

Joining Softassign and Dynamic Programming for the Contact Map Overlap Problem

Brijnesh J. Jain¹ and Michael Lappe²

¹ Berlin University of Technology, Faculty IV, Germany
jbj@cs.tu-berlin.de

² Max Planck Inst. for Molecular Genetics, Berlin, Germany
lappe@molgen.mpg.de

Abstract. Comparison of 3-dimensional protein folds is a core problem in molecular biology. The Contact Map Overlap (CMO) scheme provides one of the most common measures for protein structure similarity. Maximizing CMO is, however, NP-hard. To approximately solve CMO, we combine softassign and dynamic programming. Softassign approximately solves the maximum common subgraph (MCS) problem. Dynamic programming converts the MCS solution to a solution of the CMO problem. We present and discuss experiments using proteins with up to 1500 residues. The results indicate that the proposed method is extremely fast compared to other methods, scales well with increasing problem size, and is useful for comparing similar protein structures.

1 Introduction

Solutions to the problem of comparing protein structures are important in structural genomics for (i) protein function determination and drug design, (ii) fold prediction, and (iii) protein clustering [1]. For this, various methods have been devised. The most common are (i) RMSD [12], (ii) Distance Map Similarity [10], and (iii) Contact Map Overlap (CMO) [5,6]. Here, we focus on the CMO scheme.

To structurally compare two proteins using CMO, we first derive contact maps of each protein under consideration. A contact map consists of an ordered set of vertices representing the residues of a protein and of contacts (edges) connecting two vertices if the corresponding residues are geometrically close. The order of the vertices corresponds to the sequence order of the residues. Maximizing the CMO of both proteins aims at finding an order preserving alignment of both contact maps such that the number of common contacts is maximized.

CMO is NP-hard [8] and also hard to approximate [9]. The exponential time complexity initiated ongoing research on devising solutions to the CMO problem. Most methods employ discrete optimization techniques [1, 2, 3, 13, 14, 17, 18] and have been typically applied in a systematic way only to small-sized proteins with up to ≈ 250 residues.

In this paper, we suggest an approach with the following characteristics:

- The major tool of the proposed algorithm is based on nonlinear continuous rather than discrete optimization techniques.

- The proposed algorithm provides good suboptimal solutions for similar protein structures.
- The proposed algorithm is fast and scales well with problem size. Proteins with about 1500 residues can be aligned in less than 1.5 minutes on a Athlon 64 32000+ processor with 2 GHz. Comparable results have not been reported in the literature so far.

The proposed approach applies the softassign algorithm developed by [7, 11] for solving a related problem, the maximum common subgraph (MCS) problem. The MCS problem asks for the maximum number of common contacts without considering the sequence order of the contact maps. To convert the MCS solution of softassign to a feasible solution for CMO, we enhance softassign with a problem-dependent objective function and a dynamic programming post-processing procedure. In experiments using real proteins from the protein database (PDB), we investigated the behavior and performance of the modified softassign approach.

The rest of this paper is organized as follows: Section 2 introduces the CMO problem. In Section 3, we describe the proposed algorithm. Section 4 presents and discusses the results. Finally, Section 5 concludes with a summary of the main results and an outlook for further research.

2 Problem Statement

A *contact map* is an undirected graph $X = (V, E)$ consisting of an ordered set $V = \{1, \dots, n\}$ of vertices and a set $E \subset V^2$ of edges. Vertices represent residues of a folded protein and edges the contacts between the corresponding residues. The vertex set of X is also referred to as $V(X)$ and its edge set as $E(X)$. As usual, the adjacency matrix $\mathbf{X} = (x_{ij})$ of X is defined by

$$x_{ij} = \begin{cases} 1 & : \text{ if } (i, j) \in E(X) \\ 0 & : \text{ otherwise} \end{cases}.$$

Our goal is to structurally align two contact maps X and Y . For this, we ask for a partial vertex mapping

$$\phi : V(X) \rightarrow V(Y), \quad i \mapsto i^\phi$$

that maximizes some criterion function subject to certain constraints. In the following we specify both, the criterion function and the constraints.

