Chapter 6

Exact Protein Graph Alignment

6.1 Introduction

In this chapter we show how we can determine an exact graph-theoretical solution for the maximal common subgraph problem that has to be solved for the second stage of the GANGSTA method described in the previous chapter.

In many applications it is useful to compare objects represented as graphs in order to determine the degree and composition of the similarity. Commonly used techniques include graph matching or isomorphism techniques. Structure comparison or structure matching plays an important role in understanding the functional role of biological structures (see Chapters 3 and 5). This process is often reformulated into the problem of finding *maximal common subgraphs* (MCS) between graph representations of these structures. In chemical literature this referred as the maximal common substructure problem and denotes the largest substructure common to the collection of graphs. The graph-based similarity between graphs representing biological or chemical molecules plays an important role in many aspects of bio- or chemoinformatics: examples include protein-ligand docking [135], database searching [193], prediction of biological activity [80], reaction site modeling [11,159] or interpretation of molecular spectra [43,50]. The solution of the MCS problem is also of significant importance in many research areas outside of bioinformatics such as computer vision and image recognition [208]. The MCS problem is NP-complete [76] and we can therefore not hope to find an exact algorithm running in polynomial time. For a simple comparison of a pair of graphs having m and n vertices, the maximum number of vertex-by-vertex comparisons necessary to determine all common subgraphs of k vertices is according to Levi [144]

$$\frac{m!n!}{(m-k)!(n-k)!k!} \ ,$$

an astronomical number for non-trivial values of k, m, and n. Due to the NP-completeness exact algorithms designed to solve the MCS problem have a worst case, exponential-time complexity or are restricted to a finite class of graphs. Despite these restrictions, there exist algorithms that have proven to

be very efficient like maximum clique-based algorithms [144] or backtracking algorithms [159].

