BIOINFORMATICS

Vol. 16 no. 12 2000
Pages 1091–1104

Protein domain decomposition using a graph-theoretic approach

Ying Xu[1,*], Dong Xu[1] and Harold N. Gabow[2]

[1]Computational Biosciences Section, Life Sciences Division, Oak Ridge National Laboratory, Oak Ridge, TN 37830-6480, USA and [2]Department of Computer Science, University of Colorado, Boulder, CO 30309, USA

Received on May 10, 2000; revised on August 3, 2000; accepted on August 4, 2000

Abstract

Motivation: *Automatic decomposition of a multi-domain protein into individual domains represents a highly interesting and unsolved problem. As the number of protein structures in PDB is growing at an exponential rate, there is clearly a need for more reliable and efficient methods for protein domain decomposition simply to keep the domain databases up-to-date.*

Results: *We present a new algorithm for solving the domain decomposition problem, using a graph-theoretic approach. We have formulated the problem as a network flow problem, in which each residue of a protein is represented as a node of the network and each residue–residue contact is represented as an edge with a particular* capacity*, depending on the type of the contact. A two-domain decomposition problem is solved by finding a bottleneck (or a minimum cut) of the network, which minimizes the total cross-edge capacity, using the classical Ford–Fulkerson algorithm. A multi-domain decomposition problem is solved through repeatedly solving a series of two-domain problems. The algorithm has been implemented as a computer program, called* DomainParser*. We have tested the program on a commonly used test set consisting of 55 proteins. The decomposition results are 78.2% in agreement with the literature on both the number of decomposed domains and the assignments of residues to each domain, which compares favorably to existing programs. On the subset of two-domain proteins (20 in number), the program assigned 96.7% of the residues correctly when we require that the number of decomposed domains is two.*

Availability: *The executable of DomainParser and its web server are available at http://compbio.ornl.gov/structure/domainparser/.*

Contact: *xyn@ornl.gov*

*To whom correspondence should be addressed.

Introduction

Structural domains are considered as the basic units of protein folding, function, evolution, and design (Holm and Sander, 1994). While there has not been a precise and universally accepted definition of a structural domain, domains are generally considered as compact and semi-independent units of a protein, each of which may consist of a small number of continuous segments of the peptide chain and form a structurally 'separate' region in a protein three-dimensional (3D) structure (Wetlaufer, 1978; Richardson, 1981).

A number of popular protein structure databases, e.g. SCOP (Murzin *et al.*, 1995), DALI (Holm and Sander, 1996), and CATH (Orengo *et al.*, 1997), have been constructed based on the concept of structural domains. These databases provide an important basis for protein structure/function classification, analysis, prediction, and design. As the number of proteins being deposited into the PDB database (Bernstein *et al.*, 1977) increases at an exponential rate, we expect that the need for reliably and efficiently identifying structural domains from a solved protein structure will continue to increase, e.g. simply to keep the domain databases up-to-date.

Automatic identification (or decomposition) of domains of a given 3D structure has been an active research field since late 1970s when Wetlaufer published his study on protein domains (Wetlaufer, 1978). Numerous approaches have been proposed to formulate and solve this interesting and challenging problem. While earlier works were mainly focusing on domains consisting of a single peptide chain (Crippen, 1978; Nemethy and Scheraga, 1979; Rose, 1979; Lesk and Rose, 1981; Rashin, 1981; Zehfus and Rose, 1986), more general methods have been proposed in recent years to deal with domains containing multi-segments of a chain (Holm and Sander, 1994; Islam *et al.*, 1995; Sowdhamini and Blundell, 1995; Siddiqui and Barton, 1995; Wernisch *et al.*, 1999; Taylor, 1999). Though these approaches vary in their specific formulation of the problem, they generally follow one basic principle:

the (short-distance) residue–residue contacts are denser within a domain than between domains.

To this date, the domain identification problem remains an unsolved problem as indicated in a recent study by Jones *et al.* (1998). Based on their analysis on four popular domain identification programs (Holm and Sander, 1994; Siddiqui and Barton, 1995; Islam *et al.*, 1995; Swindells, 1995), they found that the most 'accurate' program is consistent in 76% of the cases with manually identified domains by experts on a data set consisting of 55 protein chains, and the four programs agreed in only 55.7% of the identified domains on a larger data set with 787 chains. Because of this reason, manual checking is generally required when decomposing a solved protein structure into domains and putting them into the domain databases like SCOP (Murzin *et al.*, 1995), DALI (Holm and Sander, 1996), and CATH (Orengo *et al.*, 1997). The manual process is a major barrier in updating these databases in a timely fashion.

We propose a new algorithm for the domain identification problem. The algorithm follows the same basic principle as the previous methods. We have formulated the domain identification problem as a network flow problem, which has been widely studied in the field of operations research (Ford and Fulkerson, 1962; Lawler, 1976). In this formulation, each residue is represented as a node of a connected network and each residue–residue contact, within certain cutoff distance between their atoms, is represented as an edge with a particular *capacity* value, depending on the type of interaction between the two involved residues. The basic problem we want to solve is to divide the network into two connected parts in such a way that the total edge capacity across the division is minimized. Intuitively, we want to find the *bottleneck* of the network. Based on the classical Ford–Fulkerson Theorem, this minimum-cut problem can be efficiently solved by finding the maximum flow of the network.

Using the representation by Picard and Queyranne (1980) of all minimum cuts, we can efficiently enumerate all cuts of the network that achieve the minimum cross-edge capacity. Having the capability of enumerating all minimum cuts allows us to evaluate and rank different domain decompositions in a post-processing step. Our test results have shown that this capability has helped to improve the quality of the decomposition.

For the more general situation where a protein may have multiple domains, our algorithm employs this network flow algorithm repeatedly to partition a protein into two parts until some stopping criteria are met. Currently the stopping criteria include various parameters related to the geometric and physical properties observed from known domains.

One of the key aspects of this work is to assign edge capacities in such a way that a minimum cut corresponds well with an interface between two domains. The parameters for capacity values are 'trained' based on a set of proteins with domains assigned manually by experts. Tests on a separate set of proteins are done. Similar levels of decomposition performance are achieved on the training and the test sets. Our preliminary test results suggest that this network flow formulation of the problem has captured the essence of the *basic principle* used in the various domain decomposition methods.

We have implemented the algorithm as a computer program, called *DomainParser*, using the C programming language. The program allows a user to either use default parameters or interactively change the parameters and to put constraints on the number of domains that a protein should be partitioned into. DomainParser also provides a confidence level for each assignment, based on the compactness of a domain and the tightness of the contacts between two domains. A user can decide if he/she may want to accept a partition or not, based on the confidence level. A web server for partitioning all the protein chains in PDB is freely available at http://compbio.ornl.gov/structure/domainparser/. If the web browser is configured to incorporate a molecular viewer, such as CHIME (MDL Information Systems, 1999) or RasMol (Sayle and Milner-White, 1995), the decomposition result can be viewed directly with the partitioned domains being color-coded.

Method

In this section, we first introduce a flow network representation of a protein structure, and also present an algorithm for domain decomposition, based on the network flow algorithm by Ford and Fulkerson (1962). Then we describe how the parameters in the DomainParser program are determined.

Problem formulation of two-domain decomposition

A *flow network* is a graph consisting of a set of nodes and a set of edges. A network has two distinguished nodes: a *source* s and a *sink* t. Each edge connects two nodes, and has a nonnegative *capacity*. An edge with zero capacity is equivalent to an edge that does not exist. For a 3D protein structure, we represent each residue by a node, and use an edge (between two nodes) to represent that the two residues are spatially close (e.g. the cutoff distance between their closest atoms is 4.0 Å in our current program). The capacity of an edge is defined so to reflect the packing between the two involved residues (more details in *Parameter determination* of this section). s and t are two artificially defined nodes, which we will explain later. Figure 1 shows an example of a flow network.

