PROC. 4TH WABI-04, VOL. 3240 IN LNBI, PP. 38-49, SPRINGER, 2004

A Polynomial-Time Algorithm for the Matching of Crossing Contact-Map Patterns

Jens Gramm*

International Computer Science Institute
1947 Center Street, Suite 600, Berkeley, CA 94704.
gramm@icsi.berkeley.edu

Abstract. Contact maps are a model to capture the core information in the structure of biological molecules, e.g., proteins. A contact map consists of an ordered set S of elements (representing a protein's sequence of amino acids), and a set A of element pairs of S, called *arcs* (representing amino acids which are closely neighbored in the structure). Given two contact maps (S, A) and (S_p, A_p) with $|A| \geq |A_p|$, the CONTACT MAP PATTERN MATCHING (CMPM) problem asks whether the "pattern" (S_p, A_p) "occurs" in (S, A), i.e., informally stated, whether there is a subset of $|A_p|$ arcs in A whose arc structure coincides with A_p. CMPM captures the biological question of finding structural motifs in protein structures. In general, CMPM is NP-hard. In this paper, we show that CMPM is solvable in $O(|A|^6|A_p|^2)$ time when the pattern is $\{<, \not)\}$-structured, i.e., when each two arcs in the pattern are disjoint or crossing. Our algorithm extends to other closely related models. In particular, it answers an open question raised by Vialette that, rephrased in terms of contact maps, asked whether CMPM for $\{<, \not)\}$-structured patterns is NP-hard or solvable in polynomial time. Our result stands in sharp contrast to the NP-hardness of closely related problems. We provide experimental results which show that contact maps derived from real protein structures can be processed efficiently.

1 Introduction

Since the function of biological molecules is highly associated with their three-dimensional structure, structure analysis is an important area of computational biology. Combinatorial models have been developed to capture the structure of molecules, e.g., *contact maps* for protein structure analysis [6] and *arc annotations* for RNA structure analysis [11]. Focusing in the following on proteins, we find that experimentally determined structural information is available for a large number of proteins [2]. One way to organize and analyze these data is to classify proteins according to their structure, a task that in many cases still involves human interaction. Herein, proteins are considered to belong to the same class if they share structural features, even if their primary sequence may not be similar. The resulting question is to determine whether a given protein structure exhibits a given structural feature where such a feature may be formed by amino acids which are not necessarily contiguous in the protein sequence. Here, we address

* This research was supported through a postdoc fellowship by the DAAD (German Academic Exchange Association).

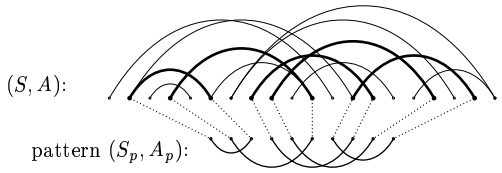

Fig. 1. Example of a yes-instance of CONTACT MAP PATTERN MATCHING, consisting of two contact maps (S, A) and (S_p, A_p). The occurrence of the pattern in the upper contact map is indicated by bold arcs, the corresponding mapping is indicated by dotted lines

this kind of pattern matching problem for the model of contact maps and a special class of patterns.

Problem definition. A *contact map* (S, A) consists of an ordered set S and a set $A = \{ (e_l, e_r) \mid e_l, e_r \in S, e_l < e_r \}$. Each pair in A is referred to as an *arc* and, consequently, A is called *arc set*. Given two contact maps (S, A) and (S_p, A_p) with $|S_p| < |S|$, we say that (S_p, A_p) *occurs* in (S_p, A_p) if there is a one-to-one mapping M of S_p to a size-$|S_p|$ subset S' of S such that, for $e, e' \in S_p$, we have both $e < e' \Rightarrow M(e) < M(e')$ and $(e, e') \in A_p \Rightarrow (M(e), M(e')) \in A$. Then, the central problem of this paper is given as follows:

CONTACT MAP PATTERN MATCHING (CMPM)
Given: Contact maps (S, A) and (S_p, A_p).
Question: Does (S_p, A_p) occur in (S, A)?

An illustration of a yes-instance of CMPM is given in Fig. 1. We refer to (S_p, A_p) as the pattern and we abbreviate $n := |A|$ and $m := |A_p|$. From the order of elements in S, we can in a natural way derive binary relations between arcs. For $a, a' \in A$ with $a = (e_l, e_r)$ and $a' = (e'_l, e'_r)$, we say that $a < a'$ (*a precedes* a') iff $e_l < e_r < e'_l < e'_r$. We say that $a \sqsubset a'$ (a is *nested* in a') iff $e'_l < e_l < e_r < e'_r$. Finally, we say that $a \between a'$ (a *crosses* a') iff $e_l < e'_l < e_r < e'_r$. A contact map is $\{<, \between\}$-structured if for each two arcs a and a' either the preceding-relation or the crossing-relation applies, i.e., $a < a'$, $a' < a$, $a \between a'$, or $a' \between a$; other restricted contact map classes are defined analogously.

Previous work. In general, CMPM is NP-complete which can be shown in analogy to [4] where a somewhat different model is used. The model of contact maps received particular attention for computing the similarity of two proteins, formalized as the question for a maximum-size pattern that occurs in both given contact maps [6, 9].