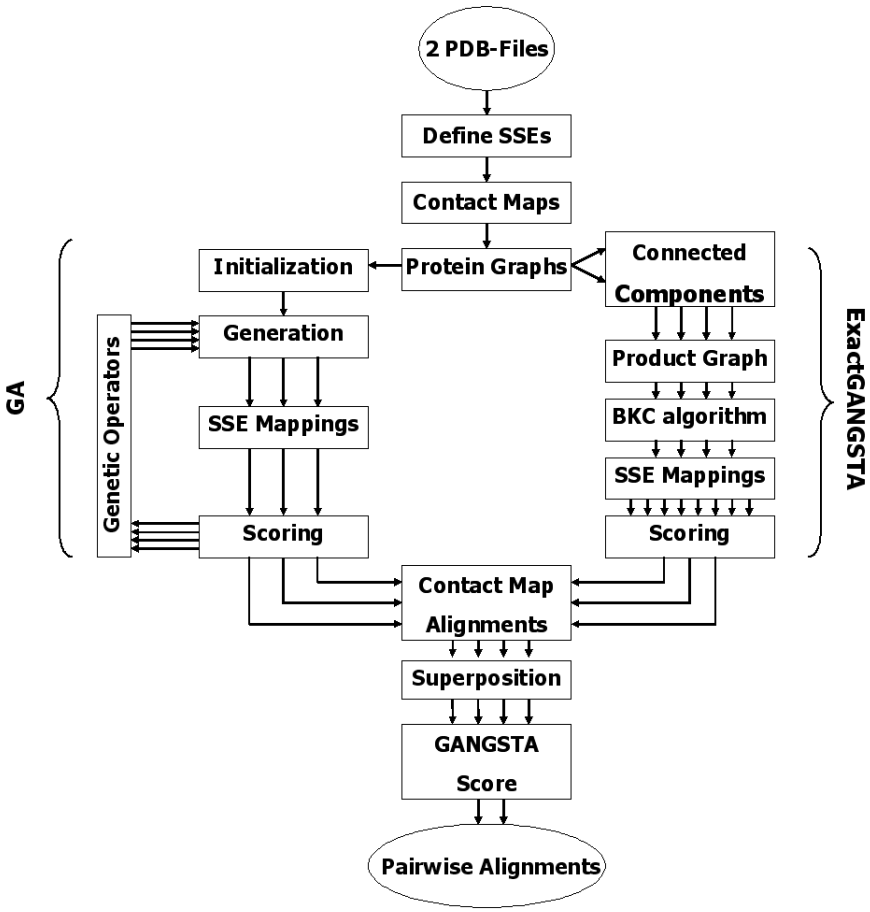

Figure 6.1: **The GANGSTA method: an overview.** The overall GANGSTA method with the two sub-workflows for the SSE alignment level, GA and ExactGANGSTA.

We introduce the ExactGANGSTA method for exact GANGSTA protein graph alignment on the secondary structure level. The ExactGANGSTA method extends the GANGSTA structure alignment method as described in the previous chapter by applying an exact MCS algorithm instead of the genetic algorithm for the SSE alignment level of the GANGSTA method. Figure 6.1 shows a schematic overview of the overall GANGSTA method for protein structure alignment involving a workflow with two sub-workflows, the GA as described in the previous chapter employing a heuristic search and ExactGANGSTA using an exact graph-theoretical algorithm for the same MCS problem. The GANGSTA method starts with two proteins represented as PDB [22] files including the atom coordinates of the protein structures as input. SSEs are defined using either DSSP [119] or Stride [74]. Then, for every protein structure GANGSTA protein graphs are defined according to Definition 18. If GANGSTA uses the GA for the SSE alignment level (Section 5.2.3) the SSE alignment problem is

encoded into individuals. An initial generation is generated and the resulting alignments are evaluated using the objective function (Equation 5.11). Genetic operators are applied on the individuals to produce new generations. The GA stops if the termination criterion is fulfilled (see Section 5.2.3) or the maximal allowed number of generations is reached. The best n alignments are passed on to the next level, the contact map alignment level.

Within the GANGSTA structure alignment method the ExactGANGSTA sub-workflow substitutes the genetic algorithm for the SSE alignment level by applying an exact MCS algorithm to search for maximal common substructures. Here, we will describe the graph-theoretical background, how we can search for maximal common substructures using graph-theory. The main idea is to search all maximal common subgraphs for all connected components (Definition 10) of the GANGSTA protein graphs. Each connected component represents a connected subgraph within a GANGSTA protein graph, called a *folding graph* ('connected components' in Figures 6.1 and 6.6). Then, the task is to search maximal common subgraphs between the folding graphs of the given protein graphs. This can be done transforming two folding graphs into an edge product graph and using a clique detection technique to determine all cliques ('product graph' and 'BKC algorithm' in Figure 6.1). Afterwards, the maximal common subgraphs have to be transformed into valid SSE alignments ('SSE mapping' in Figure 6.1). At the end of this chapter, we analyze some general properties of protein graphs and product graphs, and show that the GA algorithm is an excellent heuristic method to approximately solve the NP-complete MCS problem.

6.2 Graph-theoretical Methods

In this section we use the graph-theoretical terminology from [125, 193]. All referenced definitions throughout this section are given in the text or in the Appendix C. First, we define the graph-theoretical problem when searching for maximal common subgraphs (MCSs) in two arbitrary graphs, the MCS problem. Second, we show how we can transform the MCS problem into the clique problem using product graphs and how this can be done for GANGSTA protein graphs. Finally, we introduce the original Bron-Kerbosch algorithm [31] and a modified version of it to solve the clique problem efficiently [125].

6.2.1 The Maximal Common Subgraph Problem

In the following all graphs are simple undirected labeled graphs $G = (V, E)$ as defined in Definition 2. If two vertices $u, v \in V$ of G are connected by an edge $e \in E$ this is denoted by $e = (u, v)$ and the two vertices are said to be *adjacent*. The edge e is said to be *incident* to both vertices u and v.