An *s–t cut* is a set of edges, whose removal leaves no path from s to t. For example, edges $\{(s, 1), (s, 2), (s, 3), (s, 4)\}$ form an s–t cut in Figure 1. A *minimum s–t* cut is an s–t cut that has the smallest total

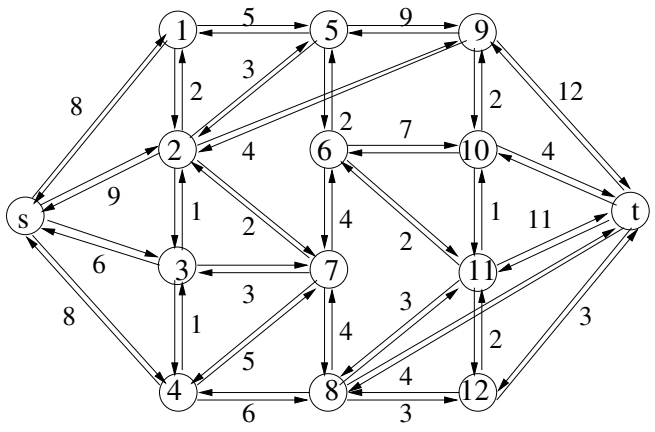

Fig. 1. A directed flow network. Each circle represents a node and a link between two nodes represents an edge. The number attached to each edge represents the capacity of the edge.

edge capacity. Edges $\{(1, 5), (2, 5), (2, 9), (4, 8), (6, 7), (7, 8)\}$ form a minimum s–t cut in Figure 1.

A minimum s–t cut can be calculated by finding a maximum flow from the source s to the sink t, based on the *maximum-flow/minimum-cut theorem*[†] (Ford and Fulkerson, 1962). In order to apply the Ford–Fulkerson algorithm[‡], we use a directed graph by assuming each edge (u, v) having two directed edges, one from node u to node v and one from v to u and both having the same capacity of (u, v), as shown in Figure 1.

A set of values assigned to the edges of the directed network forms an *s–t flow* if they satisfy the following three conditions. We use $f(u, v)$ to represent the flow value assigned to edge (u, v) and $c(u, v)$ the capacity of (u, v).

- *capacity constraint:* $f(u, v) \leqslant c(u, v)$, for each edge (u, v).
- *skew symmetry:* $f(u, v) = -f(v, u)$, for each edge (u, v).
- *flow conservation:* for each node u other than s and t, its total in-flow should be equal to its total out-flow, i.e.

$$\sum_v f(u, v) = 0,$$

where $\sum_v$ means summing over all nodes.

[†] The maximum-flow/minimum-cut theorem states: the value of a maximum flow from s to t is equal to the minimum edge capacity across a partition of the network that separates s and t.

[‡] Other algorithms can be used for the flow problem of an undirected network. Goldberg and Rao (1998) has the fastest maximum-flow algorithm for a directed network. But Ford–Fulkerson is easy to implement, and sufficient for our purpose.

The *value* of a flow f is defined as $\sum_v f(s, v)$. The *maximum s–t flow problem* is defined to find a flow f that has the largest possible value. The maximum s–t flow problem can be solved by the Ford–Fulkerson algorithm (Ford and Fulkerson, 1962; Lawler, 1976).

Ford–Fulkerson algorithm

This section outlines the Ford–Fulkerson algorithm, as implemented by Edmonds and Karp (1972). We first introduce a crucial definition of the Ford–Fulkerson algorithm. For a given flow f, the *residual capacity* of an edge (u, v) is defined as

$$c_f(u, v) = c(u, v) - f(u, v). \tag{1}$$

By the above definition of a flow, $c_f(u, v)$ is always $\geqslant 0$. Figure 2a shows the flow network of Figure 1b labeled with residual capacities, for a flow defined by

$$s \overset{f(s,2)=4}{\longrightarrow} \text{node 2} \overset{f(2,9)=4}{\longrightarrow} \text{node 9} \overset{f(9,t)=4}{\longrightarrow} t,$$

and

$$s \overset{f(s,4)=4}{\longrightarrow} \text{node 4} \overset{f(4,8)=4}{\longrightarrow} \text{node 8} \overset{f(8,t)=4}{\longrightarrow} t,$$

and the rest of the edges have flow $f = 0$, where $x \longrightarrow y$ represents a directed edge from node x to node y. Note that the residual capacity could be larger than the capacity since a flow could have a negative value (see the definition of a flow). In Figure 2, we did not draw edges with zero residual capacity.

The basic idea of the Ford–Fulkerson algorithm is to repeatedly find a directed path p from s to t, consisting of directed edges (u, v) with $c_f(u, v) > 0$; and then to increase the flow value f of each edge along p by the minimum value of $c_f(u, v)$ of p (and also update the values of $f(v, u)$ to keep the skew symmetry). This procedure continues until no such a path can be found. Initially, we set all f values to zero.

Ford and Fulkerson proved that this strategy finds a maximum s–t flow if all capacities are integral (Ford and Fulkerson, 1962). Edmonds and Karp further proved that if the directed path p has the smallest number of edges among all possible such paths, this algorithm runs in $O(nm^2)$ time (Edmonds and Karp, 1972), where n is the number of nodes and m is the number of edges[§]. Finding a path with the smallest number of edges can be done by doing a breath-first search of the network starting from the source s. By following this procedure, we can check that the value of a maximum s–t flow of the network in Figure 1b is 26. Figure 2b shows the network labeled with residual capacities when a maximum s–t flow is found. Apparently, there is no directed path that goes from s to t.

[§] In our formulation, the number of edges associated with each node is bounded from above by a small constant. Hence, $m = O(n)$.

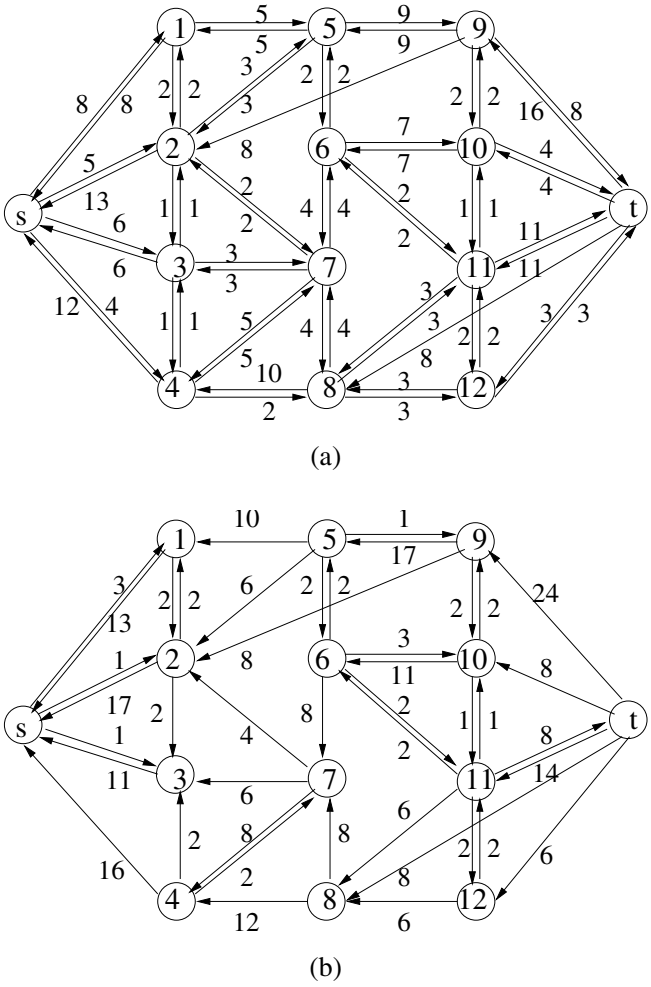

Fig. 2. The flow network labeled with residual capacities. (a) Residual capacities of a non-maximal *s–t* flow; (b) residual capacities of the maximum *s–t* flow.

A *residual network* is the network containing all edges with positive residual capacity. Given the residual network of a maximum *s–t* flow, one can find a minimum *s–t* cut by labeling all nodes that can reach the sink t, as a set T, and labeling the rest of the nodes as $\overline{T}$. Apparently there is no directed edge from $\overline{T}$ to T since otherwise more nodes will be added to T. This means that all the edges directed from $\overline{T}$ to T have zero residual capacity, and hence they form a minimum *s–t* cut. For the network of Figure 1b, the following edges form a minimum *s–t* cut: {(1, 5), (2, 5), (2, 9), (6, 7), (8, 11), (8, 12), (8, t)}. It is easy to check that the total capacity of these directed edges is 26, which is equal to the maximum *s–t* flow value as it should be. By removing these edges in Figure 1a, we get a partition of the network and of the corresponding protein.

A careful reader may have noticed that the minimum *s–t* cut is not unique, i.e. there are more than one cuts that have total cross-edge capacity of 26. For example, {(1, 5), (2, 5), (2, 9), (6, 7), (4, 8), (7, 8)} and {(s, 4), (1, 5), (2, 5), (2, 7), (2, 9), (3, 4), (3, 7)} also form minimum *s–t* cuts. The following section gives an algorithm that finds all minimum cuts of a network, based on the residual network of the Ford–Fulkerson algorithm.