Vialette studied a problem closely related to CMPM in the area of RNA structure comparison [12]. His results imply, among others, that CMPM is NP-hard for $\{\sqsubset, \between\}$-structured patterns, and is solvable in $O(n^2)$ time (in $O(n^2 \log n)$ time) for $\{\sqsubset\}$-structured ($\{\between\}$-structured) patterns. CMPM for $\{<, \sqsubset\}$-structured patterns is solvable in quadratic time [5]. A summary of some results in given in Table 1. The remaining open question in this context, raised by Vialette [12], was—rephrased in terms of contact maps—to determine whether CMPM for $\{\sqsubset, \between\}$-structured patterns is solvable in polynomial time.

pattern structure	complexity of CMPM
$\{<, \sqsubset, \between\}$	NP-complete [4]
$\{\sqsubset, \between\}$	NP-complete [12]
$\{<, \between\}$	$n^6m^{2\ (*)}$
$\{<, \sqsubset\}$	$n \log n + nm$ [5]

Table 1. Overview on results for the complexity of the CONTACT MAP PATTERN MATCHING problem for restricted pattern structures. The result of this paper is marked by an asterisk

New Results. In this paper, we answer this open question by giving a polynomial-time algorithm for CMPM with $\{<, \between\}$-structured patterns. The algorithm relies on an involved realization of the dynamic programming principle and has $O(n^6m^2)$ running time. Due to heuristic improvements, the running time of the algorithm is faster than this worst-case bound suggests. We give experimental results for simulated contact maps and ones derived from real protein structures. Our results show that these datasets can be processed efficiently.

Our result is surprising since it lies at the border of CMPM versions that are still solvable in polynomial time (see Table 1). We show that our algorithm generalizes from contact maps to closely related models, in particular to 2-interval sets as discussed by Vialette [12]. Moreover, we show how our result can be used to give a fixed-parameter algorithm searching for the maximum-size $\{<, \between\}$-structured pattern occurring in two given contact maps or 2-interval patterns, respectively. For 2-interval sets, it was recently established that this question is NP-hard [3]. Notably, our algorithm only poses a constraint on the pattern, no constraint is posed onto the contact map in which we search the pattern. The observation that basic secondary structure elements like alpha helices and anti-parallel beta sheets exhibit $\{<, \between\}$-structured patterns supports the significance of this natural class of pattern structures in proteins. Moreover, in combination with the polynomial-time algorithm for CMPM restricted to $\{<, \sqsubset\}$-structured patterns [5], our algorithm provides an important building stone for further results.

Due to lack of space, most proofs are deferred to the full version of the paper.

2 Additional Notation

As a convention, we use $A_p = \{p_0, \ldots, p_{m-1}\}$ to denote the arc set of the pattern and we use $A = \{a_0, \ldots, a_{n-1}\}$ as the arc set of the contact map in which the pattern is searched. Given an arc $a = (e_l, e_r)$, we use $\mathrm{l}(a)$ to refer to the *left endpoint* e_l of a and we use $\mathrm{r}(a)$ to refer to the *right endpoint* e_r of a. We assume that both sets are ordered increasingly by their right endpoints and, if some arcs have the same right endpoint, subsequently by their left endpoint; this order, if not existing, can be obtained for A in time $O(|A| \log |A|)$. In the following, we introduce the concepts of *anchors* and *windows*, defined for the arc set of $\{<, \between\}$-structured patterns, which will be essential for stating our algorithm. Given a $\{<, \between\}$-structured contact map (S_p, A_p) and an arc $p \in A_p$, we use $\mathrm{anchor}(p)$ to refer to the smallest (w.r.t. the order of A) arc $p' \in A_p$, if it exists, such that $\mathrm{l}(p') < \mathrm{l}(p) < \mathrm{r}(p') < \mathrm{r}(p)$. This implies that there is no arc $p'' \in A_p$

Fig. 2. Illustration of two concepts for structure patterns: (a) Arc p_j is *anchor* of both arcs p_i and p_{i+1}. (b) The pair of arcs p_{j-1} and p_j comprises the j-*window* with respect to arc p_i

with $p'' \between p'$, $p'' \between p$, and $\mathrm{l}(p) < \mathrm{r}(p'') < \mathrm{r}(p')$. If no such arc $p' \in A_p$ exists, then anchor$(p) := \perp$. The concept of anchors is illustrated in Fig. 2(a). For integer i with $0 \leq i \leq m-1$, we use anchor(i) to refer to the index of anchor(p_i), i.e., anchor$(i) = j$ iff anchor$(p_i) = p_j$. Given a contact map (S_p, A_p) and two arcs $p_j, p_i \in A_p$ such that $p_{j-1} \between p_i$, and either $p_j \between p_i$ or $p_j = p_i$, we refer to the pair (p_{j-1}, p_j) as the j-*window* of p_i. This concept is illustrated in Fig. 2(b).

