ABSTRACT

Motivation: Accurate prediction of contacts between \( \beta \)-strand residues can significantly contribute towards \( ab \) \( initio \) prediction of the 3D structure of many proteins. Contacts in the same protein are highly interdependent. Therefore, significant improvements can be expected by applying statistical relational learners that overcome the usual machine learning assumption that examples are independent and identically distributed. Furthermore, the dependencies among \( \beta \)-residue contacts are subject to strong regularities, many of which are known a priori. In this article, we take advantage of Markov logic, a statistical relational learning framework that is able to capture dependencies between contacts, and constrain the solution according to domain knowledge expressed by means of weighted rules in a logical language.

Results: We introduce a novel hybrid architecture based on neural and Markov logic networks with grounding-specific weights. On a non-redundant dataset, our method achieves 44.9% \( \beta \)-residue measure, with 47.3% precision and 42.7% recall, which is significantly better \((P < 0.01)\) than previously reported performance obtained by 2D recursive neural networks. Our approach also significantly improves the number of chains for which \( \beta \)-strands are nearly perfectly paired \((36\% \text{ of the chains are predicted with } F > 70\% \text{ on coarse map})\). It also outperforms more general contact predictors on recent CASP 2008 targets.

Contact: lippi@dsi.unifi.it

Supplementary information: Supplementary data are available at Bioinformatics online.

1 INTRODUCTION

The \( \beta \) structure is one of the most common secondary structures in proteins. It occurs in almost 85\% of experimentally determined proteins and \( \sim \) 15\% of these entirely consist of \( \beta \) structures. In \( \beta \)-sheets, two or more extended strands of the macro-molecule are arranged in a flat conformation. Strands can either run in the same (parallel) or in the opposite (antiparallel) direction. Each \( \beta \)-residue is typically hydrogen bonded to one or two other \( \beta \)-residues in different strands of the same chain. Occasionally, irregularities are found, most notably isolated \( \beta \)-residues (bulges), \( \beta \)-turns or bonds between residues belonging to different chains or proteins. Strands that are adjacent in the folded protein can be largely separated in sequence, making the prediction problem difficult because of long-ranged interactions. Indeed, a large fraction of distant contacts in contact maps involve two \( \beta \)-residues (disulfide bridges and metal binding sites being other important cases).

Our definition of contacts is based on hydrogen bonds, as determined by the DSSP program (Kabsch and Sander, 1983): two \( \beta \)-residues \( a \) and \( b \) are in contact if and only if \( b \) is listed as one of the \( \beta \)-bridge partner (fields BP1 and BP2 in the DSSP output). In this case, we say that \( a \) and \( b \) are \( \beta \)-partners. The problem addressed in this article is to predict this relation between any two \( \beta \)-residues. Accurate predictions can significantly contribute towards \( ab \) \( initio \) prediction of protein structure (Ruczinski et al., 2002), protein design (Floudas et al., 2006) and characterizing supersecondary structure and protein folding patterns (Kamat and Lesk, 2007).

One of the first solutions to this problem was developed by (Baldi et al., 2000), who started from amino acid sequence alone and trained feedforward neural networks as binary classifiers on residue pairs, with inputs consisting of two windows of residues, each centered around one of the target residues. There are two fundamental limitations when casting link prediction into a binary classification problem on pairs. First, the resulting dataset will be highly imbalanced: Baldi et al. (2000) reported 37 000 positive examples and 44 million negative examples for a dataset of 826 chains. Second, contacts are not independent (e.g. two residues are more likely to be partners, if so are the adjacent residues in the protein sequence) and ignoring this linkage can lead to higher bias in the generalization error. Cheng and Baldi (2005) assumed both the amino acid and the secondary structure sequences to be known; they employed a different architecture called 2D-recursive neural network (2D-RNN). A 2D-RNN has the structure of a 2D grid and is trained with binary targets that correspond to the adjacency matrix of the \( \beta \)-partnership graph. The main difference with respect to feedforward networks is that prediction on positions pair \( (a,b) \) not only depends on residues near those positions but also on the hidden activities of adjacent cells in the grid. 2D-RNN can be seen as a clever compromise between propositional and relational learners. Targets are presented collectively to the network, but inference is very efficient compared with belief networks because predictions in each cell are computed independently by forward propagating inputs in topological order. 2D-RNNs, however, suffer a well-known problem of vanishing gradients, which limits dependencies only to nearby pairs in the grid. Furthermore, they are unable to incorporate constraints on the predicted \( \beta \)-partnership graph.
For example, residues should have no more than two partners, there should be no mixed parallel and antiparallel bonds, and the symmetry of the contact relation should be guaranteed. To overcome these limitations of 2D-RNNs, Cheng and Baldi (2005) added a non-adaptive post-processor that collectively reassigns \( \beta \)-contacts by means of an efficient graph matching algorithm enforcing some physical constraints derived from background knowledge. The resulting BetaPro predictor (Cheng and Baldi, 2005) reaches state-of-the-art performance. We argue, however, that the accuracy for this problem can be further improved by combining two novel methodological ideas for this problem: using relational learners capable of collective assignment of \( \beta \)-contacts, and enforcing knowledge-based constraints during the learning process, rather than a posteriori.