Enumeration of all minimum cuts

Our enumeration procedure of minimum cuts consists of two components: (i) the enumeration of all 'interesting' *s–t* networks for a given protein, and (ii) the enumeration of all minimum *s–t* cuts for a given *s–t* network. As mentioned before, both s and t are artificially introduced nodes, which serve the following purpose. The Ford–Fulkerson algorithm requires a source and a sink. If we choose two nodes directly from the network representation of a protein as the source and the sink, we may get a trivial and incorrect partition, i.e. a partition consisting of one of the two nodes and the rest of the nodes. To avoid this, we want to select two groups of nodes, collectively as the source and the sink. s and t are used to implement this strategy by connecting to the two groups of nodes, respectively, with infinitely large edge capacities. This will force each group of selected nodes to stay in one domain. The enumeration of *s–t* networks is done using the following procedure.

For each node u representing a surface-exposed residue, add a source node s and create a directed edge from s to u and to each of the k residues that are closest to u spatially, where k is a parameter of the algorithm and its default value is set to be 30 (this number was selected through training as discussed in the following). Each of the added edges has a capacity of $+\infty$. For each fixed u, we go through all other surface-exposed residues v. For each such residue v, we create a sink t and $k + 1$ directed edges from v and v's k neighbors[¶] to t. Similarly, all these edges have a capacity of $+\infty$. We call u and v the *extreme* nodes. In the current implementation of DomainParser, we also require that the two extreme points should be certain distance apart and satisfy certain geometric and physical properties (more details in *Parameter determination* of this section).

For each *s–t* network, we have applied an algorithm by Picard and Queyranne (1980) to enumerate all minimum *s–t* cuts. The basis of the Picard–Queyranne algorithm is the following observation. *For a given residual network R and its source s and sink t, a partition of R's nodes into two disjoint sets S and T gives a minimum s–t cut of R if and only if* (i) *S contains s but not t, and* (ii) *no node of T can be reached from any node of S, through the*

[¶] We require that there is no overlap between s's neighbors and t's neighbors.

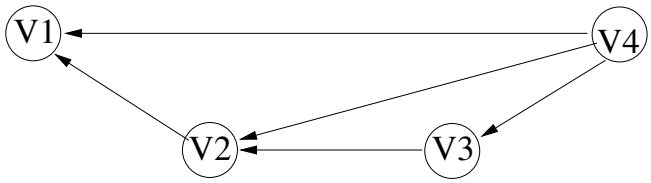

Fig. 3. The contracted network of the residual network, where $V1 = \{s, 1, 2, 3\}$, $V2 = \{4, 7\}$, $V3 = \{8\}$, and $V4 = \{5, 6, 9, 10, 11, 12, t\}$. A directed edge is placed between two contracted nodes if there is a directed edge between a pair of nodes belonging to the two contracted nodes, respectively, in R.

directed edges. The Picard–Queyranne algorithm gives an efficient way to enumerate all such S–T partitions.

We first introduce one useful concept for explaining the algorithm. A *strongly connected component* of a directed graph is a maximal subgraph such that every node of the subgraph can reach every other node of the subgraph through its directed edges. In Figure 2b, the subgraph consisting of nodes $\{4, 7\}$ forms a strongly connected component, but the subgraph consisting of nodes $\{3, 4, 7\}$ does not since node 3 can reach neither node 4 nor node 7. One simple observation about an S–T partition is that *if S contains a node x then S has to contain the strongly connected component (which was calculated in the residual graph) containing x*. The same is true for T. So conceptually, we can treat a strongly connected component as one single node. Also for the purpose of finding all the S–T partitions, we can conceptually consider all nodes reachable from s (including s) as one single node, and all nodes that can reach t (including) as one single node. Figure 3 shows the network of Figure 3b after conceptually contracting these nodes.

The enumeration of all S–T partitions can be done using the following procedure. We initialize S to be the contracted node containing s and T to be the contracted node containing t. Then we consider all possible ways to assign the other (contracted) nodes to S and T under one constraint—if a node is assigned to S then all nodes it can reach (through the directed edges) should be assigned to S. The enumeration algorithm by Schrage and Baker (1978) can be used to efficiently enumerate all such S–T partitions. The following lists all S–T partitions of Figure 3b:

- 1. $S = \{V1\}$ and $T = \{V2, V3, V4\}$;
- 2. $S = \{V1, V2\}$ and $T = \{V3, V4\}$;
- 3. $S = \{V1, V2, V3\}$ and $T = \{V4\}$.

Note that $S = \{V1, V3\}$ and $T = \{V2, V4\}$ do not form an S–T partition as defined above since node $V2$ of T is reachable from node $V3$ of S. It is not hard to check that each of these partitions gives a different minimum s–t cut of the original network. In the post-processing step, different partitions are evaluated and ranked using more global properties (see *Post processing* of this section). The following gives a pseudo-code of the Picard–Queyranne algorithm.

Procedure ENUMERATE_ALL_S-T_MIN_CUTS (R, s, t)
1. **begin**
2. find all strongly connected components of R, and contract each into one node;
3. find all nodes of R reachable from the source s, and contract them into one node;
4. find all nodes of R that can reach the sink t, and contract them into one node;
5. enumerate all S–T partitions of the contracted network using the Schrage-Baker algorithm;
6. for each S–T partition, replace each contracted node by the original nodes of R, and output it as a minimum s–t cut;
7. **end**

The strongly connected components of a network (line 2) can be found in linear time using Tarjan's algorithm (Tarjan, 1972). The following summarizes our enumeration procedure of all minimum cuts.

Procedure ENUMERATE_ALL_MIN_CUTS (P, L)
1. **begin**
2. set $L \leftarrow \emptyset$; /* L contains all minimum cuts */ set maxflow $\leftarrow 0$; /* maxflow records the current maximum flow */
3. enumerate s–t networks for protein P;
4. **for** each s–t network **do**
5. run Ford–Fulkerson algorithm to find a maximum s–t flow f and construct the residual network R;
6. **if** $f \geqslant$ maxflow **then**
7. **if** $f >$ maxflow **then** set $L \leftarrow \emptyset$; maxflow $\leftarrow f$;
8. call ENUMERATE_ALL_S-T_MIN_CUTS (R, s, t) to find all minimum s–t cuts and put them into L;
9. sort L lexicographically and merge the minimum cuts that yield the same domain interface.
10. **end**

In the worst case, a graph with n nodes may have up to 2^{n-2} s–t minimum cuts. Hence, the complexity of this procedure can be exponential. Fortunately, most partitions in actual proteins have only one minimum s–t cut. The largest number of the minimum s–t cuts observed in the proteins that we studied so far is 3. The small number is probably because protein domain interfaces are typically well defined so that not many alternatives exist.

Decomposition of multi-domain proteins

The core of DomainParser is a two-domain decomposition algorithm as outlined above. To deal with proteins having more domains (see Figure 4), DomainParser repeatedly bi-partitions a protein, using the core algorithm. It first represents a protein structure as a network as described in Section *Problem formulation of two-domain decomposition*, and finds a minimum cut of the network. Then it repeats this process by representing each partitioned sub-structure as a separate network until the following stopping criteria are met: the current (sub-)structure has less than 80 amino acids[‖], or its partitioned domains do not satisfy the necessary conditions of our domain definition (defined in terms of the compactness, the size of the interface versus the volume of a domain, etc.—details are given in Section *Parameter determination*). The following pseudo-code outlines the decomposition procedure. For simplicity of presentation, we consider here only the minimum cut of a network by the Ford–Fulkerson algorithm with fixed source and sink. The more general situation, i.e. considering different s–t pairs or a network with non-unique s–t minimum cuts, can be treated in a similar manner for each of the s–t minimum cuts.

```
Procedure Decomposition (P)
1.  begin
2.    if protein P has more than 80 amino acids then
3.      construct a network representation N of
        protein P with selected source s and sink t;
4.      call Ford–Fulkerson (N, s, t) to find an s–t
        minimum cut of N; let (P1, P2) be the
      corresponding bi-partition of P;
5.      call Decomposition (P1), and call
        Decomposition (P2);
6.      if P1 or P2 is labeled as 'rejected' then
7.        label both P1 and P2 as 'rejected';
8.        label P as 'accepted' or 'rejected' based
          on the criteria given in Section Domain
          evaluation and refinement: a post-
          processing step;
9.    else
10.     label P as 'accepted' or 'rejected' based on the
          criteria given in Section Domain evaluation
          and refinement: a post-processing step;
11.   label each 'accepted' sub-structure as a domain.
12. end
```

In the general situation (when considering non-unique minimum cuts), this procedure generates a list of possible ways to decompose a protein into individual domains. A post-processing step (see Section *Domain evaluation and refinement: a post-processing step*) is used to rank the generated 'domains', using various geometric and physical parameters.