As an abbreviation, we use, for an ordered set $A' = \{a_{i_1}, a_{i_2}, \ldots, a_{i_r}\}$ with $A' \subseteq A$, cross$(a_{i_1}, a_{i_2}, \ldots, a_{i_r})$ to denote that all $a_{i_s}, a_{i_t} \in A'$ with $1 \leq s < t \leq r$ satisfy $a_{i_s} \between a_{i_t}$. Given two contact maps (S, A) and (S_p, A_p), let $A' := \{a_{i_1}, a_{i_2}, \ldots, a_{i_r}\}$ and $A'_p := \{p_{j_1}, p_{j_2}, \ldots, p_{j_r}\}$ for some $r < m$, with $a_{i_s} \in A$ and $p_{j_s} \in A_p$ for $1 \leq s \leq r$. We say that A'_p *matches* A' if, for $1 \leq s < t \leq r$, $p_{j_s} \between p_{j_t}$ iff $a_{i_s} \between a_{i_t}$. Intuitively, this means that the arc structures of A' and A'_p coincide. We say that $A_p[j_1, j_2, \ldots, j_r]$ *matches* $A[i_1, i_2, \ldots, i_r]$ if $(S_p, \{p_{j_1}, \ldots, p_{j_r}\})$ occurs in (S, A) while mapping p_{j_1} to a_{i_1}, p_{j_2} to a_{i_2}, ..., and p_{j_r} to a_{i_r}. Note that the requirement that A'_p matches A' is local, comparing only the arc structure of A'_p and A', whereas the requirement that $A_p[j_1, j_2, \ldots, j_r]$ matches $A[i_1, i_2, \ldots, i_r]$ is global, asking, in addition, for a corresponding occurrence in A of the pattern up to arc p_{j_r}.

We say that a contact map (S, A) is *connected* if there is no bipartition of A into two non-empty subsets A_1 and A_2 such that arcs from A_1 do not cross with arcs from A_2. For an easier exposition, we assume in the following that the arc set of the pattern is *connected*. Further, we assume that in the pattern (S_p, A_p), every element in S_p is endpoint of an arc in A_p.

3 Dynamic Programming Algorithm

In this section we present the algorithm solving CMPM restricted to $\{<, \between\}$-structured patterns. We describe the algorithm in top-down fashion, explaining data structures and an overview in Sect. 3.1 and giving the details of the dynamic programming steps in Sect. 3.2 and 3.3. Sect. 3.4 states the running time and the correctness of the algorithm, Sect. 3.5 points out generalizations of the algorithm to closely related models, and Sect. 3.6 shows an application of this algorithm in a more general problem setting.

3.1 Overview

Central for our algorithm are two sets that we define for every s, $0 \leq s < n$, and for every i, $1 \leq i < m$:

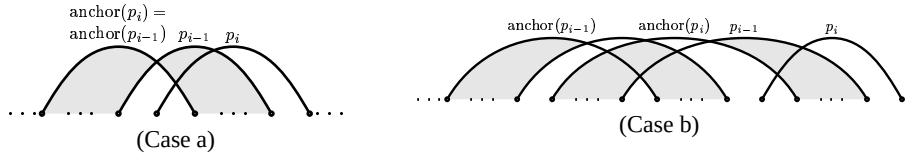

Fig. 3. Given $p_i \in A_p$ with $i > 1$, we distinguish two cases: (Case a) anchor$(p_{i-1}) =$ anchor(p_i) or (Case b) anchor$(p_{i-1}) <$ anchor(p_i)

- The set $T_{s,i}$ contains all c, $0 \leq c < n$, such that $A_p[\text{anchor}(i), i]$ matches $A[c, s]$. Thus, assuming that we match p_i with a_s, it stores possible matches for the anchor of pattern arc p_i.
- The set $S_{s,i}$ contains all 4-tuples $(c, w_1, w_2; j)$ with $0 \leq c, w_1, w_2 < n$ and anchor$(i) < j \leq i$ such that $A_p[\text{anchor}(i), j-1, j, i]$ matches $A[c, w_1, w_2, s]$. Thus, assuming that we match $p_{\text{anchor}(i)}$ with a_c and p_i with a_s, it stores possible matches for the windows of pattern arc p_i.

In fact, set $S_{s,i}$ is a refinement of set $T_{s,i}$ as the elements of $T_{s,i}$ can be easily obtained from $S_{s,i}$. For the sake of a clearer exposition of our algorithm, we choose to compute these sets separately: First, set $T_{s,i}$ is computed and for every element that is added to $T_{s,i}$, we then compute the corresponding 4-tuples that have to be added to $S_{s,i}$. The algorithm to be presented in the following sections computes the sets, starting with $i = 1$ and computing the sets for a value i with $1 < i < m$ based on the contents of the sets for value $i - 1$. It, thus, follows the classical algorithmic paradigm of dynamic programming.