A *common* or *shared contact* aligned by ϕ is a pair of edges $(i, j) \in E(X)$ and $(r, s) \in E(Y)$ such that $(r, s) = (i^\phi, j^\phi)$. The criterion function to be maximized is of the form

$$f(\phi) = \sum_{i, j \in \text{dom}(\phi)} x_{ij} y_{i^\phi j^\phi}, \quad (1)$$

where $\text{dom}(\phi)$ denotes the domain of the partial mapping ϕ . The terms $x_{ij} y_{i^\phi j^\phi}$ are either one or zero. We have $x_{ij} y_{i^\phi j^\phi} = 1$ if, and only if (i, j) and (i^ϕ, j^ϕ) are

common contacts. Hence, $f(\phi)$ counts the number of common contacts aligned by a feasible mapping ϕ .

The first constraint is that the partial mappings are injective. Let $\Pi(X, Y)$ denote the set of all injective partial mapping from $V(X)$ to $V(Y)$. Then the *maximum common subgraph (MCS) problem* is the problem of maximizing $f(\phi)$ subject to $\phi \in \Pi(X, Y)$.

The second constraint demands that a mapping ϕ from $\Pi(X, Y)$ preserves the order of the vertices. A mapping is order-preserving if

$$i < j \quad \Rightarrow \quad i^\phi < j^\phi$$

for all $i, j \in V(X)$. By $\Pi^+(X, Y) \subseteq \Pi(X, Y)$ we denote the subset of order-preserving mappings. The *maximum contact map overlap problem* is the problem of maximizing $f(\phi)$ subject to $\phi \in \Pi^+(X, Y)$. Thus, the CMO problem can be derived from the MCS problem by additionally imposing the order-preserving constraint.

3 Joining Softassign and Dynamic Programming

Our algorithm proceeds in two stages: In the first stage, we ignore the order-preserving constraint and approximately solve the MCS problem. The second stage converts the solution of the first stage to a feasible solution of CMO.

The main challenge is to find an approximate solution of the MCS problem in the first stage, which is close to a good solution of the CMO problem. To tackle this problem, the algorithm consists of three components:

- **The Softassign Algorithm.** In the first stage, we use softassign [7,11], one of the most powerful method for approximately solving the MCS problem.
- **Dynamic Programming.** The second stage takes the output of softassign as input and converts it to a feasible solution of the CMO problem using a dynamic programming approach. Dynamic programming yields an optimal solution given the output of softassign.
- **Compatibility function.** The compatibility function provides the crucial link between the first and second stage of the algorithm. It aims at reshaping the objective function of the MCS problem such that good local optima correspond to good solutions of the CMO problem.

In the following, we describe each component of the algorithm separately.

3.1 The Softassign Algorithm

Softassign minimizes a continuous quadratic formulation of the MCS problem. Section 3.1.1 presents the quadratic program and in Section 3.1.2, we describe the softassign algorithm. For a detailed presentation of softassign, we refer to [7].

In the following, we assume that X and Y are contact maps consisting of n and m vertices, respectively.

A Continuous Quadratic Program for the MCS Problem. To present a quadratic formulation of the MCS problem, we encode the partial vertex mappings as match matrices. Let $\phi : V(X) \rightarrow V(Y)$ be a mapping from $\Pi(X, Y)$. Then the matrix representation of ϕ is a binary $(n \times m)$ -matrix $\mathbf{M}_\phi = (m_{ij})$ with

$$m_{ij} = \begin{cases} 1 & : i^\phi = j \\ 0 & : \text{otherwise} \end{cases}$$

We may identify mappings ϕ from $\Pi(X, Y)$ with their associated matrices \mathbf{M}_ϕ . Therefore, we also write $\mathbf{M}_\phi \in \Pi(X, Y)$ by abuse of notation.

Using match matrices, the constraint $\phi \in \Pi(X, Y)$ can be rewritten as

$$\sum_{i=1}^n m_{ij} \leq 1, \quad \forall j \in V(Y) \tag{2}$$

$$\sum_{j=1}^m m_{ij} \leq 1, \quad \forall i \in V(X) \tag{3}$$

$$m_{ij} \in \{0, 1\}, \quad \forall i \in V(X), \forall j \in V(Y) \tag{4}$$

Let $c_{ijrs} = x_{ir}y_{js}$ denote the *compatibility* of x_{ir} and y_{js} . The quadratic integer formulation of criterion function (1) is

$$E(M) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^m \sum_{r=1}^n \sum_{s=1}^m m_{ij} m_{rs} c_{ijrs}. \tag{5}$$

Thus the MCS problem is equivalent to minimizing $E(M)$ subject to the constraints (2)-(4).