Parameter determination

The DomainParser program uses three classes of parameters: (a) parameters related to edge capacities, (b) parameters used for the selection of the extreme points, and (c) parameters used for post processing. Each of the parameters is determined in such a way to optimize the decomposition performance on a training set with domains already identified by experts.

Our training set consists of 169 single-domain protein chains and 34 two-domain chains, which are selected from a set of 284 proteins collected by the authors of Islam *et al.* (1995). These proteins are selected for the following reason. Among the 284 proteins, 55 proteins have been used for performance testing by Jones *et al.* (1998). For the purpose of comparison with other programs, we decided to use these 55 proteins as our test set and excluded them from the training set. Also excluded from the training set are (1) protein structures with only C_α coordinates, and (2) proteins with domains consisting of less than 40 residues (PDB codes: 1bbo, 1cpca, 1cpcl, and 4rcrh). We have also excluded proteins with more than two domains from the training set. This leaves 169 single-domain protein chains[**] and 34 two-domain protein chains[††].

Determination of edge capacity. Two atoms are said to be in *contact* if their distance is 4.0 Å or less, following the definition of Holm and Sander (1994). An edge is created between two residues if they have at least one pair of atoms in contact. In assigning the capacity of an edge, we are trying to capture some of the general rules used for domain decomposition by human experts. The very basic rule is that residue–residue contacts should be denser within a domain than between domains. In addition, we also intend to enforce the following rules:

[‖] A partition for a (sub-)structure having less than 80 amino acids will yield a sub-structure with 40 amino acids or less, and hence violate our requirement for the minimum domain size.

[**] The PDB codes of these proteins (with the fifth letter indicating the chain name, if any): 0acha 0sc2a 1aaf 1aapa 1aba 1ads 1aps 1atx 1ayh 1baa 1babb 1barb 1bba 1bbl 1bbt1 1bbt2 1bgc 1bop 1bova 1btc 1bw4 1c2ra 1c5a 1caj 1cbn 1cbp 1cd8 1cdta 1cis 1cmba 1coba 1csei 1ctc 1d66a 1dhr 1dnka 1eaf 1eco 1efm 1egf 1end 1erp 1fas 1fbaa 1fc2c 1fcs 1fdd 1fha 1fiab 1glaf 1glua 1gps 1hcc 1hddc 1hgeb 1higa 1hiva 1hsda 1ifa 1ifci 1isua 1ixa 1le4 1ltsa 1ltsc 1ltsd 1mdaa 1mdah 1mdc 1mrra 1ms2a 1mup 1nipb 1nrca 1nxb 1omf 1ovb 1paz 1pcda 1pdc 1phb 1phy 1poa 1poc 1ppba 1prcc 1prcm 1prf 1pte 1r094 1r1a2 1r1ee 1rea 1rnd 1rpra 1rro 1s01 1shaa 1shfa 1tabi 1ten 1tfg 1tfi 1tgsi 1tho 1tima 1tnfa 1ttba 1utg 1vaab 1wrpr 1xima 1ycc 256ba 2avia 2bds 2bpa1 2bpa2 2bpa3 2cbh 2cdv 2cpl 2crd 2cro 2dpv 2ech 2gb1 2hhma 2hipa 2hpda 2ihl 2ila 2lala 2lalb 2madl 2mev1 2mev4 2mhr 2msba 2pf2 2plv1 2plv3 2por 2scpa 2sn3 3adk 3b5c 3cbh 3dfr 3il8 3mona 3pgm 3rubs 3sgbi 3sici 4cpai 4enl 4fxn 4htci 4sbva 4sgbi 4tgf 4tms 5nn9 5tgle 7apib 8i1b 8rxna 9rnt

[††] The names of these proteins: 1abk 1abma 1arb 1caua 1caub 1cid 1dri 1fc1a 1glag 1gssa 1hila 1l92 1lgaa 1mamh 1omp 1osa 1ppfe 1sbp 2cts 2er7e 2glsa 2liv 2sga 2snv 2tbva 3cox 3gbp 4enl 4gpd1 4ts1a 6ldh 7aata 8abp 9rubb

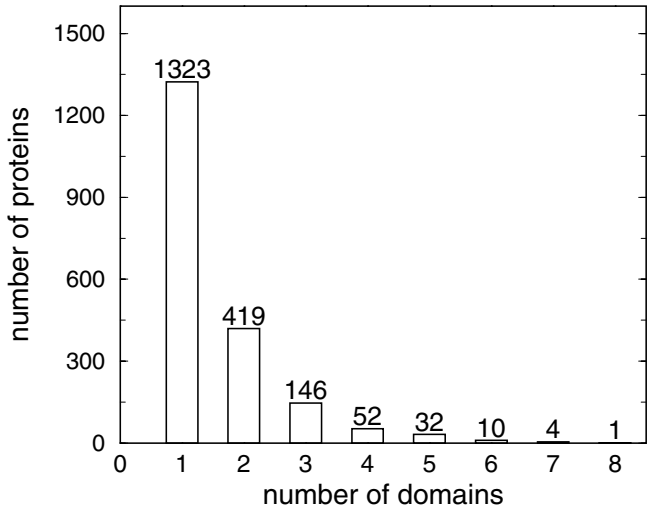

Fig. 4. Distribution of the number of domains for the protein chains in FSSP.

- each domain should not have many discontinuous sequence segments; or equivalently, the backbone contact should not be cut frequently in the decomposition process;
- a decomposition should generally avoid splitting a β-sheet into different domains;
- a β-strand should generally not be cut.

We use the following function to assign the capacity of an edge (u, v).

$$c(u, v) = k_{u,v} + k^b_{u,v}\omega_b + k^\beta_{u,v}\omega_\beta + k^e_{u,v}\omega_e \ . \qquad (2)$$

$k_{u,v}$ is the number of atom–atom contacts between residues u and v. $k^b_{u,v}$ is the number of backbone–backbone atom contacts between u and v, which gives additional weights to backbone–backbone atom contacts. $k^\beta_{u,v} = 1$ if u and v form a backbone–backbone hydrogen bond across a β-sheet, otherwise it is 0. This term is mainly used to preserve a β-sheet in a domain decomposition. $k^e_{u,v} = 1$ if u and v belong to the same β-strand, and it is 0 otherwise. ω_b, ω_β and ω_e are scaling factors. The first two are to be 'trained' on our training set. ω_e is determined separately.

In training ω_b and ω_β, our goal is to find values for them so that the total number of residues assigned to the wrong[‡‡] domains is as small as possible. The search for the 'optimal' values is done using a procedure called the *orthogonal array method* (Sun *et al.*, 1999). This procedure starts with a coarse search grid, and gradually focuses on a reduced and finer search grid. It converges to local optima quickly. The following are the values we have obtained through training:

$$\omega_b \ = \ 5 \ ; \qquad \omega_\beta \ = \ 12 \ .$$

Typically a domain partition should not split a β-strand into two different domains. So ω_e should have a value $+\infty$. But we have seen a few cases where human experts have cut a long β-strand into two domains in all-β proteins (e.g. in 3cd4). With only a very few such cases, we find it difficult to systematically 'train' the parameter ω_e. In the current version of Domainparser, we have arbitrarily assigned a large number (1000) to ω_e. This should avoid cutting a β-strand when other possible partitions exist, but still allow the possibility of cutting a β-strand in an all-β protein.

Selection of extreme points. Extreme points (see Section *Enumeration of all minimum cuts* for definition) are used as 'seeds' of domains to be identified. Clearly, different seeds may lead to different decomposition results. To overcome the problem that an incorrect selection of seeds may result in incorrect domain decomposition, we consider multiple possibilities of seeds and use a post-processing step (see Section *Domain evaluation and refinement: a post-processing step*) to rank various decompositions using more global information. Our current selection rule is a result of the trade-off between prediction accuracy and computational efficiency. We select three sets of extreme-point pairs from the top 5% of the farthest pairs (in 3D space) from the each of the following sets: (i) all residue pairs in a structure; (ii) the residue pairs whose connecting lines are perpendicular (allowing 5^o-orientation deviation) to the line between the farthest residue pair in the structure; and (iii) the residue pairs whose constituting residues are on different sides of the *minimal contact-density point* along the sequence axis, where the contact density (Islam *et al.*, 1995) at sequence position k for a structure of n residues is defined by

$$\frac{\sum_{u=1}^{k} \ \sum_{v=k+1}^{n} \ c(u, v)}{k(n-k)} \ . \qquad (3)$$

Overlaps are removed if different rules produce the same extreme-point pairs. We have found that generally, the more extreme points we consider, the better decomposition results (after ranking) we can expect and the more expensive the computation will be. Then the improvement becomes asymptotic beyond a certain number of extreme points used (see Figure 5).