A subgraph (Definition 43) of a graph G is a graph whose set of vertices and set of edges are all subsets of G. All the edges and vertices of G might not be present in the subgraph; but if a vertex is present in the subgraph it has a corresponding vertex in G and any edge that connects two vertices in the subgraph will also connect the corresponding vertices in G. An *induced subgraph* of a graph G is a subset of vertices of G and a set of edges of G with both endpoints in the subset:

Definition 24 (Induced Subgraphs)**.** *An* induced subgraph $G_s = (V_s, E_s) \subseteq G = (V, E))$ *is given, if* $\forall u, v \in V_s : (u, v) \in E \Rightarrow (u, v) \in E_s$.

Two graphs are said to be *isomorphic* (Definition 45) if there is a mapping between the vertices of both graphs such that neighbored pairs of vertices in the first graph are mapped onto neighbored pairs in the second graph. A *common subgraph* (Definition 46) of two graphs consists of a subgraph in the one graph and a subgraph in the other graph such that both subgraphs are isomorphic. The *maximal common subgraph* refers to the largest common subgraphs that cannot be extended:

Definition 25 (Maximal Common Subgraph (MCS))**.** *The* maximal common subgraph *of two graphs* G_1 *and* G_2 *is defined as the largest common subgraph: it must hold that if the pair* (G'_1, G'_2) *exists with* $G'_1 \subseteq G_1$, $G'_2 \subseteq G_2$ *and* $G'_1 \equiv G'_2$, *there exists no pair* (G''_1, G''_2) *with* $G'_1 \subseteq G''_1 \subseteq G_1$, $G'_2 \subseteq G''_2 \subseteq G_2$ *and* $G''_1 \equiv G''_2$.

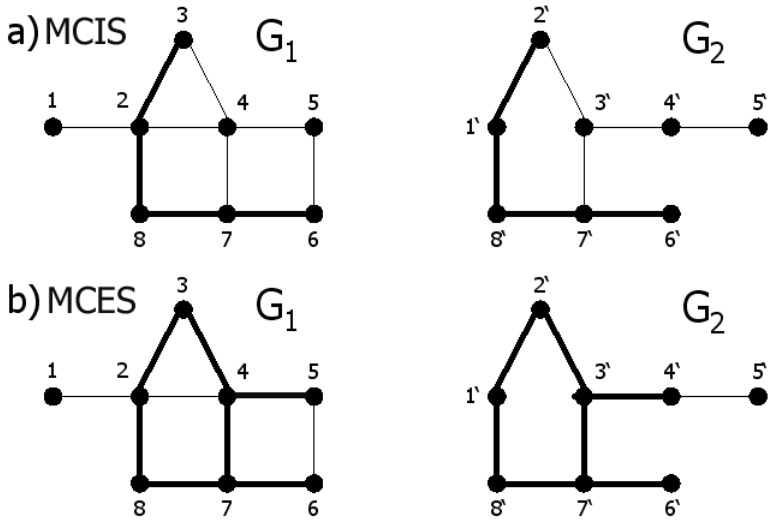

Figure 6.2: **Maximal common subgraphs.** a) Maximal common induced subgraph (MCIS). b) maximal common edge subgraph (MCES). Edges that are part of either the MCIS or the MCES are drawn in bold. The MCIS contains five vertices and four edges, the MCES seven vertices and seven edges. Example taken from [193].

A graph G_{12} is a *common induced subgraph* of two graphs G_1 and G_2 if G_{12} is isomorphic to one induced subgraph G'_1 of G as well as to one induced subgraph G'_2 of G_2. A *maximal common induced subgraph* (MCIS) of two given graphs G_1 and G_2 is defined as the common induced subgraph G_{12} with the maximum number of vertices. The *maximal common induced edge subgraph* (MCES) is defined as the common induced subgraph with the maximum number of edges common to the two given graphs. Figure 6.2 gives an illustration of two graphs G_1 and G_2 together with the MCIS and the MCES of both graphs. Here, the MCIS of G_1 and G_2 contains five vertices ((2,3,6,7,8) in G_1 and (1′,2′,6′,7′,8′) in G_2), whereas the MCES consists of seven vertices ((2,3,4,6,7,8) in G_1 and (1′,2′,3′,4′,6′,7′,8′) in G_2). The MCES is simply the common subgraph with