We replace constraint (4) by

$$m_{ij} \in [0, 1], \quad \forall i \in V(X), \forall j \in V(Y). \tag{6}$$

to obtain a continuous version of the quadratic integer program for the MCS problem. Matrices satisfying constraints (2), (3), and (6), are called *doubly stochastic*.

Softassign. Softassign minimizes $E(M)$ subject to the constraints (2), (3), and (6). The core of the algorithm implements a deterministic annealing process with annealing parameter T by the following iteration scheme

$$m_{ij}^{(t+1)} = a_i b_j \exp \left(-\frac{1}{T} \sum_{r=1}^n \sum_{s=1}^m m_{rs}^{(t)} c_{ijrs} \right), \tag{7}$$

where t denotes the time step. The scaling factors a_i, b_i computed by Sinkhorn's algorithm [16] enforce the constraints of a doubly stochastic matrix.

Algorithm 1 outlines the softassign algorithm (see [7]). To convert the inequality constraints (2) and (3), softassign uses slack variables by adding an

extra row and column to the matrix \mathbf{M} . By $\hat{\mathbf{M}} = (\hat{m}_{ij})$ we denote the augmented $(n + 1) \times (m + 1)$ -matrix. The parameters T_0 , T_f , and α describe the annealing schedule. The annealing parameter T is initialized with T_0 and gradually lowered by αT until it reaches the final value T_f . The parameters I_0 and I_1 specify the maximum number of steps of the Assignment and Sinkhorn loop.

Algorithm 1. Softassign

1. **Initialization:** $T \leftarrow T_0$ and $\hat{m}_{ij} \leftarrow 1 + \varepsilon$
 2. **while** $T > T_f$ **do**
 1. **for** $t_0 = 1$ **to** I_0 **do** (Assignment Loop)
 1. $q_{ij} \leftarrow \partial E / \partial m_{ij}$
 2. $m_{ij} \leftarrow \exp(q_{ij} / T)$
 3. **for** $t_1 = 1$ **to** I_1 **do** (Sinkhorn Loop)
 1. $r_i \leftarrow \sum_{j=1}^{m+1} \hat{m}_{ij}$
 2. $\hat{m}'_{ij} \leftarrow \hat{m}_{ij} / r_i$
 3. $c_j \leftarrow \sum_{i=1}^{n+1} \hat{m}'_{ij}$
 4. $\hat{m}_{ij} \leftarrow \hat{m}'_{ij} / c_j$
 2. $T \leftarrow \alpha T$
-

Since $c_{ijrs} = 1$ if, and only if, $(i, r) \in E(X)$ and $(j, s) \in E(Y)$, the complexity of softassign is $O(|E(X)| \cdot |E(Y)|)$.

3.2 Dynamic Programming

We use dynamic programming to convert the output \mathbf{M} of softassign to a feasible solution of the CMO problem, which is optimal with regard to \mathbf{M} .

Generally, let $\mathbf{W} = (w_{ij})$ be a $(n \times m)$ -matrix with elements $w_{ij} \in [0, 1]$. The dynamic programming approach maximizes the weight function

$$\omega(\mathbf{P}) = \sum_{i=1}^n \sum_{j=1}^m w_{ij} p_{ij}$$

subject to $\mathbf{P} = (p_{ij}) \in \Pi^+(X, Y)$.

The main idea of dynamic programming is to use optimal solutions of subproblems to find an optimal solution of the overall problem. To describe the subproblems, we need the notion of *induced subgraph*. Let Z be a graph and let $k \leq |V(Z)|$. By Z_k we denote the subgraph of Z induced by the first k vertices from Z . The vertex and edge set of Z_k are given by

$$V(Z_k) = \{1, \dots, k\} \subseteq V(Z) \quad \text{and} \quad E(Z_k) = E(Z) \cap V(Z_k)^2.$$

Let $k \leq n$ and $l \leq m$. The (k, l) -th subproblem maximizes

$$\omega(\mathbf{P}) = \sum_{i=1}^k \sum_{j=1}^l w_{ij} p_{ij}$$

subject to $\mathbf{P} \in \Pi^+(X_k, Y_l)$.

The dynamic programming algorithm consists of two stages. In the first stage, it converts the matrix \mathbf{W} to a score matrix $\mathbf{S} = (s_{kl})$. The scores s_{kl} correspond to optimal solutions of the (k, l) -th subproblem with respect to X_k and Y_l . In the second stage, the algorithm selects in a backward process correspondences with highest score such that all constraints of the CMO problem are satisfied.

