same family in the SCOP classification **[16].** Structures in the 12-th pair both have a *4-helix bundle* structure but of different topologies. The last three pairs are protein interfaces retrieved from PRINT database **[17].**

We compared the sizes of the alignments generated by SOIL with that by C-alpha match, MultiProt and MultiBind*. The results are shown in Table 1 **(see supplementary material).** We can see that SOIL has the largest alignment for all cases, even when the other tools are measured with their own distance metrics. In particular, SOIL gave the largest alignment for all pairs of protein interfaces. This suggests that sequence order independence in alignment is important in aligning structures with non-contiguous sequences. The average running times of MultiProt, MultiBind* and SOIL were 0.211s, 1.968s and 0.235s respectively. The web server of C-alpha match returned the results within a few seconds.

Multiple alignment:
We perform multiple alignments on 10 sets of protein structures. It includes various *superfamilies* of SCOP **[16] (***calcium-binding*, *4-helix bundle*, *superhelix*, *concanavalin*, *tRNA synthetase*, *G-proteins* and *PTB domains*), and clusters of protein interfaces from PRINT **[17].**

The results are shown in Table 2 (**supplementary material**). It shows the largest alignment when the cardinality is 2 up to the total number of input structures. In the comparison with MultiProt, there are cases where SOIL performed better and also cases where MultiProt performed better. The results show that sometimes sequence order may be useful in protein structural comparison and at the other times it may limit the alignment result. The running time of both tools were roughly the same. SOIL usually generated a larger alignment than MultiBind* and ran much faster. The experimental results suggest that SOIL is more efficient than MultiBind* mainly due to the application of frequent itemset mining for subset alignment.

Note that MultiBind* takes a pivot-based approach by selecting a pivot and then aligning every structure with it so that every pair of aligned points has distance at most ε. However, we observed from the results that when multiple structures are aligned in this way, many aligned points become more than ε distance apart. On the contrary, SOIL compares structures simultaneously and produces a larger alignment, ensuring that all common substructures discovered are similar to each other.

Conclusion:
This paper studies the multiple structural alignment problem for protein structures. We designed an algorithm SOIL that works well with less information and assumptions. In particular, SOIL is sequence order independent and can perform subset alignment with a more restrictive similarity measurement. Our proposed algorithm SOIL makes use of the geometric hashing technique from computer vision, and the frequent itemset mining technique from data mining. Both techniques have been used in a wide range of applications and algorithms. We demonstrated the efficiency and effectiveness of the SOIL algorithm through experiments. SOIL compares structures simultaneously, discovers large common substructures, and ensures that the common substructures detected are similar to each other. Experiments have shown its applications to the alignment of various protein structures including protein chains and protein interfaces.

Acknowledgements:
We thank Christian Borgelt for providing an open source implementation of FP-Growth algorithm. This research was partially supported by Hong Kong RGC GRF grant (HKU 775207M).

References:
[1] T. Akutsu and M. M. Halldorson, *Theoretical Computer Science*, 233: 33 (2000).
[2] T. Akutsu and K. Sim, *Genome Informatics*, 10: 23 (1999).
[3] C. Ambühl, *et al.*, *Proc. ESA*, (2000).
[4] K.S. Arun, *et al.*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5): 698 (1987).
[5] O. Bachar, *et al.*, *Protein Engineering*, 6(3): 279 (1993).
[6] C. Borgelt, *Proc. OSDM*, (2005).
[7] O. Carugo, *Current protein and peptide science*, 8: 219 (2007).
[8] S.C. Chen and T. Chen, *Proc. Workshop on Genomic Signal Processing and Statistics* (2002).
[9] O. Dror, *et al.*, *Protein Science*, 12: 2492 (2003).
[10] J.F. Gibrat, *et al.*, *Current Opinion in Structural Biology*, 6(3): 377 (1996).
[11] J. Han, *et al.*, *Proc. SIGMOD* (2000).
[12] L. Holm and C. Sander. *J. Mol. Biol.*, 233: 123 (1993).
[13] J. Huan, *et al.*, *J. Comput. Biol.*, 12(6): 657 (2005).
[14] N. Leibowitz, *et al.*, *J. Comput. Biol*. 8(2): 93 (2001).
[15] D. Lupyan, *et al.*, *Bioinformatics* 21(15): 3255 (2005).
[16] A.G. Murzin, *et al.*, *J. Mol. Biol.*, 247: 536 (1995).
[17] http://prism.ccbb.ku.edu.tr/interface/
[18] A. Sacan, *et al.*, *Bioinformatics*, 23(6): 709 (2007).
[19] M. Shatsky, *et al.*, *Proteins: Struture, Function, and Bioinformatics*, 56: 143 (2004).
[20] M. Shatsky, *et al.*, *J. Comput. Biol*, 13(2): 407 (2006).
[21] A. Shulman-Peleg, *et al.*, *Proc. International Symp. on Computational Life Science*, 91 (2005).
[22] W. Taylor, *et al.*, *Protein Sci.* 3(10): 1858 (1994).
[23] H.J. Wolfson and I. Rigoutsos. *IEEE Computer Science and Engineering*, 4(4): 10 (1997).
[24] Y. Ye and A. Godzik. *Bioinformatics*, 21(10): 2362 (2005).
[25] M. Menke, *et al*., *PLoS Comput. Biol*., 4(1): e10 (2008).