In this article, we therefore formulate \( \beta \)-residue contact prediction as a link prediction task in the context of statistical relational learning (Getoor and Taskar, 2007). As in Cheng and Baldi (2005), we assume that secondary structure is given. We present a solution based on Markov logic (Domingos et al., 2008; Richardson and Domingos, 2006), a probabilistic model where first-order logic allows us to express declarative domain knowledge and to specify the model structure. Logical formulae are used to enforce constraints that \( \beta \)-sheets should obey. For example, a residue cannot have more than two partners, no more than one partner in a given strand, and residue contacts between any two paired strands must either follow the parallel or the antiparallel pattern, but avoiding crossing edges. A major difference between traditional logic and Markov logic is that uncertainty is naturally dealt with in the latter, by attaching real-valued weights to formulae, which express their strength. In this way, it becomes possible to specify ‘soft’ rules that are likely to hold in the domain, but subject to exceptions. For example, two strands that are separated by a short distance in sequence, especially when a glycine occurs between them, are likely arranged in a parallel or the antiparallel pattern and linked in an antiparallel fashion. Similarly, the first two residues of parallel strands that have the same length are likely to be partners.

2 METHODS

2.1 Background and notation

2.1.1 Essential first-order logic terminology A logic is a formal system of statements, endowed with syntax and semantics, together with mechanisms for asserting and deducting the truth of sentences. Different logics can be introduced by making different ontological assumptions and by restricting the set of well-formed sentences. In first-order logic, we postulate that there only exist objects, functions mapping tuples of objects to objects and relations on tuples of objects. Syntactically, an object can be denoted by either a constant (that we may regard as the object identifier) or a variable, i.e. a placeholder that can be bound to some constant. We denote by \( C \) the set of constants in the domain of interest. A term is recursively defined as consisting of either a constant, a variable or a function applied to a tuple of terms. A term is said to be ground, if it contains no variables. A predicate is a truth statement about a tuple of objects. Unary predicates describe properties of their only argument [e.g. Hydrophobic(Pdb_1pty_res22)]. Predicates with higher arity describe relations among the object in a tuple [e.g. Partner(Pdb_1pty_res22,Pdb_1pty_res34)]. A predicate applied to a tuple of terms is called an atom. A formula is recursively defined as consisting of either an atom or a combination of formulae combined by means of connectives (disjunction \( \lor \), conjunction \( \land \), negation \( \neg \) and implication \( \rightarrow \)) and/or quantifiers. If \( F \) is a formula containing the variable \( x \), the existentially [universally] quantified formula \( \exists F \) [\( \forall F \)] is a shorthand for the disjunction [conjunction] of all formulae obtained by replacing \( x \) with a domain constant.

A formula \( F \) whose variables have all been replaced by ground terms will be referred to as a grounding of \( F \). In this article, we will always associate a type to each object, where a type \( \tau \) is defined to be a set of constants, \( \tau \subseteq C \), and a type system is a partition of \( C \). By restricting the arguments to functions and predicates to specific types, we avoid the construction of groundings that are semantically non-meaningful [like, e.g. Hydrophobic(Pdb_1pty_res22)]. We can infer the type of a variable in a formula by observing its position as an argument to a function or a predicate. Logical inference in first-order logic is the problem of determining if a knowledge base \( \mathcal{F} \) entails a given formula \( F \), denoted \( \mathcal{F} \models F \), which means that \( F \) is true in every world where all formulae in \( \mathcal{F} \) are true.

2.1.2 Markov networks Given a set of random variables \( U \), a Markov network (or Markov random field) is an undirected graph whose node set is \( U \) and whose edge set encodes conditional independence assertions (Pearl, 1988). The semantic of the graph is defined as follows: for any \( X, Y, Z \subseteq U \), \( X \) is conditionally independent of \( Y \) given \( Z \) if every path from a node in \( X \) to a node in \( Y \) intersects \( Z \). The Hammersley and Clifford (1971) theorem states that any joint probability distribution over \( U \) that satisfies the conditional independencies implied by the graph, is decomposable as

\[
P(U = u) = \prod_C \phi_C(u_C)
\]

where \( C \) is a clique of the network, \( \phi_C \) is a real-valued function associated with the clique, \( u_C \) are the realizations of the variables \( C \) when \( U = u \) and \( \phi_C \) is a normalization factor also known as partition function. Probabilistic inference in graphical models is the problem of evaluating conditional probabilities \( P(Y = y | X = x) \) where \( X \) is a set of random variables receiving evidence (i.e. whose configuration is assigned to \( x \)) and \( Y \) a set of query random variables.

2.1.3 Standard Markov logic networks A standard Markov logic network [MLN; see Domingos et al. (2008) for a detailed review] consists of a knowledge base \( \mathcal{F} = \{ F_1, \ldots, F_n \} \), a finite set of constants \( C \) associated with domain objects, and a set of real-valued weights \( w = \{w_1, \ldots, w_n\} \) expressing the ‘strengths’ of the formulae in \( \mathcal{F} \). An MLN can also be interpreted as a Markov network whose structure derives from \( \mathcal{F} \) and \( C \) and parameterized by \( w \). Specifically, the node set of the network consists of all possible ground atoms, and there is an edge between two nodes if the corresponding ground atoms appear together in at least one grounding of some formula \( F_i \).