The algorithm computes sets $T_{s,i}$ and $S_{s,i}$ in the order of ascending values of i. The algorithm has an *initialization phase* that computes the sets $T_{s,1}$ and $S_{s,1}$: Set $T_{s,1}$ contains all c, $1 \leq c < n$, such that $a_c \between a_s$. Since p_0 is necessarily the anchor of p_1 and since (p_0, p_1) constitutes the only window of arc p_1, set $S_{s,1}$ contains all 4-tuples $(c, c, s; 1)$ for which $c \in T_{s,1}$. The *dynamic programming phase* of the algorithm, then, computes $T_{s,i}$ and $S_{s,i}$ for $i > 1$. This computation will be explained in the next two subsections. Having computed sets $T_{s,i}$ and $S_{s,i}$ for all $0 \leq s < n$ and all $1 \leq i < m$, the algorithm reports that the pattern is found when there exist c^* and s^*, $0 \leq c^*, s^* < n$, such that $c^* \in T_{s^*,m-1}$: Then, by definition of $T_{s^*,i}$, $A_p[\text{anchor}(m-1), m-1]$ matches $A[c^*, s^*]$, and thus (S_p, A_p) occurs in (S, A).

3.2 Matching Pattern Anchors

In this subsection, we show how we compute, for given $a_s \in A$ and $p_i \in A_p$, $i > 1$, the set $T_{s,i}$, based on the knowledge of sets $T_{q,i-1}$ and $S_{q,i-1}$ for all $a_q \in A$. The corresponding procedure in pseudocode is given in Fig. 4.

We distinguish two situations concerning p_i which have to be treated differently by the algorithm: Either (Case a) anchor$(p_{i-1}) =$ anchor(p_i), illustrated in Fig. 3(a), or (Case b) anchor$(p_{i-1}) <$ anchor(p_i), illustrated in Fig. 3(b). For each of these cases, we describe in the following how $T_{s,i}$ is computed.

procedure matchAnchor(a_s; p_i)
Global: Contact maps (S, A) and pattern (S_p, A_p), sets $T_{q,i-1}$ and $S_{q,i-1}$ for all $a_q \in A$.
Input: Arcs $a_s \in A$ and $p_i \in A_p$, $i > 1$.
Output: Sets $T_{s,i}$ and $S_{s,i}$.

Method:

```
T_{s,i}:=∅;
S_{s,i}:=∅;
if anchor(p_{i-1}) = anchor(p_i) then                          /*** (Case a) ***/
    for all a_c, a_r ∈ A with cross(a_c, a_r, a_s) do
        if {c} ∈ T_{r,i-1} then
            T_{s,i}:=T_{s,i}+{c};
            S_{s,i}:=S_{s,i}∪ matchWindows(a_c, a_c, a_r, a_s; p_i);
        end if
    end for
else /* anchor(p_{i-1}) < anchor(p_i) */                       /*** (Case b) ***/
    for all 0 ≤ w_1, w_2, r < n do
        if (w_1, w_2, r, s) matches (anchor(p_i) − 1, anchor(p_i), i − 1, i) then
            c_r:=minAnchor(w_1, w_2, r; anchor(i), i − 1);
            if (c_r, w_1, w_2; anchor(i)) ∈ S_{r,i-1} then
                T_{s,i}:=T_{s,i}+{w_2};
                S_{s,i}:=S_{s,i}∪ matchWindows(a_{c_r}, a_{w_2}, a_r, a_s; p_i);
            end if
        end if
    end for
end if
return (T_{s,i}, S_{s,i});
```

Fig. 4. Overview in pseudocode on the computation of set $T_{s,i}$ for some $a_s \in A$ and some $p_i \in A_p$ based on the knowledge of sets $T_{q,i-1}$ and $S_{q,i-1}$ for all $a_q \in A$. Procedure `matchWindows()` that is used to compute $S_{s,i}$ is given in Fig. 5; the call to Function `minAnchor()` is explained in Sect. 3.2

(Case a): For every $a_r \in A$ with $a_r \between a_s$, and for every $c \in T_{r,i-1}$ with cross(a_c, a_r, a_s), we add c to $T_{s,i}$. The resulting set is the final set $T_{s,i}$. Thus, for the computation of $T_{s,i}$ in this case, we iterate over all possible choices of $a_c, a_r \in A$.

(Case b): For every $a_r \in A$ with cross(a_c, a_r, a_s) and every $(c, w_1, w_2; \text{anchor}(i)) \in S_{r,i-1}$ with $a_{w_1} < a_s$ but $a_{w_2} \between a_s$, we add w_2 to $T_{s,i}$. The resulting set is the final set $T_{s,i}$. Thus, for the computation of $T_{s,i}$ in this case, we iterate over all possible choices of $a_c, a_{w_1}, a_{w_2}, a_r \in A$.

In fact, this computation can be improved in (Case b), saving the necessity to iterate over possible choices of a_c. Given $a_{w_1}, a_{w_2}, a_r \in A$ and $p_i \in A_p$, we use in Fig. 4 Function `minAnchor(`w_1`, `w_2`, `r`; anchor(`i`), `$i-1$`)` to return the minimum c—i.e., a_c has minimum right endpoint—such that $A_p[\text{anchor}(i-1), \text{anchor}(i) - 1, \text{anchor}(i), i-1]$ matches $A[c, w_1, w_2, r]$. This value of c is then used in the computation of $T_{s,i}$. The call `minAnchor(`w_1`, `w_2`, `r`; anchor(`i`), `$i-1$`)` needs only

constant time since—as will become evident in the following subsection—we can during the computation of set $S_{r,i-1}$ maintain an array that keeps, for every $a_{w_1}, a_{w_2} \in A$, track of the minimum value c having the described property.