Parameters for post processing. The post-processing step is used to evaluate and rank decomposition results.

[‡‡]Here we consider the domains assigned by the authors of these proteins as the correct assignments.

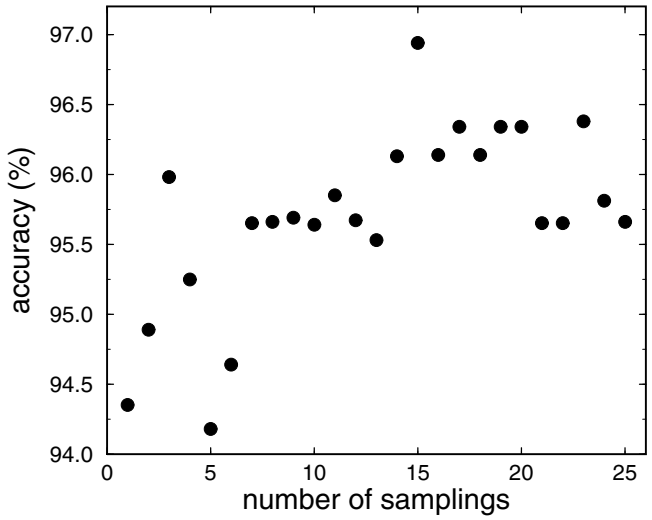

Fig. 5. The accuracy of domain assignment *vs.* the number of extreme-point pairs used. The data consists of both the 34 two-domain protein chains in the training set and the 20 two-domain protein chains in the test set. The accuracy is the percentage of agreement between the manual assignments by experts and the assignments by DomainParser when we require that the number of decomposed domains is two.

Three parameters g_m, f_m, and l_s are used in this step, for determining if a decomposed domain is consistent with the general characteristics of known domains. g_m is a threshold for the compactness of a partitioned domain; f_m is a threshold for the 'size' of a domain interface relative to the 'volume' of the domain; and l_s is a threshold for the number of residues per segment in a domain.

We have used the same search procedure as outlined above to find the 'optimal' parameter values. The training set for these parameters consists of only the 34 two-domain proteins. The objective here is to find values of g_m, f_m, and l_s so that the number of structures that are partitioned into two domains is as high as possible. The following are the values we have obtained:

$$g_m \;=\; 0.54\,; \quad f_m \;=\; 0.52\,; \quad l_s = 35\,.$$

Domain evaluation and refinement: a post-processing step

The post-processing step serves two purposes: (i) ranking or rejecting decomposed domains, and (ii) refining the accepted domain decompositions. It applies more global information about a domain to evaluate and improve the quality of domain decompositions.

Evaluation of decomposed domains. Certain partitioned 'domains' are simply not consistent with the general characteristics of a *domain*; and some partitioned domains look more reasonable than the others. Here we use the overall geometric and physical properties observed from known domains to evaluate the partitioned 'domains'. DomainParser uses the following rules, similar to those used in Holm and Sander (1994), to reject a *bad* domain decomposition:

- A domain should have at least 40 residues.
- At most one β-strand can be cut at the interface between two domains; and a β-sheet having more than 2 residues in each strand can belong to only one domain.
- A domain must be compact enough to satisfy the following condition (Holm and Sander, 1994):

$$\frac{\sum_{i,j}\, p_{i,j}}{n_a} \;\geqslant\; g_m\,, \tag{4}$$

  where i and j are any two atoms separated by at least three residues on the sequence; $p_{i,j} = 1$ if the distance between i and j is 4.0 Å or less, otherwise $p_{i,j} = 0$; and n_a is the number of atoms in the domain.
- The interface between two domains must be small enough to satisfy

$$\frac{\sum_{\text{inter-domain}}\, p_{i,j}}{\sum_{\text{intra-domain}}\, p_{i,j}} \;\leqslant\; f_m\,. \tag{5}$$

- The number of segments in a domain, D, is not too many such that

$$\frac{r(D)}{s(D)} \;\geqslant\; l_s\,, \tag{6}$$

  where $r(D)$ and $s(D)$ are the numbers of residues and segments in a domain D, respectively.

DomainParser ranks the partitioned domains that which pass these rules, using the following ranking function:

$$q \;=\; \frac{s(D_1)\; s(D_2) \sum\limits_{\text{inter-domain}} p_{i,j}}{r(D_1)\; r(D_2) \sum\limits_{\text{intra-domain}-1} p_{i,j} \sum\limits_{\text{intra-domain}-2} p_{i,j}}. \tag{7}$$

In DomainParser, the lower the q value, the higher the rank. In addition, DomainParser uses a combination of (1) q, (2) the compactness, and (3) the interface size versus the volume of a domain as an indicator of its prediction confidence.

Decomposition refinement. DomainParser may refine an 'accepted' partitioned domain, using various empirical rules. For example, some short segments may 'dip' in and out of one domain while most of its flanks are in another

Table 1. Decomposition of multi-domain proteins.

Protein	Literature	DomainParser	Agreement
2 domains:			
1ezm	1–134/135–298	1–133/134–298	99.7%
1fnr	19–161/162–314	19–152/153–314	97.0%
1gpb	19–489/490–841	19–63/64–484;828–841/558–648; 712–792/485–557;649–711;793–827	overcut
1lap	1–150/171–484	1–173/174–484	99.4%
1pfka	0–138;251–301/139–250;302–319	0–137;254–319/138–253	93.1%
1ppn	1–10;112–208/21–111;209–212	1 domain	undercut
1rhd	1–158/159–293	1–63;74–157/64–73;158–293	96.3%
1sgt	22–123;234–245/129–233	1 domain	undercut
1vsga	1–29;92–251/42–75;266–362	1–32;86–255/33–85;256–362	100.0%
1wsyb	9–52;86–204/53–85;205–393	90–189/9–89;190–393	83.6%
2cyp	3–145;266–294/164–265	2–144;273–294/145–272	97.1%
2had	1–155;230–310/156–229	1 domain	undercut
3cd4	1–98/99–178	1–98/99–178	100.0%
3gapa	1–129/139–208	1 domain	undercut
3pgk	1–185;403–415/200–392	0–188;402–415/189–401	100.0%
4gcr	1–83/84–174	1–83/84–174	100.0%
5fbpa	6–201/202–335	1 domain	undercut
8adh	1–175;319–374/176–318	1–173;321–374/174–320	98.9%
8atca	1–137;288–310/144–283	1–130;292–310/131–291	96.3%
8atcb	8–97/101–152	8–97/101–153	100.0%
3 domains:			
1phh	1–175/176–290/291–394	32–124/180–268/1–31;125–179;269–394	72.6%
3grs	8–157;294–364/158–293/365–478	8–161;290–368/162–289/369–478	97.5%
4 domains:			
1atna	1–32;70–144;338–372/33–69 /145–180;270–337/181–269	0–33;97–147;337–372/34–96 /148–180;273–336/181–272	90.6%
2pmga	1–188/192–315/325–403/408–561	1–188/189–303/304–406/407–561	97.8%
8acn	2–200/201–317/320–513/538–754	2–530/531–754	undercut

The four columns show protein PDB codes, residue ranges of domains assigned by the literature ('/' is used to separate domains), residue ranges of domains assigned by DomainParser, and the percentage of overlap between the literature assignments and the DomainParser assignments, respectively.

domain. They are typically formed due to the structural adjustment in the packing between the domains (Xu *et al.*, 1998). To make the domain partition more biologically meaningful and to avoid creating too many short segments in a domain, the program re-assigns the short segment to the domain which contains its flanks if such a segment has less than 10 residues. A similar rule is applied to a segment at the terminus of a protein sequence and with less than five residues in the domain. The decomposition refinement prevents too many segments in a domain, and is generally used by the other domain assignment programs.