the largest number of edges. The MCIS in Figure 6.2(a) is less intuitive. It consists of the common subgraph with the largest number of vertices under the constraint that every edge present in G_1 that is incident to a vertex contained in the MCIS must also have a corresponding edge in the G_2 and vice versa. For instance, in the MCES vertex 4 in G_1 maps to vertex $3'$ in G_2, because edges $(3,4)$, $(4,5)$, and $(4,7)$ in G_1 correspond to edges $(2',3')$, $(3',4')$, and $(3',7')$ in G_2, respectively. In the MCIS, however, vertex 4 in G_1 does not match to vertex $3'$ in G_2, because the edge $(2,4)$ incident to 4 in G_1 but does not have a corresponding edge incident on vertex $3'$ in G_2.

Most of all MCS algorithms are solving only the subgraph isomorphism problem, i.e., they determine if one graph is contained within another graph, but this relation is often not unique, because often there exist more than one solution for this problem. In addition, a MCS between a pair of graphs is not necessarily unique as there may be more than one MCS. Therefore, the *MCS problem* for two given graphs is reformulated to the problem of finding all MCSs for the two graphs:

Definition 26 (MCS problem)**.** *Given two graphs* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *then the* MCS problem *searches for all MCSs* $H = (V_H, E_H)$ *such that* H *is isomorphic to* $G'_1 \subseteq G_1$ *and* $G'_2 \subseteq G_2$*, and appropriate isomorphic mappings* $f_1 : V_H \rightarrow V_1$ *and* $f_2 : V_H \rightarrow V_2$ *exist.*

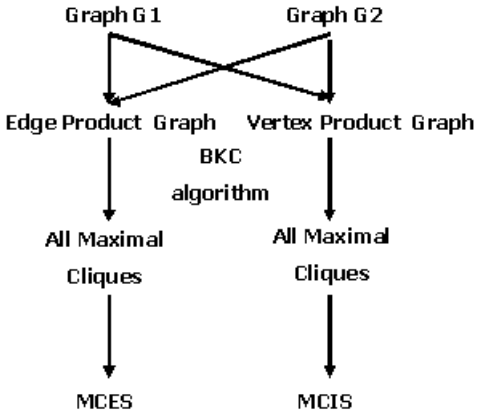

Figure 6.3: **The MCS problem.** Interdependencies of the different graph representations that are used to solve the MCS problem.

The MCS problem can be solved by transforming it into the *clique problem* (Definition 31), another NP-complete problem. In Figure 6.3 we give an overview on the graph-theoretical interdependencies of the proposed solution of the MCS problem. The transformation can be done constructing a *product graph* (Definitions 30 and 29) using the adjacency properties of the graphs being compared [144]. Product graphs are also known as association graphs in the image matching literature [184] and the compatibility graphs in the mathematical literature [23]. In the case of labeled graphs, e.g., protein graphs, the size of the product graph is further restricted by the prerequisite that vertex and

edge labels have to be considered. The product graph has the property that an MCS between the graphs being compared is equivalent to a clique in the product graph [144]. Therefore, MCESs in the original graphs correspond to cliques in the edge product graph. A clique (Definition 44) is a maximal complete subgraph of a graph in which every vertex is connected to every other vertex and which is not contained in any other larger clique, also known in literature as *maximal clique*. A clique is often also referred to as a simple complete subgraph, but not necessarily a maximal complete subgraph. Here, a clique denotes a maximal complete subgraph. A *maximum clique* is the largest clique present in the graph. Since there can exist various maximal complete subgraphs, we are not only interested in only the maximum clique, but in all maximal common subgraphs. Biologically, the maximum clique does not necessarily represent the best structure alignment. Therefore, we have to search all maximal cliques in the product graph. So, we can transform the MCS problem for two GANGSTA protein graphs into the clique problem for the edge product graph. All non-compatible edge pairs were excluded at the outset of this transformation. In the following, we describe how to find all MCESs between two protein graphs by finding all maximal complete subgraphs with more than three vertices in the edge product graph.