An MLN defines a probability distribution over possible worlds (i.e. truth assignments to ground atoms). Random variables are the truth state of ground atoms (the distinction between random variables and logical variables should be clear from the context in the following). Inference in MLN subsumes the above concepts of logical inference in first-order logic and probabilistic inference in probabilistic graphical models. In particular, logical inference (entailment) is recovered when all formulae have infinite weights. If the weight of a formula is finite, then every world where the formula is false is less probable (the probability depending on the magnitude of the weight) but not impossible. MLNs can be either trained in generative or discriminant mode. In this article, we focus on discriminant learning and assume that every ground atom is either an evidence atom (representing predictive information) or a query atom (representing a prediction). In this setting, an MLN is a model for the conditional distribution of the set of query atoms \( Y \) given the set of evidence atoms \( X \). Equation (1) can be conveniently expressed as a log-linear model:

\[
P(Y = y | X = x) = \frac{1}{Z_x} \sum_{\phi \in \mathcal{F}} \exp(w_i \phi(x))
\]
An alternative (and commonly used) MLN modeling choice is to avoid leading, after expansion, to $20 \cdot 2$ introducing the predicate appearing in a separate formula. In our example, this could be obtained by parameters makes this approach unsuitable for the present application.

where the variables $a$ and $b$ have type Residue (i.e. take values on the set of 20 amino acid letters), the variables $a$ and $b$ have type Residue (i.e. take values on the set of residue identifiers), Window($a_4$, $a_5$) is true if and only if variables $a_4$ and $a_5$ are bound to the amino acid constants that correspond to those found in the window of size $2k+1$ centered around $a$, and $\text{Partners}(a,b)$ is true if and only if residues $a$ and $b$ are $\beta$-partners. Attaching a single weight to Formula (3) would produce a too simple model (the corresponding MLN would merely replace the exponential growth of the number of formulae with amino acid constants $\geq k+1$ that would be expanded into $20 \cdot 4$). Then $\text{IR}_{hi} = \sum_{i=1}^{k} g_{ij}$.

In this formulation, depending on how $a$ is parameterized, the number of free parameters can be small even if the number of possible distinct configurations $g_{ij}$ grows exponentially. Weight reparametrization has a second interesting advantage: it allows us to let the formula weight to depend on any predictive attribute we may know to be correlated with the strength of the formula, including numerical attributes describing properties of the selected variables, such as conservation in multiple alignment profiles.

$\omega$ can be implemented in many ways, for example, using kernel logistic regression (KLR; Wahba, 1999) or neural networks with parameters $\theta$. In the case of KLR, $a(x) = \phi(x)$, where $\phi$ is the feature mapping induced by the kernel. For example, one might use a kernel between configurations of sets of logical variables (Gärtner et al., 2004). The results reported in this article were obtained using neural networks. In this case, $a(x) = \phi(x)$ and defined as follows. Let $q_{ij}$ be the number of selected variables in formula $F_i$ having type $x = \sum_{i=1}^{k} c_{i1}$, $c_{i2} = 1$. Then $\text{IR}_{hi} = \sum_{i=1}^{k} g_{ij}$.

Unlike kernel machines, learning with neural networks requires us to solve non-convex optimization problems, leading potentially to suboptimal solutions. The benefits of convex optimization are less strong in the present context since even probabilistic inference is $\#P$-complete (Roth, 1996) and requires approximate solutions (see below). On the other hand, the computation of $a(x)$ with a kernel machine would require time proportional to the number of cases, e.g. the number of candidate partners when applied to Formula (7), while using neural networks we can compute it in constant time.

2.2.2 Grounding-specific weights We propose here to reparametrize a MLN by letting weights to depend on the specific grounding of a subset of the variables appearing in a formula, called the selected variables. In this kind of MLN, Equation (2) is extended as follows:

$$F(x; \theta) = \prod_{i=1}^{n} \prod_{j=1}^{m} \omega_{ij}^{(s)}(x; \theta)$$

where $g_{ij}$ denotes the $j$-th ground configuration of the selected variables in the $i$-th formula, $\omega_{ij}$ a real-valued parameterized function returning the weight attached to each ground formula after the selected variables have been bound to the constants in $g_{ij}$ and $a_{ij}(x,y)$ is the number of true groundings in $(x,y)$ matching $g_{ij}$. In general, $\sigma_i$ is a parameter vector. The special case where $\sigma_i$ is a scalar and $\omega_{ij} = \sigma_i$ recovers standard MLN, so Equation (6) subsumes Equation (2). In our notation, selected variables in a formula are prefixed by the $\dagger$ sign. For example, a grounding-specific version of Formula (3) is:

$$\text{Window}(a_4, \ldots, a_8) \land \text{Window}(b_2, \ldots, b_8) \land \text{Partners}(a,b).$$