Results

Using the 'trained' parameters, we have tested the performance of DomainParser on a set of 55 proteins, the same test used by Jones *et al.* (1998). Among the 55 proteins, 30 are single-domain proteins, 20 are two-domain proteins, 3 are three-domain proteins, and 2 are four-domain proteins. Jones *et al.* consider a domain decomposition as *correct* if the number of decomposed domains is the same as in the literature (i.e. the manual assignments by the authors of the structures) and the residue assignment is at least 85% in agreement with the structure authors (Jones *et al.*, 1998). Using this definition, DomainParser correctly assigned 27 single-domain proteins, 13 two-domain proteins, 1 three-domain protein (1phh), and 2 four-domain protein (1atna and 2pmga), i.e. 78.2% of the 55 proteins. Table 1 lists the decomposition results for all 25 multi-domain proteins. Figure 6 shows four decomposition examples from this set.

Among the single-domain proteins with incorrect decompositions, all three are decomposed into two domains: 1gky (15–97;182–186/0–14;98–181), 1ula (1–114;220–236;255–289/115–219;237–254), and 3dfr (36–109/1–35;110–162). For the seven two-domain

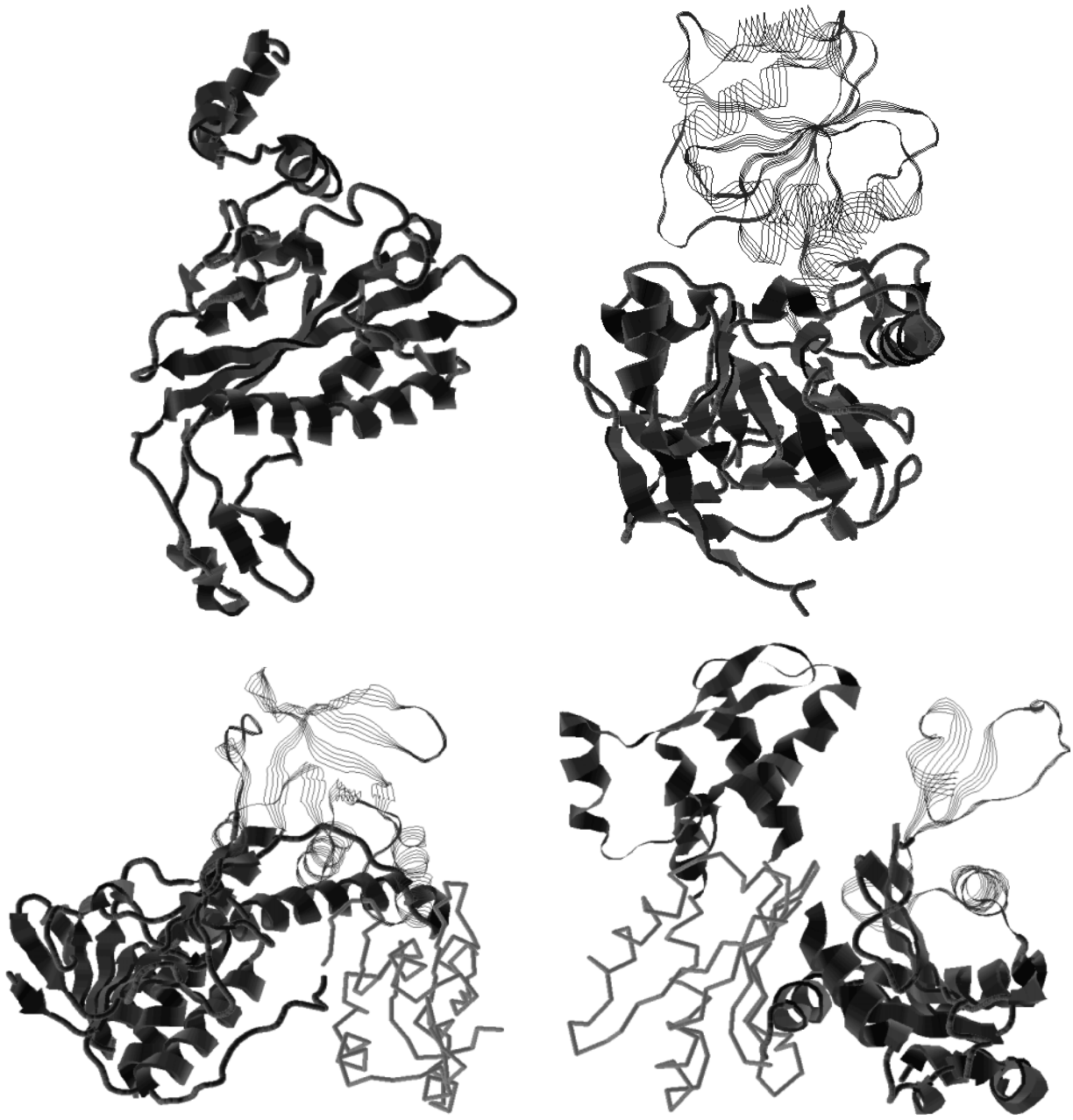

Fig. 6. Examples of domain decompositions by DomainParser. The different representations (thick ribbons, thin ribbons, strands, and backbone traces) show different domains.

chains with wrong predictions, five chains (1ppn, 1sgt, 2had, 3gapa, and 5fpba) are undercut and assigned to single domains; one chain (1gpb) is overcut into four domains; and one chain (1wsyb) is correctly cut into two domains, but the residue assignment is only 83.6% in agreement with the manual assignment by experts. For the five chains with more than two domains, DomainParser has clearly done a better job than the four existing programs (Holm and Sander, 1994; Siddiqui and Barton, 1995; Islam *et al.*, 1995; Swindells, 1995) as assessed by Jones *et al.* (1998). While DomainParser has assigned three of them correctly, only two of these programs assigned one correctly.

We have tested the stability of DomainParser in terms of its prediction accuracy versus the exact choice of extreme points. For each protein, if we call the extreme points

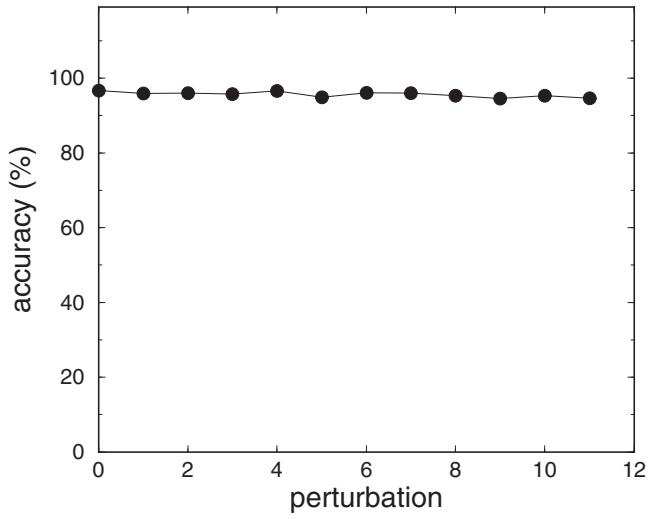

Fig. 7. The accuracy of domain assignments (on 20 two-domain proteins in the test set) as the extreme points drift away from the optimal ones. The x-axis represents the kth spatially closest residue (measured between the C_α distance) of the optimal extreme point (with 0 representing the optimal one). The y-axis represents the percentage of agreement between the manual assignments by experts and the assignments by DomainParser when we require that the number of decomposed domains is two.

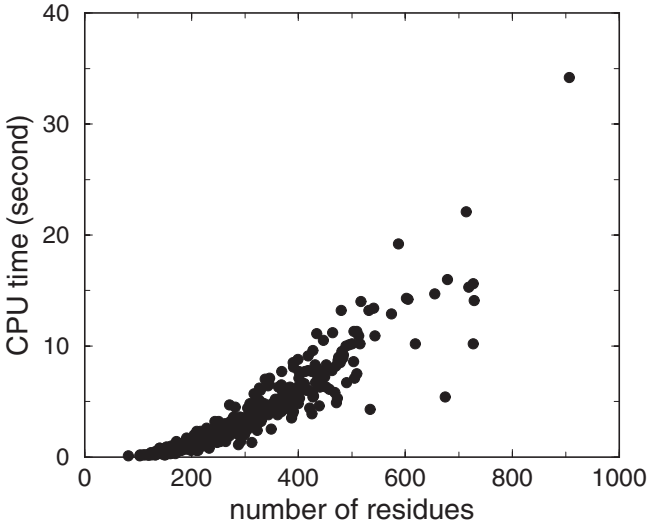

Fig. 8. CPU time of running DomainParser as a function of the number of residues for 419 two-domain chains in FSSP.

which give the best decomposition (ranked #1) the *optimal* ones, we have found that its performance level does not change much if we select other residues as extreme points, which are spatially close to the optimal ones (see Figure 7). This indicates that our program is quite stable.