In Fig. 4, the calls to Procedure `matchWindows()` are used to compute set $S_{s,i}$ and will be explained in the following subsection.

3.3 Matching Pattern Windows

In this subsection, we show how we compute, for given $a_s \in A$ and $p_i \in A_p$, $i > 1$, the set $S_{s,i}$, based on the knowledge of sets $S_{q,i-1}$ for all $a_q \in A$. The corresponding pseudocode is given in Figs. 4 and 5.

We compute the set $S_{s,i}$ in "slices", meaning that for given values a_{c_r}, a_{c_s}, $a_r \in A$ for which

$$A_p[\text{anchor}(i-1), \text{anchor}(i), i-1, i] \text{ matches } A[c_r, c_s, r, s], \tag{1}$$

Procedure `matchWindows()` computes the set $S'_{s,i} \subseteq S_{s,i}$ of 4-tuples $(c_s, w_1, w_2; j)$ for which $A_p[\text{anchor}(i-1), \text{anchor}(i), j-1, j, i-1, i]$ matches $A[c_r, c_s, w_1, w_2, r, s]$. Iterating over all possible choices of a_{c_r}, a_{c_s}, and a_r, we then compute all "slices" of $S_{s,i}$. In the following, we explain, firstly, how `matchWindows()` can be called in a more efficient way in the process of computing the sets $T_{s,i}$ (the computation of $T_{s,i}$ was explained in Sect 3.3). Secondly, we explain how `matchWindows()` is computed.

Instead of enumerating all possible values of a_{c_r}, a_{c_s}, and a_r in order to find, for given $a_s \in A$ and p_i, all values which satisfy (1), we observe that we already obtain these values in the process of computing $T_{s,i}$. Namely, if (1) is satisfied then Procedure `matchAnchor()` adds c_s to $T_{s,i}$ while matching $A_p[\text{anchor}(i-1), i-1]$ to $A[c_r, r]$. Therefore, as shown in Fig. 4, we call `matchWindows()` exactly when c_s is added to $T_{s,i}$, either in (Case a) or in (Case b).

Now we describe how Procedure `matchWindows()` is computed. With respect to the arc matches determined by (1)—these arc matches are the given arguments of Procedure `matchWindows()`—we call a pair $(w; j)$ for $a_w \in A$ and $\text{anchor}(i) \leq j < i$ *reachable* if there exists w_h for every $h = \text{anchor}(i), \text{anchor}(i)+1, \ldots, j$, such that (1) holds while matching p_h with w_h for all these h. The reachable pairs determine the 4-tuples of the set $S'_{s,i}$ to be constructed, as will be indicated in the following.

We compute the reachable pairs in an inductive way and store them in set $R_{s,i}$. More precisely, the induction starts with pair $(c_s; \text{anchor}(i))$ which belongs to $R_{s,i}$ in any case. Then, we loop through integers j with $\text{anchor}(i) < j < i$ in ascending order. For a given j, we iterate over all pairs $a_{w_1}, a_{w_2} \in A$ with $\text{cross}(a_{c_s}, a_{w_1}, a_{w_2}, a_s)$, and add (a_{w_2}, j) to $R_{s,i}$ if $(a_{w_1}, j-1)$ is already in $R_{s,i}$ and if $(c_r, w_1, w_2; j) \in S_{r,i-1}$. Moreover, we add $(c_s, w_1, w_2; j)$ to $S_{s,i}$. In addition to these 4-tuples added to $S_{r,i-1}$ during the above induction, we add, in any case, $(c_r, r, s; i)$ to $S_{r,i-1}$.

3.4 Running Time and Correctness

The running time and correctness of the algorithm presented in Sect. 3.1 to 3.3 is summarized in the following theorem.

procedure matchWindows(a_{c_r}, a_{c_s}, a_r, a_s; p_i)
Global: Contact maps (S, A) and pattern (S_p, A_p), set $S_{q,i-1}$ for every $a_q \in A$.
Input: Arcs $a_{c_r}, a_{c_s}, a_r, a_s \in A$ and arc $p_i \in A_p$ such that
$A_p[\text{anchor}(i-1), \text{anchor}(i), i-1, i]$ matches $A[c_r, c_s, r, s]$.
Output: Set $S'_{s,i}$ containing all 4-tuples $(c_s, w_1, w_2; j)$, $j < i$ such that
$A_p[\text{anchor}(i-1), \text{anchor}(i), j-1, j, i-1, i]$ matches $A[c_r, c_s, w_1, w_2, r, s]$.
Method:

```
S'_{s,i}:=∅;
R_{s,i}:=∅; /* to store reachable pairs */

/* initialize the set R_{s,i} */
R_{s,i}:=R_{s,i} + (c_s; anchor(i));

/* inductive computation of R_{s,i} and S'_{s,i} */
for all j = anchor(i) + 1 upto i − 1 do
    for all w_1, w_2 ∈ A do
        if (anchor(p_i), p_{j−1}, p_j, p_{i−1}, p_i) matches (a_{c_s}, a_{w_1}, a_{w_2}, a_r, a_s) then
            if (w_1; j − 1) ∈ R_{s,i} and (c_r, w_1, w_2; j) ∈ S_{r,i−1} then
                R_{s,i}:=R_{s,i} + (w_2; j);
                S'_{s,i} := S'_{s,i} + (c_s, w_1, w_2; j);
            end if;
        end if;
    end for;
end for;

/* final step in computation of S'_{s,i} */
S'_{s,i} := S'_{s,i} ∪ {c_s, r, s; i};

return S'_{s,i};
```

Fig. 5. Overview in pseudocode on the computation of (subsets of) $S_{s,i}$, for $a_s \in A$ and $p_i \in A_p$

Theorem 3.1. CONTACT MAP PATTERN MATCHING *restricted to* $\{\between, <\}$*-patterns is solvable in* $O(n^6 \cdot m^2)$ *time for a contact map of size* n *and a pattern of size* m.