The following algorithm outlines the procedure to construct a scoring matrix \mathbf{S} . The vector \mathbf{s}^* records the highest score for each column.

Algorithm 2. Scoring

1. **Initialization:** $\mathbf{S} = \mathbf{0}$ and $\mathbf{s}^* = \mathbf{0}$
2. **for** $i = 1$ **to** n **do**
 1. **for** $j = 1$ **to** m **do**
 1. $s_{ij} \leftarrow w_{ij} + \max\{s_1^*, \dots, s_{j-1}^*\}$
 2. **for** $j = 1$ **to** m **do**
 1. $s_j^* \leftarrow \max\{s_j^*, s_{ij}\}$

The complexity of Algorithm 2 is $O(nm)$. The matrix \mathbf{S} has the property that $s_{ij} \leq s_{kl}$ for all $i < k$ and $j < l$. We exploit this property in a backward process to construct a feasible solution of the CMO problem.

Algorithm 3. Recover feasible Solution

1. **Initialization:** $\mathbf{P} = \mathbf{0}$
2. $k \leftarrow m$
3. **for** $i = n$ **to** 1 **do**
 1. $k' \leftarrow \arg \max_{j \leq k} \{s_{ij}\}$
 2. $p_{ik'} \leftarrow 1$
 3. $k \leftarrow k' - 1$

Without using a more sophisticated data structure the complexity of Algorithm 3 is $O(nm)$.

3.3 Compatibility Function

In its original formulation, softassign minimizes the objective function (5), where the compatibilities are of the form $c_{ijrs} = x_{ir}y_{js}$. This choice of compatibilities ignores the order-preserving constraint for each partial mapping $\{i \mapsto j, r \mapsto s\}$ and may therefore result in a poor final solution for CMO.

Figure 1 illustrates some of the problems using the original compatibilities. Suppose that the MCS problem is the problem of maximizing an objective $f_{MCS}(\mathbf{M})$ subject to $\mathbf{M} \in \Pi(X, Y)$. For example, $f_{MCS}(\mathbf{M}) = -E(\mathbf{M})$, where $E(\mathbf{M})$ is the objective function (5). Similarly, the COM problem is the problem of maximizing some objective function $f_{CMO}(\mathbf{P})$ subject to $\mathbf{P} \in \Pi^+(X, Y)$. Figure 1 shows that f_{MCS} has three maxima $\mathbf{M}_1, \mathbf{M}_2$, and \mathbf{M}_3 . Applying dynamic programming converts the local maxima \mathbf{M}_i to feasible solutions \mathbf{P}_i of the CMO problem. The following undesirable situations can occur:

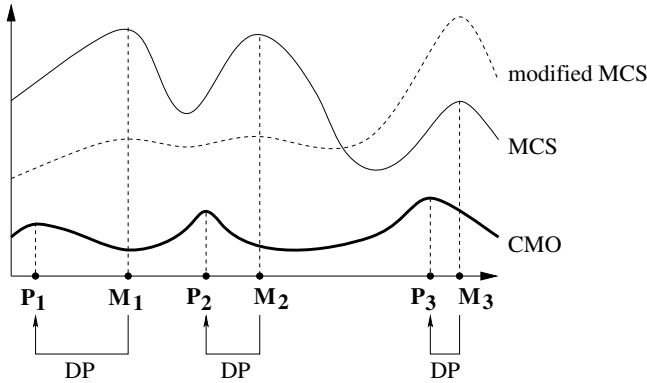


Fig. 1. Idealized example for illustrating the effects of compatibility values. The horizontal axis represents the continuous search space. The vertical axis shows the values of (fictive) objective functions to be maximized for the CMO, original MCS, and modified MCS problem. The original MCS uses $c_{ijrs} = x_{ir}y_{js}$ as compatibility values. The modified MCS uses more appropriate compatibility values for the CMO problem. Dynamic programming (DP) transforms the local maxima M_i of the original and modified MCS problem to local maxima P_i of the CMO problem.

- $f_{MCS}(M_1) = f_{MCS}(M_2)$, but $f_{CMO}(P_1) < f_{CMO}(P_2)$.
- $f_{MCS}(M_3) < f_{MCS}(M_i)$, but $f_{CMO}(P_3) > f_{CMO}(P_i)$ for $i = 1, 2$.