6.2.2 Transformation of the MCS Problem

The transformation of one algorithmic problem into another problem for which better or faster algorithms exist is a widely used technique in computer science. The MCS problem for two graphs can be reduced to the clique problem for a single graph. Levi [144] proposed this for the calculation of MCISs. Here, we use for the GANGSTA protein graphs the same description as Koch [126], who has calculated the transformation also for MCESs.

Let us consider two undirected, labeled GANGSTA protein graphs $PG_1 = (V_1, E_1, f_T, f_L, f_C, f_O)$ and $PG_2 = (V_2, E_2, f_T, f_L, f_C, f_O)$ as defined in Definition 18 with n and m vertices, respectively. Certain parts of both graphs can be mapped onto another, i.e., certain vertices and edges of graph PG_1 are *compatible* with certain vertices and edges of PG_2. All possible compatibilities can be stored in a graph, the so-called *product graph* or *compatibility graph*. Since in the GA these compatibilities are directly encoded in the used objective function (Equation 5.11) we have to define vertex and edge compatibility, respectively, to encode the SSE alignment as used in the GA (see Section 5.2.2) into an exact MCS search:

Definition 27 (Vertex Compatibility)**.** *A pair of vertices (u_1, u_2) with $u_1 \in V_1$ and $u_2 \in V_2$ is said to be compatible, if and only if u_1 and u_2 satisfy both the SSE type criterion (Definition 19) and the SSE length criterion (Definition 20).*

For the edge compatibility we cannot use the *contact number* criterion (Definition 21) and the *minimal orientation mismatch* criterion (Definition 22) directly, because they are part of the optimized objective function:

Definition 28 (Edge Compatibility)**.** *Two edges, $e_1 = (u_1, v_1) \in E_1$ and $e_2 = (u_2, v_2) \in E_2$, are compatible, if and only if both edges e_1 and e_2 satisfy the following properties:*

$$|f_C(e_1) - f_C(e_2)| \leq CD \tag{6.1}$$

where CD is the maximal allowed difference between numbers of residue contacts between two SSEs. For $x = f_O(e_1)$ *and* $y = f_O(e_2)$ *one of the following Boolean expressions have to be* true*:*

$$x = y \qquad \text{or}$$
$$(x \neq y) \wedge (x = X \vee y = X) \quad ,$$

i.e., two orientations are compatible if both orientations are equal or one of the orientations is a mixed orientation.

We can distinguish two types of product graphs: the *vertex product graph* and the *edge product graph*. The vertex product graph can be used to search for MCISs, whereas the edge product graph can be used to find MCESs.

Definition 29 (Vertex Product Graph)**.** *The vertex product graph* $G_V = PG_1 \circ_v PG_2$ *includes the vertex set* $V_V \subseteq V_1 \times V_2$ *that is defined by all vertex pairs* (u_i, u_j) *with* $u_i \in V_1$ *and* $1 \leq i \leq n$ *and* $u_j \in V_2$ *and* $1 \leq j \leq m$ *that are compatible.*

An edge between two vertices $u_V, v_V \in V_V$ *with* $u_V = (u_1, u_2)$ *and* $v_V = (v_1, v_2)$ *with* $u_1, v_1 \in V_1$ *and* $u_2, v_2 \in V_2$ *exists, if* $u_1 \neq v_1$ *and* $u_2 \neq v_2$*, and if the two edges* $e_1 = (u_1, v_1) \in E_1$ *in* PG_1 *and* $e_2 = (u_2, v_2) \in E_2$ *in* PG_2 *exist with* e_1 *and* e_2 *being compatible, or if* u_1*,* v_1 *and* u_2*,* v_2 *are not adjacent in* PG_1 *and* PG_2*, respectively.*

If some vertices $(u_1, v_1), \ldots, (u_k, v_k)$ are pairwise adjacent in the vertex product graph, the subgraph in PG_1 induced by the vertices $u_1,\ldots,u_k$ is isomorphic to the subgraph in PG_2 induced by the vertices $v_1,\ldots,v_k$. Consequently, the MCIS corresponds to a maximal complete subgraph in the vertex product graph G_V, as proven by Levi [144].