For the proof of Theorem 3.1, we employ the following two lemmas (proofs omitted).

Lemma 3.2. *For* $i > 1$, *if* $T_{s,i-1}$ *and* $S_{s,i-1}$ *are computed correctly for* $j < i$, *then Procedure* `matchAnchor()` *correctly computes set* $T_{s,i}$. □

Lemma 3.3. *For* $i > 1$, *if* $T_{s,i-1}$ *and* $S_{s,i-1}$ *are computed correctly and if* $T_{s,i}$ *is computed correctly, then Procedure* `matchWindows()` *correctly computes* $S_{s,i}$. □

Proof (of Theorem 3.1). We prove this theorem by analyzing the algorithm presented in Sect. 3.1 to 3.3. We discuss the algorithm's running time and show its correctness.

Running Time. A set $T_{s,i}$ is computed for n values of s and m values of i. The dynamic programming step to compute a set $T_{s,i}$ for some fixed values of s and i can be done in $O(n^3)$ time as shown in Procedure `matchAnchor()`, Fig. 4: The worst case is determined by (Case b) in which we iterate over possible choices for $a_{w_1}, a_{w_2}, a_r \in A$.

The update of set $S_{s,i}$, for every found match, can be done in $O(n^2m)$ time as shown in Procedure `matchWindows()`, Fig. 5: We iterate over possible choices for $a_{w_1}, a_{w_2} \in A$ and $p_j \in A_p$. Altogether, this results in a total running time of $O(n^6m^2)$.

Correctness. Showing the correctness of the algorithm, basically comes down to showing that sets $T_{s,i}$ and $S_{s,i}$ are computed correctly. We prove this by induction on i, starting with $i = 1$. It is easy to check that sets $T_{s,1}$ and $S_{s,1}$ are computed correctly. For $i > 1$, the correctness of computing $T_{s,i}$ and $S_{s,i}$ follows with Lemmas 3.2 and 3.3. □

The running time estimate given in Theorem 3.1 is in fact a worst-case estimate. In practice, the six nested loops on A do not all have to consider every element of A as assumed in the worst-case estimation. In a preprocessing step, we can compute, for every $a \in A$, the set of arcs $a' \in A$ for which $a' \between a$. Then, e.g., in (Case a) of Procedure `matchAnchor()`, we only consider those arcs a_c and a_r for which $a_c \between a$ and $a_r \between a$. In the same way, we can speed-up all loops over A in Procedures `matchAnchor()` and `matchWindows()`.

3.5 Generalizations

In the following, we discuss the extension of the presented algorithm to two closely related models, namely the one of arc annotations and the one of 2-interval sets. Due to lack of space, we only sketch these extensions, omitting the details.

Arc annotations. Arc annotations are a model mainly used for modelling RNA structures [4, 5, 8, 11]. The main difference between arc annotations and contact maps is that the elements in S carry additional sequence information. Reframing this question in terms of contact maps, for a given alphabet Σ and a given contact map (S, A), we are also given a labeling $\ell : S \longrightarrow \Sigma$, resulting in a *labeled* contact map (S, A, ℓ). Given two labeled contact maps (S, A, ℓ) and (S_p, A_p, ℓ_p), we say that (S_p, A_p, ℓ_p) *occurs* in (S, A, ℓ) if there is a one-to-one mapping M of S_p to a size-$|S_p|$ subset S' of S such that, for $e, e' \in S_p$, we have $e < e' \Rightarrow M(e) < M(e')$, $(e, e') \in A_p \Rightarrow (M(e), M(e')) \in A$, and—in addition to the definition for "usual" contact maps—$\ell_p(e) = \ell(M(e))$. The corresponding CMPM problem is then formulated as in Sect. 1. The labeling can be used, e.g., to capture the primary sequence of the protein or to encode chemical properties of amino acids. In this way, we obtain a problem formulation that incorporates *both* sequence *and* structure information.

We extend our algorithm to labeled contact maps, by requiring, whenever we try to match an element in S with an element in S_p, to check whether their labelings coincide.

2-interval sets. Vialette introduced 2-interval sets in the context of RNA structure analysis. A 2-interval set is given as a pair (I, A) where I, in contrast to S in the definition of contact maps, is given as a set of intervals $[i_l, i_r]$ with positive integers i_l and i_r. Consequently, A is a set of interval pairs, the intervals of one pair being non-overlapping; non-paired intervals can, however, overlap. We specify a partial order on the intervals, saying that two intervals $i = (i_l, i_r)$ and $i' = (i'_l, i'_r)$ satisfy $i \prec i'$ iff $i_r < i'_l$. Apart from this change, we can still employ the definition of CMPM as given in Sect. 1, replacing contact maps by 2-interval sets.