In this formulation, depending on how $\omega_{ij}$ is parameterized, the number of free parameters can be small even if the number of possible distinct configurations $g_{ij}$ grows exponentially. Weight reparametrization has a second interesting advantage: it allows us to let the formula weight to depend on any predictive attribute we may know to be correlated with the strength of the formula, including numerical attributes describing properties of the selected variables, such as conservation in multiple alignment profiles.
As discussed above, MAP inference is much faster and more effective in MLN, as shown by Parallel et al. by at least one $\beta$ query predicate. Predicate $(9)$ is used to capture the alignment of strands adjacent to a residue. We found that this is impractical for two reasons: exact inference is intractable, and the expectations require us to compute conditional probabilities of query atoms given evidence atoms, rather than finding the most probable truth assignment. Therefore, the evidence predicate is always true, as is the MAP grounding of (9). The evidence predicate is used to capture the alignment of strands adjacent to a residue.

In general, MLN can be applied to fully connected relational domains (i.e. where each object is linked to every other object via a path of relations). There are three important advantages that can be gained under this assumption. First, we can partition the set of domain constants $C$ into disjoint subsets (associated with individual chains) and run inference separately on each of them. In this way, there is no need to try to satisfy predicates taking arguments from different chains [e.g. $\text{Partner}(a,b)$ when $a$ and $b$ belong to different chains]. Second, gradients can be computed for a chain at a time, enabling the application of stochastic gradient descent where the weights of the MN and the embedded neural networks are updated after the presentation of a mini-batch of groundings corresponding to an individual chain. This approach yields considerable training speedups. Third, inference can be easily parallelized on multiple CPUs.

In order to compute true gradients, inference should compute the exact values of $E(y_i|x_i,a)$ for each formula $F_i$. This is impractical for two reasons: exact inference is intractable, and the expectations require us to compute conditional probabilities of query atoms given evidence atoms, rather than finding the most probable truth assignment, $y^*_i$, to query atoms. As discussed above, MAP inference is much faster and more effective in MLN, but using MaxWalkSAT we only obtain approximate gradients, as follows:

$$\frac{\partial \ell}{\partial a} = \frac{\partial \ell}{\partial a} = n(a|x_i,y^*_i) - E[n(a|x_i)]$$

In particular, non-zero contributions to the gradients are only collected when inferred groundings disagree with the target groundings that play the role of labels in the training set (and the contribution is $+1$ if the target is true and the MAP grounding is false, and $-1$ otherwise). When considering formulae like (7), whose weights are computed by neural networks, this setting has an interesting interpretation in terms of active learning. At each iteration, each network is only trained from atoms that MAP inference fails to assign to the correct truth value.

2.3 Knowledge base design

We developed a set of 54 formulae expressing domain knowledge derived from the scientific literature and inspection of the available data. The two main object types are $\text{Residue}$ and $\text{Strand}$. Strands are defined as subsstrings of residues that are consecutively assigned to secondary structure class ‘E’ or ‘B’ by the DSSP program (Kabsch and Sander, 1983). In the following, we will use symbols $a$, $b$, etc., for variables of type $\text{Residue}$ and $r$, $s$, etc., for variables of type $\text{Strand}$. Features associated with residues and strands are real vectors and are used in conjunction with using grounding-specific weights as, detailed in Section 2.2. We designed four query predicates. Predicate $\text{Partner}$ is true if and only if two residues are $\beta$-partners, as assigned by the DSSP program. Predicate $\text{Coarse}$ describes secondary structure and is true if and only if two strands are linked by at least one $\beta$-partners pair [we included this predicate since coarse contact maps can be predicted from sequence information (Pollastri et al., 2006)]. Finally, we included predicates $\text{Parallel}$ and $\text{AntiParallel}$. Two strands are considered to be parallel if connected by at least one pair $(a,b)$ of $\beta$-partners and no other pair $(a',b')$ connects them with $a$ preceded by $a'$ and $b'$ preceded by $b$ in sequence. The knowledge base is divided in seven sections and briefly summarized below (an input file for a modified version of the Alchemy system (Kok et al., 2007) is given as Supplementary Material). In Markov logic, we can flag some rules to be soft, as opposite to hard, meaning that the associated weight is very large. All formulae in these sections were flagged as hard, except (11–23). Figure 1 gives a graphical representation of some of the rules.

2.1 Characterization of query predicates

Fine- and coarse-grained $\beta$-contacts are antireflexive and symmetric. Hence, we have, e.g. $\text{Partner}(a,a) \rightarrow \text{Partner}(b,a)$ (and similarly for the remaining three query predicates). A coarse contact is either parallel or antiparallel: $\text{Coarse}(r,s)$ (where $r$ and $s$ belong to the same strand). Two residues belonging to the same strand cannot be partners: $\text{Partner}(a,a)$.