This observation suggests to define a compatibility function that reshapes the objective function of MCS in a similar way as shown in Figure 1.

The second issue to take into account when constructing a suitable compatibility function is time complexity. The time complexity of softassign using modified compatibility values is $O(|C|)$, where $C = \{(i, j, r, s) : c_{ijrs} \neq 0\}$. Thus, to keep the time complexity low the number of nonzero values c_{ijrs} should be small. Therefore, we avoid penalties $c_{ijrs} < 0$ for partial mappings $\{i \mapsto j, r \mapsto s\}$ that violate the constraints.

We suggest the following modified compatibility function:

$$c_{ijrs} = \begin{cases} 1/(1 + \alpha|r_{ir} - r_{js}|) & : (i, r) \sim (j, s) \\ \beta & : r_{ir} = r_{js} = 1 \\ 0 & : \text{otherwise} \end{cases}$$

where α and β are problem dependent parameters. The quantity $r_{kl} = |k - l|$ is the *range* of edge (k, l) . The notion $(i, r) \sim (j, s)$ means that the partial mapping $\phi = \{i \mapsto j, r \mapsto s\}$ satisfies both constraints of the CMO problem, that is $\phi \in \Pi^+(X, Y)$.

The modified compatibility function has the following characteristics:

- Since contact maps are sparse, most compatibility values are zero keeping the time complexity of softassign low.
- Only feasible partial mappings $\phi = \{i \mapsto j, r \mapsto s\}$ and their direct neighbors are positively weighted.

- Common contacts with similar range are considered to be more compatible to one another than common contacts with different range.

3.4 The SADP Algorithm: Assembling the Components

The SADP (Softassign and Dynamic Programming) procedure to approximately solve the CMO problem operates as follows:

Algorithm 4. CMO Algorithm

1. Define compatibility function
 2. Algorithm 1: $M = \text{softassign}(X, Y)$
 3. Algorithm 2: $S = \text{getScoreMatrix}(M)$
 4. Algorithm 3: $P = \text{getSolution}(S)$
-

Note that for solving the CMO problem, it is not necessary to exactly meet the constraints of a doubly stochastic matrix in Step 2. The task of softassign is merely to return a useful weighted matrix that yields a good scoring in Step 3. Therefore, we can terminate the Assignment and Sinkhorn loop after a maximum number of iteration steps. Additionally, we terminate a loop prematurely when the change of the previous and current match matrix is below a given threshold.

4 Experiments

To assess the performance, we applied SADP to three test problems. The algorithm was implemented in Java using JDK 1.2. Whenever possible, we compared SADP with a C implementation of the CAP algorithm proposed by Caprara and Lancia [1].¹ All experiments were run on a Linux PC with Athlon 64 3200+ processor (2.0 GHz CPU).

4.1 Sokol Test

In our first test series, we compared the performance of SADP against the following algorithms implemented in C:

Acronym	Reference	Platform
CAP	Caprara & Lancia [1]	Linux PC, Athlon 64 3200+, 2.0 GHz CPU
STR	Strickland et al. [17]	SGI workstation, 200 MHz CPU
XIE	Xie & Sahinidis [18]	Dell workstation, 3.0 GHz CPU

As test instances, we used the Sokol test set consisting of 11 alignment pairs of small-size proteins compiled by [17].

¹ The implementation of CAP has been provided by Lancia. In all experiments, we used the default parameter setting.

Table 1. Results of the Sokol test set. The first column shows the proteins to be structurally aligned. Columns 2-5 show the number of vertices and edges of the contact maps. Column 6 with identifier f_{SADP} shows the number of shared contacts found by SADP. Column 7 with identifier f_* shows the optimal solution found by CAP, STR, and XIE. Columns with identifiers of the form t_{ALG} refer to the computation time required by the respective algorithm $\text{ALG} \in \{\text{CAP}, \text{SADP}, \text{STR}, \text{XIE}\}$. Computation times are measured in seconds. Entries with value 0 (0.00) refer to computation times of less than 1 (0.01) second.