The only change required for the algorithm is to adjust the definition of whether two arcs a and a' are crossing or preceding. For 2-interval sets, two arcs $a = (i, i')$ and

$a' = (j, j')$, for intervals i,i',j, and j', are crossing iff $i \prec j \prec i' \prec j'$. Analogously, a and a' are preceding iff $i \prec i' \prec j \prec j'$. Note that, consequently, intervals of a $\{<, \wr\}$-structured 2-interval pattern have pairwisely disjoint intervals.

3.6 Applications

In this subsection we discuss how we can use the algorithm presented in Sect. 3.1 to 3.3 in order to solve a related NP-hard problem discussed by Vialette [12] and Blin *et al.* [3]: the 2-INTERVAL PATTERN problem restricted to $\{<, \wr\}$-structured patterns. Given a 2-interval set (S, A) where S is a set of intervals and A is a set of 2-interval pairs, it asks for a maximum size 2-interval set (S_p, A_p) such that A_p is $\{<, \wr\}$-structured and occurs in (S, A); as usual, we denote $n = |A|$ and $m = |A_p|$. 2-INTERVAL PATTERN is NP-hard even when restricted to $\{<, \wr\}$-structured patterns [3]. With Theorem 3.1 and its generalization outlined in 3.5, we can show that this problem is fixed-parameter tractable with respect to parameter m, i.e., it is solvable in $f(m) \cdot \text{poly}(n, m)$ time:

Theorem 3.4. *The* 2-INTERVAL PATTERN *problem restricted to* $\{<, \wr\}$*-structured patterns is solvable in* $O(3^m \cdot n^6 m^2)$ *time.* □

The proof of Theorem 3.4 relies on a one-to-one correspondence of $\{<, \wr\}$-structures and the Dyck language [7]; similar arguments were already exploited in [1]. Theorem 3.4 directly translates to a special case of the well-known MAXIMUM CONTACT MAP OVERLAP (CMO) problem [6, 9]. Given two contact maps (S_1, A_1) and (S_2, A_2) ($n = \max(|A_1|, |A_2|)$), CMO asks for a maximum-size contact map (S_p, A_p) ($m = |A_p|$) that occurs both in A_1 and in A_2. Then, CMO restricted to $\{<, \wr\}$-structured overlaps requires that (S_p, A_p) is $\{<, \wr\}$-structured. Theorem 3.4 implies that CMO restricted to $\{<, \wr\}$-structured overlaps is fixed-parameter tractable with respect to $|A_p|$. However, it is open to show that this problem is NP-hard [3].

4 Experimental Results

In this section, we describe experiments that have been conducted with a straightforward C implementation of the algorithm presented in this work. Running times were measured on a Sun SunFire V100 with 1024MB of RAM running Solaris. The contact maps derived from entries of the PDB database are generated based on the distance between the c_α atoms of the protein sequence; the c_α atoms constitute the nodes of the elements of the contact map. Two c_α atoms share an arc if their distance is less than 5.5 Å. A pattern of size m is implanted into a given contact map by randomly choosing the $2m$ endpoints of the pattern among the elements of the contact map.

To give an idea about the practical running times of the algorithm, we took measurements when running our implementation on contact maps into which we implanted a random pattern. We used two kinds of contact maps, in which the pattern was searched, random ones and ones derived from protein domain structures selected from the PDB database such that representatives from different structural classes and architectures [10] were included. Our results are displayed in Fig. 6. We see that the algorithm can process contact maps containing several hundred arcs in acceptable running times (Fig. 6(a)).

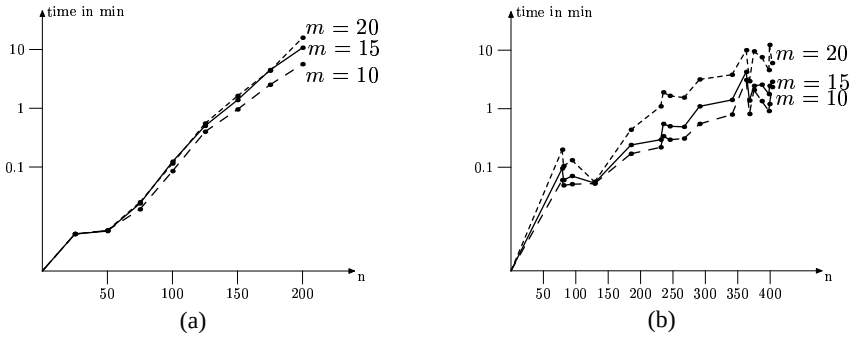

Fig. 6. Performance, displayed on a logarithmic scale, on contact maps of size n into which we implanted a random $\{<, \between\}$-structured contact map pattern of size m. (a) Random contact maps. (b) Contact maps derived from real protein domain structures

architecture	superfamily	m	#pos	#neg	fn	fp
Trefoil	Lectin	9	27	27	0	2
TIM Barrel	Glycosidases	15	20	20	2	4
3-Layer Sandwich	GroEL	10	17	18	1	1
2-Layer Sandwich	D-Amino Acid Oxidase	17	20	20	0	0

Fig. 7. Using the algorithm for CMPM for classification of protein domain structures, following the classification provided by the CATH database. The length of the chosen pattern is denoted by m, #pos denotes the number of the investigated domains that are in CATH listed as members of the superfamily, #neg denotes the number of investigated non-members. In our classification, we observed fp false positives and fn false negatives

Considering our results on contact maps which were derived from protein domain structures (Fig. 6(b)), it turns out that they were less difficult to process than random contact maps. In summary, the results show that the algorithm can efficiently process real protein structure data.