No residue can have two partners that belong to the same strand: $\text{Partner}(a,a) \rightarrow \text{Partner}(b,b)$. (9)

2.2 Adjacency rules

We found it very important to constrain predictions to avoid crossing edges, which are rather common mistakes of neural network-based predictors. For example:

$$\text{Partner}(a,b) \rightarrow \text{Partner}(a+1,b+1)$$

We also included formulae that capture the alignment of strands adjacent to each other. For example, if two residues are partners and belong to parallel strands, their successors are also partners:

$$\text{Parallel}(r,s) \rightarrow \text{Partner}(a,b) \rightarrow \text{Partner}(a+1,b+1)$$

Note that this is only a soft rule (i.e. not always satisfied) and in Markov logic it has the effect of encouraging adjacent arrangements of $\beta$-partners, propagating predicted links from left to right. A similar formula that uses $a+1$ and $a-1$ are shorthands for the successor and the predecessor of $a$, respectively. This is occasionally an abuse of notation (e.g. $a-1$ does not exist when $a$ ends a strand), but makes formulae more readable. Formulae actually used in the experiments are listed in the Supplementary Materials.
predicates is used to propagate links from right to left. Bulges are allowed as follows:

\[ \text{Parallel}(r,a) \land \text{Partner}(a,b) \land \text{Overlap}(b,n) \rightarrow \text{Partner}(a+b+1) \land \text{Partner}(a+b+1). \]  

(12)

Analogous formulae cover the case of antiparallel strands.

2.3.3 \( \beta \)-hairpins For modeling this supersecondary structure, we considered only strands separated by less than six residues, since almost 70% of \( \beta \)-hairpins have such a short loop (Gusnaceskar et al., 1997). In our dataset, only 39% of such strand pairs are antiparallel, but the ratio grows to 77% if we constrain the length of the strands to be at least of four residues and to 83% if one of the loop residues is a glycine. Hence, we introduced the following formula:

\[ \text{Hairpin}(r,a) \rightarrow \text{Partner}(a+b). \]  

(13)

where \( \text{Hairpin}(r,a) \) is true if there is a glycine residue between strands \( r \) and \( a \), \( \text{Partner} \) returns the sequence separation between two strands and \( |r| \) is a shorthand for the function returning the length of strand \( r \). The \( \text{Hairpin} \) predicate can be used to infer both coarse- and fine-grained contacts:

\[ \text{Hairpin}(r,a) \rightarrow \text{AntiParallel}(r,a). \]  

(14)

\[ \text{Hairpin}(r,a) \land \text{Partner}(a+b) \rightarrow \text{Partner}(a+b+1). \]  

(15)

2.3.4 Rules involving helices In the \( \beta \)-hair configuration, two strands which are separated by a helix are very often arranged in a parallel contact (in our dataset, only 7% are antiparallel). Conversely, helices do separate most (89% in our dataset) parallel consecutive strands. Note however that the presence of a helix is not sufficient to predict a contact (only 38% helix-separated strands are parallel). The condition is, therefore, strengthened by constraining the strand lengths to be greater than four residues, improving the confidence to 47%. These analyses lead to the following formulae:

\[ \text{HelixBetween}(r,a) \land |r| |a| |r| |a| |r| |a| \rightarrow \text{Parallel}(r,a). \]  

(16)

\[ \text{Hairpin}(r,a) \land |r| |a| |r| |a| \rightarrow \text{Parallel}(r,a). \]  

(17)

where \( \text{HelixBetween}(r,a) \) is true when \( r \) and \( a \) are two consecutive strands separated by a helix.

2.3.5 Rules involving strand endpoints In our preliminary experiments, we found that correct prediction of \( \beta \)-partners involving strand endpoints (first or last residue in a strand) strongly correlate with the overall prediction accuracy. We therefore designed \( \text{near} \) rules to encourage correct assignment of partners at the endpoints. For example, in two parallel strands, either their first or last residues are partners:

\[ \text{Parallel}(r,a) \land \text{First}(a,r) \land \text{Last}(b,a) \land \text{Last}(b,a) \rightarrow \text{Partner}(a,b) \land \text{Partner}(b,a). \]  

(18)

(a similar rule is used for antiparallel strands). If two strands have the same length, then both endpoints should be partners. For example, in the parallel case:

\[ \text{Parallel}(r,a) \land |r| = |a| \land \text{First}(a,r) \land \text{First}(b,a) \rightarrow \text{Partner}(a,b). \]  

(19)

\[ \text{Parallel}(r,a) \land |r| = |a| \land \text{Last}(a,r) \land \text{Last}(b,a) \rightarrow \text{Partner}(a,b). \]  

(20)

2.3.6 Rules embedding discriminative classifiers The following formulae form the core part of the model and implement discriminative classifiers using grounding-specific weights, as detailed in Section 2.2. They have query predicates on their right-hand side: \( \text{Partner}, \text{Parallel} \) and \( \text{AntiParallel} \). A separate neural network is associated with each formula.

\[ \text{ResidueFeatures}(r,a) \land \text{ResidueFeatures}(b,b) \rightarrow \text{AntiParallel}(r,a). \]  

(21)

\[ \text{PairFeatures}(r,a) \land \text{SequenceFeatures}(r) \rightarrow \text{Partner}(r,a) \land \text{Partner}(a,b). \]  

(22)

\[ \text{StrandFeatures}(r,a) \land \text{StrandFeatures}(s,s) \rightarrow \text{Parallel}(r,a). \]  

(23)

Each ‘feature’ predicate above is true if and only if the \( S \)-prefixed argument is expanded into the feature vector characterizing the object appearing as the first argument (predicate \( \text{SequenceFeatures} \) only takes one argument since, as explained in Section 2.2, we run inference and learning on a chain at a time, thus there is always exactly one chain in the domain of the formula). Feature vectors are described in the next subsection.