X-Y	$ V(X) $	$ E(X) $	$ V(Y) $	$ E(Y) $	f_{SADP}	f_*	t_{SADP}	t_{CAP}	t_{STR}	t_{XIE}
3ebx-1era	48	52	48	49	31	31	0.08	1.88	236	0.72
1bpi-2knt	51	58	44	42	27	29	0.06	1.94	182	0.46
1knt-1bpi	43	43	51	58	28	30	0.06	1.47	110	0.38
6ebx-1era	35	34	48	49	19	20	0.04	1.11	101	0.73
2knt-5pti	44	42	47	52	25	28	0.05	1.08	95	0.32
1bpi-1knt	51	58	43	43	28	30	0.05	1.53	19	0.24
1bpi-5pti	51	58	47	52	41	42	0.05	0.88	30	0.29
1knt-2knt	43	43	44	42	39	39	0.03	0.00	0	0.15
1knt-5pti	43	43	47	52	26	28	0.05	1.16	46	0.57
1vii-1cph	20	19	12	9	6	6	0.00	0.02	0	0.00
3ebx-6ebx	48	52	35	34	27	28	0.04	0.18	6	0.12
Total:					297	311	0.51	11.24	825	3.98

Table 1 summarizes the results. The results for STR and XIE are taken from [18]. The CAP, STR, and XIE algorithm returned an optimal solution f_* in all 11 trials giving a total of 311 common contacts. With 297 shared contacts, the solution quality of SADP is in average about 95.5% to optimal.

A fair comparison of computation time is difficult, because the other three algorithms were implemented in C and two of them (STR, XIE) were tested on different platforms. The results indicate that SADP has the best time performance. This indication will be approved for large-scale problems.

4.2 Experiment with Large-Scaled Data

The aim of our second test series is to evaluate the behavior and performance of SADP for large-scaled data. A comparison of SADP against the other three algorithms mentioned in the Sokol test is not possible because of time constraints. For this reason, we designed a semi-synthetical evaluation procedure.

We randomly selected 21 proteins from PDB having 800 up to 1500 residues. The corresponding contact maps have the following characteristics:

Vertices				Edges			
max	min	mean	std	max	min	mean	std
1488	801	959.4	172.2	5273	2717	3287	604.1

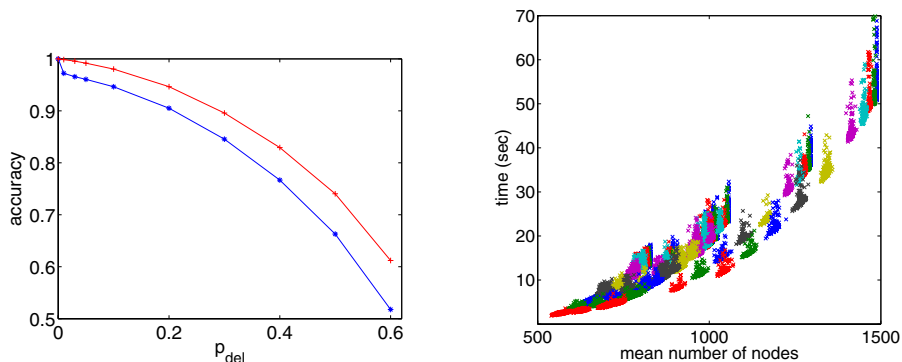


Fig. 2. Left: Average accuracy of SADP as a function of p_{del} . The upper curve shows the average accuracy normalized by the lower bound $|E(Y_2)|$ and the lower curve is the average accuracy normalized by $|E(Y)|$. **Right:** Computation time in *sec* as a function of the mean $(|V(X)| + |V(Y)|)/2$ for each trial (X, Y) .

Let \mathcal{X} denote the set of all 21 selected contact maps. We structurally aligned pairs (X, Y) of contact maps, where X is a contact map from \mathcal{X} and Y is a corrupted version of X . To create Y , we proceeded as follows: First, we randomly deleted each vertex of X with probability p_{del} . Let Y_1 denote the resulting contact map. Next, we randomly deleted each edge from Y_1 with the same probability p_{del} and to obtain contact map Y_2 . In a last step, we randomly connected each vertex with 10% probability to a randomly chosen vertex by an edge to obtain the final corrupted version Y . For each contact map $X \in \mathcal{X}$ and each parameter $p_{del} \in \{0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$, we generated 100 corrupted copies Y giving a total of 20,000 trials. This setting allows us to assess the solution quality of SADP: Given an alignment pair (X, Y) , we find that $|E(Y_2)|$ is a lower bound of the maximum number of shared contacts, because Y_2 is the contact map after randomly removing edges from X . Hence, Y_2 and X have $|E(Y_2)|$ edges in common. In the final step we randomly add edges to Y_2 to obtain Y . Therefore X and Y share at least $|E(Y_2)|$ common edges.