We have applied DomainParser to the non-redundant protein chains (a total of 1987) in the FSSP database (Holm and Sander, 1996) (release of January, 2000). The decomposition results of all 1987 chains can be found at http://compbio.ornl.gov/structure/domainparser/. A summary of the results can be found in Figure 4. DomainParser runs efficiently. For virtually every protein chain in FSSP, it takes less than 30 seconds CPU time to complete the decomposition on a DEC/alpha workstation. Figure 8 gives the computational time for all two-domain chains of this set (419 in total).

Discussion

On the test set used by Jones *et al.* (1998), DomainParser compares favorably to other existing programs. Comparing to the overall accuracy level ranging from 67 to 76% by the four exiting programs (Holm and Sander, 1994; Siddiqui and Barton, 1995; Swindells, 1995; Islam *et al.*, 1995), DomainParser achieves an accuracy level of 78.2% on the same set. Particularly worth mentioning is that the improvement by DomainParser on proteins with more than two domains is quite significant. Since the definitions of domains and parameters used in DomainParser are similar to the ones used in other programs, we believe that the main strengths of DomainParser are (a) that it does not rely on the topological information of a protein structure (i.e. how residues connect with each other on the sequence), while the four existing methods all do directly or indirectly; and (b) that the Ford–Fulkerson method provides a rigorous and robust way for doing partitioning. Relying solely on geometric information, i.e. the contact densities within a domain and between domains, makes DomainParser more general and robust. One test result is quite revealing. On the 20 two-domain test proteins, DomainParser has assigned 96.7% of residues correctly if we require that the number of decomposed domains is two (DomainParser allows a user to do that). This suggests that our current rules for stopping is inadequate, particularly knowing that most of our incorrect decomposition results (on multi-domain proteins) are caused by undercutting. By using more sophisticated rules for defining the necessary conditions of a 'domain', we believe that we can significantly improve the performance of DomainParser. Studies are ongoing to test the performance of using different new rules and related parameters.

Some of the discrepancies between our assignments and the ones in the literature may not necessarily indicate that our assignments are incorrect in these cases. It may simply be a result of the lack of precise definition of a structural *domain*, as pointed out by several studies (Taylor, 1999; Wernisch *et al.*, 1999). The manual assignments by experts are sometimes quite subjective, depending on what they believe constitutes a protein domain. Figure 9 shows

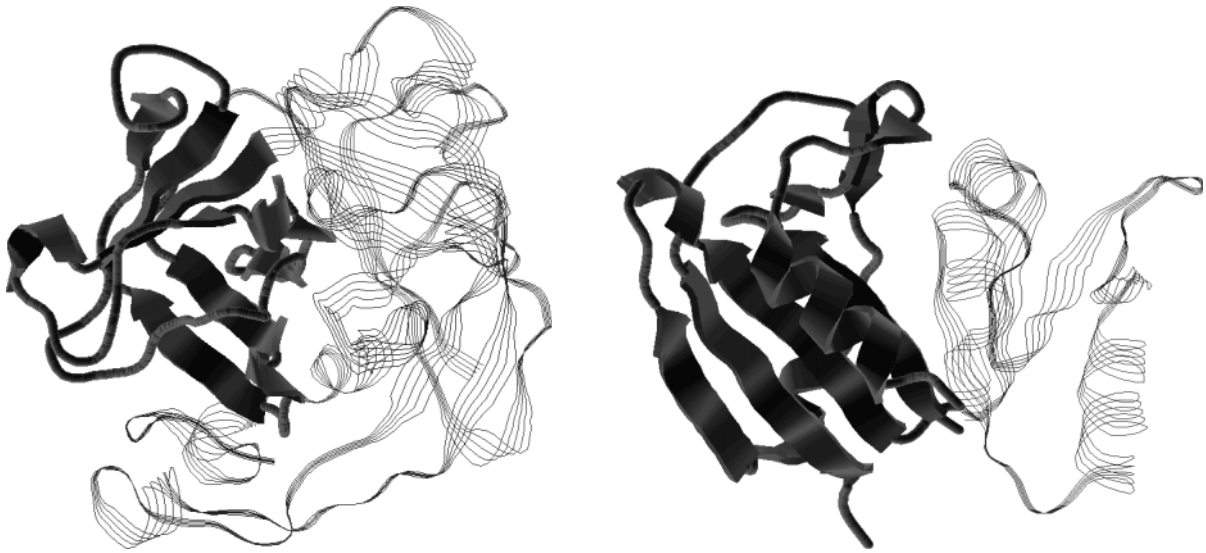

Fig. 9. (a) Manual decomposition of 1arb by experts (1–139;229–263/140–228). (b) Domain decomposition of 3dfr by DomainParser (36–109/1–35;110–162). The thick ribbons and thin strands show different domains.

two examples (1arb and 3dfr). The manual decomposition of 1arb by experts contains two domains, while DomainParser assigns only one domain for the protein. The packing between two manually assigned domains is actually very tight. In contrast, DomainParser assigns two domains for 3dfr, while the experts' assignment contains only a single domain. However, the packing between the two manually-assigned domains in 1arb is tighter than the packing between the two domains assigned by DomainParser for 3dfr. So it is clear that there exist inconsistencies in the decomposition rules used by different human experts. We have also noticed that different experts may give different domain assignments even for the same proteins. For example, 1sgt was assigned as a two-domain protein by the authors of the structure (Read and James, 1988). But SCOP (Murzin *et al.*, 1995) has assigned it as a single-domain protein (as assigned by DomainParser), based on the close interactions between the two 'domains'. Another ambiguity is the assignment of short segments. If a short segment 'dips' in and out of one domain while most of its flanks are in another domain, it can be assigned to the domain of its flanks depending on the size of the segment. Different experts use different size cutoff. Some have used a cutoff smaller than 10 residues (e.g. residues 1–10 in 1ppn was assigned to the domain it dipped into), while others use a cutoff as large as 14 residues (e.g. residues 828–841 in 1gpb was assigned to the domain of its flanks).

The inconsistencies in experts' assignments suggest that a rigorous definition of a structural domain is needed. Only then one can have a set of systematic rules to correctly assign domains without manual checking. An interesting experiment we have done is to modify the parameters and the decomposition rules used in DomainParser. We found that DomainParser can correctly assign each of the 55 proteins with very high agreement with the literature assignment. This indicates that once a set of decomposition rules are rigorously defined, DomainParser can assign domains very reliably.

Other discrepancies between our assignments and the ones in the literature suggest possible directions for improvements. We found that most of the wrong assignments are caused by our rules of accepting or rejecting a partitioned domain in the post processing. Inequalities (4) and (5) are too simple for enforcing the compactness of a domain and the tightness of the contacts between two domains. Figure 10 shows two examples (1mrra and 3gapa) of poor partitions by DomainParser. The chain 1mrra, which is assigned as a single-domain structure by experts, is overcut by DomainParser into two domains. DomainParser regards the interactions between the two 'domains' as weak compared with the tight helical packing within each 'domain'. On the other hand, the chain 3gapa, which is assigned as a two-domain structure by experts, is undercut by DomainParser into one domain. Although DomainParser initially partitioned 3gapa correctly (1–123/124–208, with an accuracy of 97.0%), the post processing regards the interactions between the two domains as too tight compared with the packing density within the small domain (124–208). More studies are ongoing to explore possible improvement in the post-processing criteria. For example, we are trying to use information about surface area and hydrophobic-