2.4 Input features Residue features are computed on a window of nine flanking residues: (i) a \( 180D \) vector of multiple alignment profiles; (ii) a \( 27D \) vector describing secondary structure in three classes (helices, strands, coil); (iii) a \( 9D \) vector describing the solvent accessibility in two states (buried or exposed at 25% threshold). Multiple alignment profiles were computed by running one iteration of PSI-BLAST (Altschul et al., 1997) on the non-redundant (nr) NCBI protein database posted on June 19, 2003.

Residue pairs features consist of: (i) sequence separation between the two residues, divided by 200; (ii) number of \( \beta \)-strands between the two residues, divided by 10; (iii) a \( 20D \) vector of amino acid composition in the substring between the two residues; (iv) a \( 3D \) vector of frequencies of secondary structure assignments in the substring between the two residues; (v) the average solvent accessibility between the two residues; (vi) a \( 10D \) vector encoding unordered pairwise combinations of physicochemical properties of the two amino acids: non-polar (A,F,G,I,L,M,P,V,W), polar (C,N,Q,S,T,Y), acidic (D,E), basic (H,K,R), as in Cheng and Baldi (2007); and (vii) a \( 125D \) vector containing sequence encoding, secondary structure and solvent accessibility of a window of diameter five centered on position \((i+j)/2\).

Sequence features consist of: (i) a \( 20D \) vector of amino acid composition; (ii) a \( 3D \) vector of frequencies of secondary structure assignments; (iii) the average solvent accessibility; (iv) chain length, described by a unary-encoding \( 4D \) vector as in Ponta and Rost (2005).

Residue pairs features consist of: (i) a \( 10D \) vector encoding the strand position within the sequence: after dividing the sequence in 10 portion of equal length, the \( j \)th component of this vector is \( 1 \) or \( 0 \) depending on whether the strand overlaps that portion; (ii) a \( 20D \) vector of amino acid composition of the strand; (iii) average solvent accessibility of the strand; and (iv) a \( 12D \) vector containing a unary encoding of strand length.

Inter-strand features consist of: (i) a \( 20D \) vector of amino acid composition in the substring between the two strands; (ii) a \( 3D \) vector of frequencies of secondary structure assignments in the substring between the two strands; (iii) the average solvent accessibility between the two strands; (iv) chain length, described by a unary-encoding \( 4D \) vector.

2.5 Iterative relabeling We found it useful to feed predicted \( \beta \)-partners atoms back as evidence to a second-stage refinement MLN. To this end, we introduce a new evidence predicate, \( \text{Candidate} \), which is true for residues \((a,b)\) if and only if \( a \) and \( b \) were predicted to be partners in the first stage. The second stage MLN
The method was validated on the Cheng and Baldi (2005) dataset, several times, although in practice we observed no significant performance in inference running time. In principle, the relabeling process can be iterated atom: when false, the formula would be satisfied, thus not contributing to (using the relabeling trick, instead, the antecedent is given as an evidence into the Markov logic architecture. We run MaxWalkSAT separately networks were pretrained for a few epochs before incorporating them protein chain. In order to speed up the learning process, neural learning (voted perceptron) and stochastic gradient descent for modified version of the Alchemy system is based on discriminative splits of the experiments reported in Cheng and Baldi (2005). Our comparison, profiles were derived from the June 2003 release of β-strands, which participate in 8172 predictions of the two systems. First of all, BetaPro does not precision–recall breakeven threshold. More precisely, MAP inference flips neural network prediction in 60586 cases; in 46376 cases (76.5%), MAP inference takes the correct decision, 42824 times by flipping from positive to negative class, and only 3552 times from negative to positive one. The errors induced by MAP inference are instead almost balanced between the two classes: in 6071 cases inference wrongly predicts the negative class, while in 8139 the opposite happens. In the attempt to further improve performance by leveraging these differences, we designed a simple combination of the two methods, called MLN-2S in the following. After training the first stage of BetaPro, we collected the predicted conditional probabilities \( \hat{p}_{ij} \) of two residues \( i \) and \( j \) being partners, and rescaled them to force a precision–recall break-even threshold \( \bar{\gamma} = 0.5 \), consistently with the BetaPro calibration. We then introduced a new formula similar to (7), using grounding-specific weights \( \bar{w}_i = \log(p_{ij}/(1-p_{ij})) \). As shown in Tables 1 and 2, this combination of the two predictors improves recall and overall performance. There are other interesting qualitative differences in the predictions of the two systems. First of all, BetaPro does not constrain the number of partners a residue can have, while our method penalizes the physically unfeasible configurations in which a residue is linked to more than two partners: Table 3 shows that in over 60% of the chains, BetaPro presents such an inconsistency. This can be crucial when the unfeasible configurations affect many strands: for example, in chain 1B7GO BetaPro predicts three strands having residues with more than two partners, with a resulting \( F_1 \) on coarse map equal to 48.3%; Markov logic constraints, in this case, respectively. The MLN outperforms state-of-the-art BetaPro \( F_1 \) both at the residue level and at the strand level. In both cases, differences are statistically significant (\( P < 0.01 \)).