Figure 2 shows the accuracy and computation time of SADP. From the left plot of Figure 2 we see that the accuracy of softassign degrades with increasing level of corruption. This observation is in accordance with [?] for the maximum common subgraph problem. First attempts to improve this deficiency apply kernelized compatibility functions [?]. Thus, in its current form, SADP is suitable for problems, where detection of mid to high structural similarity between two proteins is relevant.

The right plot of Figure 2 shows that the computation time of SADP is less than 1.5 minutes for aligning the largest contacts maps of order ≈ 1500 . Aligning contact maps of order ≈ 800 took about 10-15 seconds. In contrast, we aborted CAP applied on two contact maps of order ≈ 800 after 5 days without obtaining a result.

Table 2. Protein domains of the Skolnick test set. Columns with identifier ID refer to the index assigned to the domains; columns with identifier PDB refer to the PDB code for the protein containing the domain; and columns with identifier CID refer to the chain index of a protein. If a protein consists of a single chain, the corresponding entry in the CID column is left empty. Note that the IDs differ from those used in [14].

ID	PDB	CID	ID	PDB	CID	ID	PDB	CID	ID	PDB	CID
1	1b00	A	11	4tmy	B	21	2b3i	A	31	1tri	
2	1dbw	A	12	1rn1	A	22	2pcy		32	3ypi	A
3	1nat		13	1rn1	B	23	2plt		33	8tim	A
4	1ntr		14	1rn1	C	24	1amk		34	1ydv	A
5	1qmp	A	15	1baw	A	25	1aw2	A	35	1b71	A
6	1qmp	B	16	1byo	A	26	1b9b	A	36	1bcf	A
7	1qmp	C	17	1byo	B	27	1btm	A	37	1dps	A
8	1qmp	D	18	1kdi		28	1hti	A	38	1fha	
9	3chy		19	1nin		29	1tmh	A	39	1ier	
10	4tmy	A	20	1pla		30	1tre	A	40	1rcd	

4.3 Skolnick Clustering Test

The aim of the Skolnick clustering test originally suggested by Skolnick and described in [14] is to classify 40 proteins into four families according to their cluster membership. The protein domains are shown in Table 2. In the following, we use their assigned indexes to refer to the respective domains.

Table 3 describes the protein domains and their families. Its fourth column with identifier *Seq. Sim.* indicates that sequence alignment fails for clustering the protein domains according to their family membership. This motivates structural alignment for solving the Skolnick clustering test.

The contact maps of the domains were provided by [18] and differ slightly from the ones compiled by [14]. To cluster the Skolnick data, we used the similarity measure [18]

$$\sigma(X, Y) = \frac{2f_{\text{SADP}}(X, Y)}{|E(X)| + |E(Y)|},$$

where $f_{\text{SADP}}(X, Y)$ denotes the number of shared contacts found by SADP.

Table 3. Protein domains of the Skolnick test set and their categories as taken from [1]. Shown are the characteristics of the four families, the mean number of residues, the range of similarity obtained by sequence alignment and the identifiers of the protein domains.

Family	Style	Residues	Seq. Sim.	Proteins
1	alpha-beta	124	15-30%	1-14
2	beta	99	35-90%	15-23
3	alpha-beta	250	30-90%	24-34
4		170	7-70%	35-40

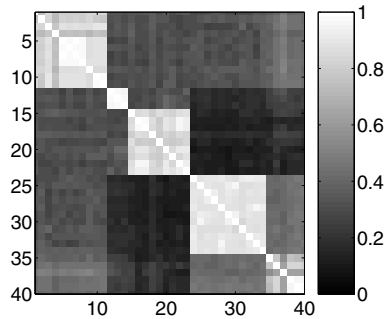


Fig. 3. Pairwise similarity matrix of the Skolnick clustering test. The color bars show the gray-scaled shading for similarities in the range $[0, 1]$. Brighter shadings indicate larger similarities and vice versa. We can identify five clusters.

Table 4. Descriptions of the clusters of proteins from the the Skolnick test. Fold, family, and superfamily are according to SCOP.