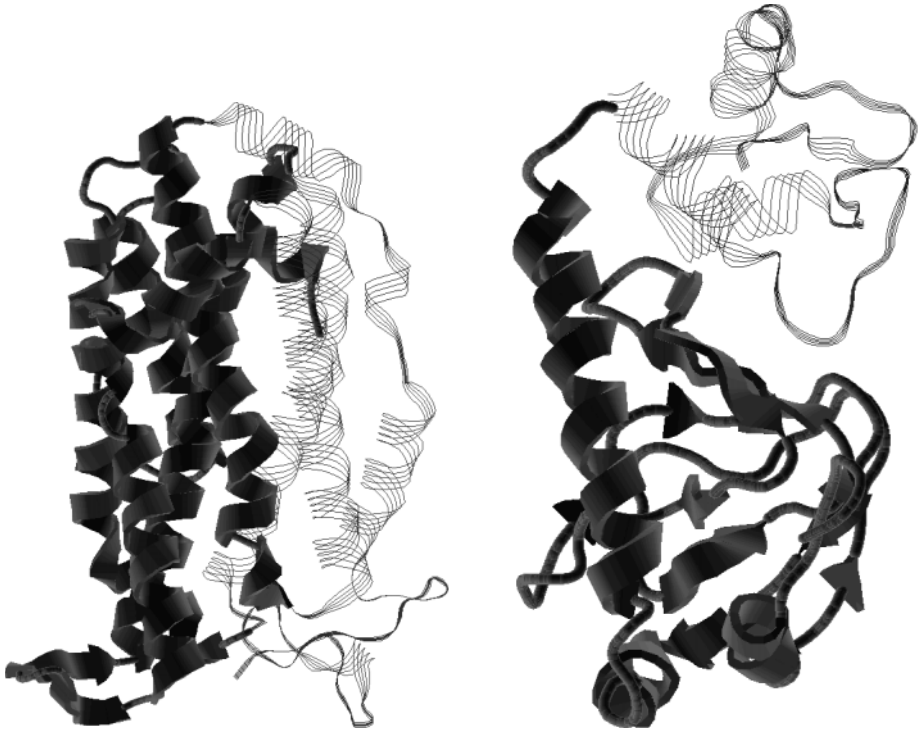

Fig. 10. (a) Decomposition of 1mrra by DomainParser (1–65;100–131;224–257/66–99; 132–223;258–340). (b) Manual domain decomposition of 3gapa by experts (1–129/139–208). The thick ribbons and thin strands show different domains.

ity Tsai and Nussinov (1997) as well as use cutoff values depending on the size of domain and the composition of secondary structure types in a domain. Some new definitions about the packing density (Tsai *et al.*, 1999) can be employed to see their impact on the performance of domain partitions. In addition, using information of recurrent domains (Holm and Sander, 1998) in PDB and domains determined through multiple sequence alignment (e.g. ProDom Corpet *et al.* (1999) and Pfam Bateman *et al.* (1999)) will probably make the domain partition more reliable and more function related.

In summary, we have developed a computer program for protein domain decomposition, based on a network-flow representation of a protein and a rigorous algorithm for finding minimum cut of the network. Our preliminary test results have been quite encouraging. We expect that using the network flow algorithm as a partition technique will help us move one step closer towards reliable and automated domain assignments.

Acknowledgements

The authors thank Drs Victor Olamn and Ed Uberbacher for helpful discussions. We also thank the anonymous reviewers for their constructive suggestions, which have helped improve the presentation of the paper. Y.Xu and D.Xu were supported by the Office of Biological and Environmental Research, U.S. Department of Energy, under Contract DE-AC05-00OR22725, managed by UT-Battelle, LLC.

References

Bateman,A., Birney,E., Durbin,R., Eddy,S.R., Finn,F.D. and Sonnhammer,E.L.L. (1999) Pfam 3.1: 1313 multiple alignments match the majority of proteins. *Nucleic Acids Res.*, **27**, 260–262.

Bernstein,F.C., Koetzle,T.F., Williams,G.J.B., Meyer,E.F., Brice,M.D., Rodgers,J.R., Kennard,O., Shimanouchi,T. and Tasumi,M. (1977) The protein data bank: a computer based archival file for macromolecular structures. *J. Mol. Biol.*, **112**, 535–542.

Corpet,F., Gouzy,J. and Kahn,D. (1999) Recent improvements of the ProDom database of protein domain families. *Nucleic Acids Res.*, **27**, 263–267.

Crippen,G. (1978) The tree structural organization of proteins. *J. Mol. Biol.*, **126**, 315–332.

Edmonds,J. and Karp,R.M. (1972) Theoretical improvemnts in the algorithmic efficiency for network flow problems. *J. ACM*, **19**, 248–264.

Ford,L.R. and Fulkerson,D.R. (1962) *Flows in Networks*. Princeton University Press, Princeton, New Jersey.

Goldberg,A. and Rao,S. (1998) Beyond the flow decomposition barrier. *J. ACM 45*, **5**, 783–798.

Holm,L. and Sander,C. (1994) Parser for protein fodling units. *Proteins: Struct. Funct. Genet.*, **19**, 256–268.

Holm,L. and Sander,C. (1996) Mapping the protein universe. *Science*, **273**, 595–602.

Holm,L. and Sander,C. (1998) Dictionary of recurrent domains in protein structures. *Proteins: Struct. Funct. Genet.*, **33**, 88–96.

Islam,S.A., Luo,J. and Sternberg,M.J. (1995) Identification and analysis of domains in proteins. *Protein Eng.*, **8**, 513–525.

Jones,S., Stewart,M., Michie,A., Swindells,M.B., Orengo,C. and Thornton,J.M. (1998) Domain assignment for protein structures using aconcensus approach: characterization and analysis. *Protein Sci.*, **7**, 233–242.

Lawler,E.L. (1976) *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York.

Lesk,A.M. and Rose,G.D. (1981) Folding units in globular proteins. *Proc. Natl. Acad. Sci. USA*, **78**, 4304–4308.

MDL Information Systems, (1999) *CHIME*. MDL Information Systems, Inc., San Leandro, Carlifornia.

Murzin,A.G., Brenner,S.E., Hubbard,T. and Chothia,C. (1995) Scop: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, **247**, 536–540.

Nemethy,G. and Scheraga,H.A. (1979) A possible folding pathway of bovine pancreatic RNase. *Proc. Natl. Acad. Sci. USA*, **76**, 6050–6054.

Orengo,C.A., Michie,A.D., Jones,S., Jones,D.T., Swindells,M.B. and Thornton,J.M. (1997) CATH: a hierarchic classification of protein domain structures. *Structure*, **5**, 1093–1108.

Picard,J.C. and Queyranne,M. (1980) On the structure of all minimum cuts in a network and applications. *Mathematical Programming Study*, **13**, 8–16.

Rashin,A.A. (1981) Location of domains in globular proteins. *Nature*, **291**, 86–87.

Read,R.J. and James,M.N. (1988) Refined crystal structure of streptomyces griseus trypsin at 1.7 A resolution. *J. Mol. Biol.*, **200**, 523–551.

Richardson,J.S. (1981) The anatomy and taxonomy of protein structure. **34**, 167–339.

Rose,G.D. (1979) Hierarchic organization of domains in globular proteins. *J. Mol. Biol.*, **134**, 447–470.

Sayle,R.A. and Milner-White,E.J. (1995) RASMOL: biomolecular graphics for all. *Trends Biochem. Sci.*, **20**, 374–376.

Schrage,L. and Baker,K.R. (1978) Dynamic programming solution of sequencing problems with precedence constraints. *Oper. Res.*, **28**, 444–449.

Siddiqui,A.S. and Barton,G.J. (1995) Continuous and discontinuous domains: an algorithm for the automatic generation of reliable protein domain definitions. *Protein Sci.*, **4**, 872–884.

Sowdhamini,R. and Blundell,T.L. (1995) An automatic method involving cluster analysis of secondary structures for the identification of domains in proteins. *Protein Sci.*, **4**, 506–520.

Sun,Z., Xia,X., Guo,Q. and Xu,D. (1999) Protein structure prediction in a 210-type lattice model: parameter optimization in the genetic algorithm using orthogonal array. *J. Protein Chem.*, **18**, 39–46.

Swindells,M.B. (1995) A procedure for detecting structural domains in proteins. *Protein Sci.*, **4**, 103–112.

Tarjan,R.E. (1972) Depth first search and linear graph algorithms. *SIAM J. Comput.*, **1**, 146–160.

Taylor,W. (1999) Protein structural domain identification. *Protein Eng.*, **12**, 203–216.

Tsai,C.J. and Nussinov,R. (1997) Hydrophobic folding units derived from dissimilar monomer structures and their interactions. *Protein Sci.*, **6**, 24–42.

Tsai,J., Taylor,R., Chothia,C. and Gerstein,M. (1999) The packing density in proteins: Standard radii and volumes. *J. Mol. Biol.*, **290**, 253–266.

Wernisch,L., Hunting,M. and Wodak,S.J. (1999) Identification of structural domains in proteins by a graph heuristic. *Proteins: Struct. Funct. Genet.*, **35**, 338–352.

Wetlaufer,D.B. (1978) Nucleation, rapid folding, and globular intrachain regions in proteins. *Proc. Natl. Acad. Sci. USA*, **70**, 697–701.

Xu,D., Tsai,C.J. and Nussinov,R. (1998) Mechanism and evolution of protein dimerization. *Protein Sci.*, **7**, 533–544.

Zehfus,M.H. and Rose,G.D. (1986) Compact units in proteins. *Biochemistry*, **25**, 5759–5765.