To give an idea about a realistic scenario in which the presented algorithm could be applied for structural classification of proteins, we investigated four examples of homologous protein domain superfamilies from the CATH database [10]. We chose, for each of these superfamilies, three protein domain structures from the superfamily and—by visual inspection—identified a significant $\{<, \between\}$-structured contact map pattern shared by the three structures. Then, we used this pattern to classify structures, on the one hand of protein domains listed in CATH as members of the superfamily, and of proteins domains listed in CATH as non-members but as 'structural relatives' of members. In difference to CATH, our classification thus *merely* relied on structural features and not on sequence similarity. The results of this experiment are displayed in Fig. 7. The result shows that already using this straightforward approach allows a correct classification in most cases. Making this idea competitive naturally requires an automatic detection of common patterns (e.g., using ideas of Sect. 3.6) and the use of a *set* of characteristic contact map patterns; these extensions remain to be explored.

5 Conclusion

This paper positively answered the previously open question [12] whether matching $\{<, \between\}$-structured patterns in contact maps (or 2-interval sets, respectively) can be done in polynomial time. Constraining ourselves to $\{<, \between\}$-structured patterns may seem a stringent requirement. However, we point out that the algorithm presented in this work gives, together with the quadratic-time algorithm for CMPM restricted to $\{<, \sqsubset\}$-structured patterns [5], the building-stones for a much more general pattern matching framework. It remains an issue of future research to classify those patterns that can now be matched in polynomial time by a combination of these two algorithms. Further, an open question is whether the running time for CMPM restricted to $\{<, \between\}$-structured patterns can be improved. It is to be explored how our results regarding the MAXIMUM CONTACT MAP OVERLAP (CMO) problem (see Sect. 3.6) perform in practice. Finally, it is open to prove or disprove the conjecture that CMO restricted to $\{<, \between\}$-structured overlaps is NP-hard.

Acknowledgements: I thank Nadja Betzler (University of Tübingen, Germany) for providing her implementation which generates contact maps from PDB structure files. Further, I thank Markus Stirnkorb and Jiong Guo (University of Tübingen) and Stéphane Vialette (LRI Orsay, France) for helpful discussions on this topic.

References

1. J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Computing the similarity of two sequences with nested arc annotations. *Theoretical Computer Science* 312:337–358, 2004.
2. H.M. Berman *et al.*. The Protein Data Bank. *Nucleic Acids Research*, 28:235–242, 2000. `http://www.rcsb.org/pdb/`.
3. G. Blin, G. Fertin, and S. Vialette. New results for the 2-interval pattern problem. In *Proc. of the 15th CPM*, to appear in LNCS, 2004, Springer.
4. P. A. Evans. Finding common subsequences with arcs and pseudoknots. In *Proc. of 10th CPM*, pages 270–280, volume 1645 in LNCS, Springer, 1999.
5. J. Gramm, J. Guo, and R. Niedermeier. Pattern matching for arc-annotated sequences. In *Proc. of the 22nd FSTTCS*, number 2556 in LNCS, pages 182–193, 2002, Springer.
6. D. Goldman, S. Istrail, and C. H. Papadimitriou. Algorithmic aspects of protein structure similarity. In *Proc. of the 40th FOCS*, pages 512–521, 1999, IEEE Computer Society.
7. M. A. Harrison. Introduction to Formal Language Theory. Addison-Wesley, Reading, 1978.
8. T. Jiang, G.-H. Lin, B. Ma, and K. Zhang. The Longest Common Subsequence problem for arc-annotated sequences. In Proc. of the 11th CPM, pages 154–165, volume 1848 in LNCS, Springer, 2000.
9. G. Lancia, R. Carr, B. Walenz, and S. Istrail. 1001 optimal PDB structure alignments: Integer Programming methods for finding the maximum contact map overlap. *Journal of Computational Biology*, 11(1):27–52, 2004.
10. C.A. Orengo, A.D. Michie, S. Jones, D.T. Jones, M.B. Swindells, and J.M. Thornton. CATH—A Hierarchic Classification of Protein Domain Structures. *Structure* 5(8):1093–1108. 1997. `http://www.biochem.ucl.ac.uk/bsm/cath/`.
11. Z. Wang and K. Zhang. RNA secondary structure prediction. In T. Jiang *et al*. (eds): *Current Topics in Computational Molecular Biology*, pages 345–364, MIT Press, 2002.
12. S. Vialette. On the computational complexity of 2-interval pattern matching problems. *Theoretical Computer Science*, 312(2–3):223–249, 2004.