Cluster	Domains	Fold	Superfamily	Family
1	1-8, 12-14	Flavodin-like	Che Y-like	Che Y-related
2	9-11	Microbial ribonucleases	Microbial ribonucleases	Fungi ribonucleases
3	15-23	Cupredoxin-like	Cupredoxins	Plastocyanin-like Plastoazurin-like
4	24-34	TIM-beta alpha-barrel	Triosephosphate isomerase (TIM)	Triosephosphate isomerase (TIM)
5	35-40	Ferritin-like	Ferritin-like	Ferritin

For 780 pairwise structural alignments, SADP required about 12 minutes. Figure 3 summarizes the results. The results indicate an agreement with the SCOP categories as shown in Table 4. Hence, SADP has sufficient discriminative power to solve the Skolnick clustering test within a relatively short period of time.

5 Conclusion

We proposed a continuous solution to the CMO problem combining softassign with dynamic programming linked together via a problem dependent compatibility function. Experiments show that the proposed method is extremely fast and well suited for similar protein structures, and therefore useful for solving clustering tasks.

Since we have not thoroughly tested alternative compatibility functions, one future research direction aims at constructing compatibility functions that improve the performance of SADP, in particular for dissimilar pairs of contact maps.

References

1. A. Caprara and G. Lancia. Structural alignment of large-size proteins via Lagrangian relaxation. In *RECOMB*, pages 100–108, 2002.
2. A. Caprara, R. Carr, S. Istrail, G. Lancia, and Brian Walenz. 1001 optimal pdb structure alignments: Integer programming methods for finding the maximum contact map overlap. *Journal of Computational Biology*, 11(1):27-52, 2004.
3. R. Carr, W. Hart, N. Krasnogor, J. Hirst, E. K. Burke, and J. Smith. Alignment of protein structures with a memetic evolutionary algorithm. In *GECCO 02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1027-1034, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.
4. A. M. Finch, R. C. Wilson, and E. R. Hancock. An energy function and continuous edit process for graph matching. *Neural Computation*, 10(7):1873-1894, 1998.
5. A. Godzik, A. Kolinski, and J. Skolnick. Topology fingerprint approach to the inverse protein folding problem. *Journal of Molecular Biology*, 5(1):227-238, 1992.
6. A. Godzik and J. Skolnick. Flexible algorithm for direct multiple alignment of protein structures and sequences. *Computer Applications in the Biosciences*, 10(6):587-596, 1994.
7. S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377-388, 1996.
8. D. Goldman. *Algorithmic Aspects of Protein Folding and Protein Structure Similarity*. PhD thesis, University of California, Berkeley, 2000.
9. D. Goldman, S. Istrail, and C. H. Papadimitriou. Algorithmic aspects of protein structure similarity. In *40th Annual Symposium on Foundations of Computer Science*, pages 512-521. IEEE Computer Society, 1999.
10. L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233:123-138, 1993.
11. S. Ishii and M. Sato. Doubly constrained network for combinatorial optimization. *Neurocomputing*, 43:239-257, 2002.
12. W. Kabash. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica*, A32(5):922-923, 1978.
13. N. Krasnogor. Self generating metaheuristics in bioinformatics: The proteins structure comparison case. *Genetic Programming and Evolvable Machines*, 5(2):181-201, 2004.
14. G. Lancia, R. Carr, B. Walenz, and S. Istrail. 101 optimal pdb structure alignments: a branch-and-cut algorithm for the maximum contact map overlap problem. In *RECOMB 01: Proceedings of the fifth annual international conference on Computational biology*, pages 193-202, New York, NY, USA, 2001. ACM Press.
15. M.A. Lozano and F. Escolano. A significant improvement of softassign with diffusion kernels. In A. Fred, Terry Caelli, R. P. W. Duin, A. Campilho, and D. de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops, SSPR 2004 and SPR 2004*, volume 3138 of *Lecture Notes in Computer Science*, pages 76-84, 2004.
16. R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Annals of Mathematical Statistics*, 35(2):876-879, 1964.

17. D. M. Strickl and, E. Barnes, and J. S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, 53(3):389-402, 2005.
18. W. Xie and N. V. Sahinidis. A branch-and-reduce algorithm for the contact map overlap problem. In A. Apostolico, C. Guerra, S. Istrail, P. Pevzner, and M. Waterman, *editors*, *RECOMB*, volume 3909 of *Lecture Notes in Computer Science*, pages 516-529. Springer Berlin / Heidelberg, 2006.