The two methods generate different predictions in many cases, mainly because several false positive links are removed by enforcing logical rules. This results in a much higher precision, at the expense of lower recall. More precisely, MAP inference flips neural network prediction in 60586 cases; in 46376 cases (76.5%), MAP inference takes the correct decision, 42824 times by flipping from positive to negative class, and only 3552 times from negative to positive one. The errors induced by MAP inference are instead almost balanced between the two classes: in 6071 cases inference wrongly predicts the negative class, while in 8139 the opposite happens. In the attempt to further improve performance by leveraging these differences, we designed a simple combination of the two methods, called MLN-2S in the following. After training the first stage of BetaPro, we collected the predicted conditional probabilities \( \hat{p}_{ij} \) of two residues \( i \) and \( j \) being partners, and rescaled them to force a precision–recall break-even threshold \( \bar{\gamma} = 0.5 \), consistently with the BetaPro calibration. We then introduced a new formula similar to (7), using grounding-specific weights \( \bar{w}_i = \log(p_{ij}/(1-p_{ij})) \). As shown in Tables 1 and 2, this combination of the two predictors improves recall and overall performance. There are other interesting qualitative differences in the predictions of the two systems. First of all, BetaPro does not constrain the number of partners a residue can have, while our method penalizes the physically unfeasible configurations in which a residue is linked to more than two partners: Table 3 shows that in over 60% of the chains, BetaPro presents such an inconsistency. This can be crucial when the unfeasible configurations affect many strands: for example, in chain 1B7GO BetaPro predicts three strands having residues with more than two partners, with a resulting \( F_1 \) on coarse map equal to 48.3%; Markov logic constraints, in this case, respectively. The MLN outperforms state-of-the-art BetaPro \( F_1 \) both at the residue level and at the strand level. In both cases, differences are statistically significant (\( P < 0.01 \)).

The two methods generate different predictions in many cases, mainly because several false positive links are removed by enforcing logical rules. This results in a much higher precision, at the expense of lower recall. More precisely, MAP inference flips neural network prediction in 60586 cases; in 46376 cases (76.5%), MAP inference takes the correct decision, 42824 times by flipping from positive to negative class, and only 3552 times from negative to positive one. The errors induced by MAP inference are instead almost balanced between the two classes: in 6071 cases inference wrongly predicts the negative class, while in 8139 the opposite happens.
Table 3. Qualitative comparison on Cheng and Baldi dataset

<table>
<thead>
<tr>
<th></th>
<th>BetaPro</th>
<th>MLN</th>
<th>MLN-2S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Residues with ≥3 partners</td>
<td>7.0</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Chains having residues with ≥3 partners</td>
<td>61.7</td>
<td>2.8</td>
<td>3.3</td>
</tr>
<tr>
<td>Chains having F₁ on coarse map ≥70%</td>
<td>31.7</td>
<td>33.7</td>
<td>36.2</td>
</tr>
<tr>
<td>F₁ on parallel strands</td>
<td>50.2</td>
<td>49.5</td>
<td>52.8</td>
</tr>
<tr>
<td>F₁ on antiparallel strands</td>
<td>57.0</td>
<td>59.9</td>
<td>60.9</td>
</tr>
<tr>
<td>Correct pairing directions over correctly predicted</td>
<td>93.0</td>
<td>95.6</td>
<td>95.6</td>
</tr>
<tr>
<td>F₁ on β-bulges in coarse contacts</td>
<td>–</td>
<td>19.7</td>
<td>23.9</td>
</tr>
<tr>
<td>β-bulges correctly identified within one residue</td>
<td>–</td>
<td>76.1</td>
<td>78.3</td>
</tr>
<tr>
<td>Correct β-sheet connected components</td>
<td>17.1</td>
<td>24.3</td>
<td>24.8</td>
</tr>
<tr>
<td>β-hairpins</td>
<td>78.2</td>
<td>80.0</td>
<td>80.4</td>
</tr>
</tbody>
</table>

Fig. 3. F₁ distribution on coarse maps for the three predictors on the Cheng and Baldi dataset.

evident for chains containing at least two α-helices (79% of the whole dataset). For these chains, MLN-2S is more accurate in 62% of the cases. For chains with one or no helices, MLN-2S is more accurate in 53% and 55% of the cases, respectively.

The advantages of Markov logic are also evident if we consider performance measures that take into account relational dependencies between β-partners (Table 3). For example, BetaPro does not allow the presence of β-bulges, while our method is able to identify their existence (F₁ = 23.9%) and, when it does, to correctly identify their position within one residue 78.3% of the times. Pairing directions of strands are also more accurate (2.6 and 3.9 improvements of F₁ on parallel and antiparallel strands, respectively), as well as β-hairpins (2.2 F₁ improvement). A measure which considers the overall β-sheet topology of a chain is the accuracy on β-sheet connected components, defined as the percentage of correctly identified connected components of the coarse contact graph (7.7% improvement).