Edited by Tan Tin Wee
Citation: Siu *et al***,** Bioinformation 4(8): 366-370 (2010)

Supplementary material:

Equation 1 & 2:

$$\vec{e_1} = \frac{\vec{CN} \times \vec{CC_\alpha}}{|\vec{CN} \times \vec{CC_\alpha}|}, \quad \vec{e_2} = \frac{\vec{CN}}{|\vec{CN}|}, \quad \vec{e_3} = \vec{e_1} \times \vec{e_2} \tag{1}$$

Note that the above vectors are normalized and hence are unit vectors. Any other point S, $j \neq i$, in the same structure can be transformed by this LRF to new coordinates (x, y, z), after solving the equation

$$S_j^k - S_i^k = x\vec{e_1} + y\vec{e_2} + z\vec{e_3} \tag{2}$$

Table 1. Comparison of pairwise alignment results. It shows the sizes of the alignments measured under the bottleneck metric. Numbers in the brackets are the original sizes measured under the algorithms' own metrics.

S_1 (size)	S_2 (size)	C-alpha match	MultiProt	MultiBind	SOIL
1fxiA (96)	1ubq (76)	51 (51)	50 (50)	56	58
1ten (89)	3hhrB (195)	75 (81)	82 (82)	81	82
3hlaB (99)	2rhe (114)	62 (62)	66 (66)	65	68
2azaA (129)	1paz (120)	51 (51)	76 (76)	73	81
1cewI (108)	1molA (94)	66 (66)	70 (72)	71	75
1cid (177)	2rhe (114)	70 (70)	81 (81)	78	82
1crl (534)	1ede (310)	177 (180)	190 (195)	187	205
2sim (381)	1nsbA (390)	194 (199)	219 (228)	206	233
1bgeB (159)	2gmfA (121)	72 (72)	82 (83)	80	87
1tie (166)	4fgf (124)	85 (87)	80 (81)	94	96
1dly (121)	1uvy (116)	112 (112)	112 (113)	113	114
2cblA (305)	1b3q (738)	130 (130)	140 (142)	149	160
1a04AB (43)	1hacAC (45)	20 (20)	18 (19)	22	23
1a3rLP (46)	1nakLP (40)	29 (30)	30 (31)	31	31
1ggiLP (39)	1a3rLP (46)	29 (29)	28 (28)	30	30

Table 2. Comparison of multiple alignment results. The table shows the running time in the first line followed by the sizes of the alignments at different cardinalities. The numbers in brackets specify the sizes of the alignment under the tools own metric. Entries marked with a '-' mean that the alignment of the corresponding test cases took over 3 hours and still did not terminate.

Structures	Average size	MultiProt	MultiBind*	SOIL
Calcium binding (6)	140	7s	2h22m42s	5s
4cpv, 2scpA, 2sas,		2: 127 (129)	2: 127 (127)	2: 128
1top, 1scmB, 3icb		3: 53 (61)	3: 54 (62)	3: 57
		4: 44 (56)	4: 44 (52)	4: 49
		5: 40 (53)	5: 28 (43)	5: 40
		6: 28 (45)	6: 19 (38)	6: 27
4-helix bundle (4)	324	6s	4m4s	7s
1f4n, 2cblA, 1b3q		2: 140 (142)	2: 149 (149)	2: 146
1rhgA		3: 58 (78)	3: 58 (79)	3: 71
		4: 39 (73)	4: 26 (62)	4: 37
Superhelix (5)	205	8s	1h32m21s	1m44s
1lxa, 1qq0, 1xat,		2: 143 (144)	2: 157 (157)	2: 158
2tdt, 1fwyA:252-328		3: 106 (114)	3: 112 (124)	3: 116
		4: 81 (87)	4: 73 (89)	4: 81
		5: 64 (68)	5: 57 (71)	5: 63

Supersandwich (3) 1bgmI:731-1023, 1cb8A:336-599, 1oacA:301-724	327	4s 2: 141 (150) 3: 63 (95)	3m52s 2: 153 (153) 3: 67 (98)	3s 2: 153 3: 71
Concanavalin (10) 2bqpA, 1gbg, 2galA, 1d2sA, 1sacA, 1a8d:1-247, 1kit:25-216, 2sli:81-276, 6cel, 1xnb	220	1m22s 2: 157 (160) 3: 87 (109) 4: 59 (76) 5: 34 (69) 6: 32 (60) 7: 23 (51) 8: 21 (42) 9: 15 (37) 10: 8 (32)	-	7m5s 2: 157 3: 102 4: 58 5: 46 6: 33 7: 21 8: 19 9: 16 10: 14
tRNA synthetase (4) 1adjA, 1hc7A, 1qf6A 1atiA-AntiCodon	490	19s 2: 321 (321) 3: 183 (240) 4: 117 (163)	1m23s 2: 305 (305) 3: 141 (200) 4: 55 (117)	55s 2: 303 3: 160 4: 66
G-proteins (6) 1agr, 1tad, 1gfi, 1tx4, 1grn, 1wql	596	1m36s 2: 365 (365) 3: 305 (310) 4: 134 (148) 5: 116 (139) 6: 42 (79)	-	17m19s 2: 366 3: 307 4: 129 5: 113 6: 33
PTB domain (6) 1x11, 1irs, 1shc, 1ddm, 2nmb, 1evh	168	9s 2: 127 (127) 3: 84 (97) 4: 62 (76) 5: 41 (61) 6: 20 (50)	3h13m30s 2: 123 (123) 3: 60 (82) 4: 35 (61) 5: 15 (51) 6: 13 (39)	7s 2: 121 3: 72 4: 40 5: 24 6: 14
PRINT cluster 45 (3) 1g4yBR, 1a2xAB, 1yv0IC	101	0.3s 2: 67 (67) 3: 49 (52)	0.8s 2: 66 (66) 3: 31 (42)	0.1s 2: 66 3: 34
PRINT cluster 8158 (3) 43caAC, 1b88AB, 1d9kAE	52	0.1s 2: 53 (53) 3: 42 (42)	0.1s 2: 53 (53) 3: 42 (42)	0.1s 2: 53 3: 42