3.2 CASP 2008 dataset

The second dataset used in our experiments consists of a selection of CASP 2008 targets: we extracted 90 chains (see Supplementary Materials) containing at least 10 β-residues (according to DSSP), and whose 3D structure had been determined by X-ray diffraction.

We first compared MLN-2S against BetaPro, using for both systems secondary structure and solvent accessibility predicted by SSPro and ACCPro, respectively (Cheng et al., 2005). In 62 out of 90 chains MLN-2S performed better than (or equal to) BetaPro. The difference is statistically significant according to a Wilcoxon paired test with \( P < 0.05 \).

We then compared MLN-2S against some of the best fully automated contact map predictors in CASP 2008: 3Dpro (Cheng et al., 2005), SAM-T08-2stage (Katzman et al., 2008) and MULTICOM-CMFR (Cheng, 2008). Since CASP predictors give as output a list of predicted contacts with an associated reliability score, we had to select a threshold for contact acceptance. In a first setting (Max-F₁ in Table 4), we chose the threshold maximizing the F₁ on contacts on our test set.

MLN-2S performs better than 3Dpro, SAM-T08-2stage and MULTICOM-CMFR in 72, 69 and 73 out of 90 chains, respectively (statistical significance is given by Wilcoxon paired test with \( P < 0.05 \)). To analyze the impact of secondary structure prediction, we also compared the four systems using true secondary structure. In this setting, our predictor performs better in 86, 87 and 87 chains, respectively. Comparative results are detailed in the scatter plots shown in Figure 4. Since only the most confident contact predictions submitted to CASP are usually evaluated, the overall precision of these systems is penalized when looking at a large number of contacts. Therefore, we compared performances in a second setting where only the top 1/5 predicted contacts are taken into account. Results in Table 4 show that in spite of a significant reduction in recall, these methods still achieve a precision comparable with MLN-2S. Finally, we chose the threshold to compare the four systems at the same recall level. Precision of MLN-2S is substantially higher in this setting.

4 CONCLUSIONS

In this work, we presented a statistical relational learning approach to the prediction of β-partners within a protein chain. We extended MLNs, in order to embed other discriminative classifiers like neural networks to compute grounding-specific weights. This approach also allows us to incorporate continuous features (such as multiple alignment profiles) in a natural way. Results show the benefits of the method, in particular where constraints induced by probabilistic logic rules eliminate unfeasible candidate configurations for the proteins. Improvements can be observed in the prediction of both coarse- and fine-grained contact map. Following previous literature, we have restricted our attention to intrachain hydrogen bonds. Interaction between β-strands is known to stabilize many protein complexes. Our approach can be naturally extended to handle interchain β-contacts, e.g. by introducing an additional predicate describing contacts between chains in the quaternary structure. Complexity would not be significantly increased if this predicate is given as evidence.

This kind of approach may be applicable in many other bioinformatics tasks, in which relational dependencies and knowledge of the domain play a crucial role. Multi-task learning can be easily implemented using Markov logic. Hence, an interesting future direction of research is the jointly prediction of β-partners together with other structural properties of proteins like, for example, secondary structure, solvent accessibility and dihedral angles.
Table 4. Comparative results on 90 CASP 2008 targets in three different settings

<table>
<thead>
<tr>
<th></th>
<th>BetaPro</th>
<th>MLN-2S</th>
<th>3Dpro</th>
<th>SAM</th>
<th>MLT</th>
<th>3Dpro</th>
<th>SAM</th>
<th>MLT</th>
<th>3Dpro</th>
<th>SAM</th>
<th>MLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>18.3</td>
<td>22.8</td>
<td>10.1</td>
<td>4.3</td>
<td>3.3</td>
<td>21.1</td>
<td>22.1</td>
<td>19.9</td>
<td>8.5</td>
<td>12.5</td>
<td>13.7</td>
</tr>
<tr>
<td>R</td>
<td>17.9</td>
<td>17.9</td>
<td>12.1</td>
<td>37.0</td>
<td>38.6</td>
<td>0.4</td>
<td>7.0</td>
<td>6.4</td>
<td>17.9</td>
<td>17.9</td>
<td>18.0</td>
</tr>
<tr>
<td>F₁</td>
<td>18.1</td>
<td>20.0</td>
<td>11.0</td>
<td>7.6</td>
<td>6.0</td>
<td>0.8</td>
<td>10.7</td>
<td>9.8</td>
<td>11.5</td>
<td>14.7</td>
<td>15.5</td>
</tr>
</tbody>
</table>

SAM is SAM-T08-2stage, MLT is MULTICOM-CMFR.

Fig. 4. Comparison on 90 CASP8 targets. For each target, we plot the $F_1$ measure on β-contacts achieved by MLN-2S (X-axis) versus the one achieved by a given contact prediction method in CASP8 (Y-axis). Upper panel: using secondary structure prediction by 3SPro. Lower panel: using true secondary structure.

ACKNOWLEDGEMENTS

We are grateful to Pierre Baldi and Arlo Randall for useful discussions.

Conflict of Interest: none declared.

REFERENCES