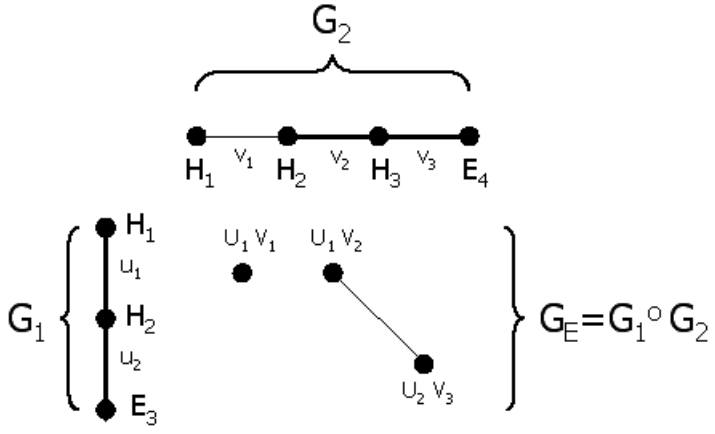

Figure 6.4: **Edge product graph.** We show two graphs G_1 and G_2 as well as the resulting edge product graph G_E. Edges that are part of the MCESs are drawn in bold. For sake of clarify, all edges in G_1 and G_2 have identical labels and vertices have only one label (H or E).

The definition of an edge product graph is analogous to the definition of the vertex product graph:

Definition 30 (Edge Product Graph)**.** *The edge product graph* $G_E = PG_1 \circ_e PG_2$ *includes the vertex set* $V_E = E_1 \times E_2$ *consisting of all compatible edge pairs* (e_i, e_j) *with* $1 \leq i \leq |E_1|$ *and* $1 \leq j \leq |E_2|$ *that have additionally compatible vertices, i.e., for* $e_1 = (u_1, v_1)$ *and* $e_2 = (u_2, v_2)$ *one of the following Boolean expressions has to be* true*:*

$$u_1, u_2 \text{ are compatible } \wedge \text{ } v_1, v_2 \text{ are compatible} \quad \text{or}$$
$$u_1, v_2 \text{ are compatible } \wedge \text{ } v_1, u_2 \text{ are compatible } .$$

There exists an edge between two vertices $e_E, f_E \in V_E$ *with* $e_E = (e_1, e_2)$*,* $f_E = (f_1, f_2)$*,* $e_1, f_1 \in E_1$*, and* $e_2, f_2 \in E_2$*, if* $e_1 \neq f_1$ *and* $e_2 \neq f_2$*, and if either* (e_1, f_1) *are connected via a vertex* $v_1 \in V_1$*,* (e_2, f_2) *are connected via a vertex* $v_2 \in V_2$*, and* v_1 *and* v_2 *are compatible, or* (e_1, f_1) *and* (e_2, f_2) *are not connected over a common vertex in* PG_1 *and* PG_2*, respectively.*

Each vertex of the edge product graph corresponds to an edge pair (e, f) with $e \in PG_1$, $f \in PG_2$. If some vertices (e_1, f_1),...,(e_k, f_k) with $1 \leq k \leq |E_1||E_2|$ in the edge product graph are pairwise adjacent, then the subgraph in PG_1, represented by the edges e_1,...,e_k, is isomorphic to the subgraph in PG_2, represented by the edges f_1,...,f_k. This isomorphism is only valid if the subgraphs in PG_1 and PG_2 contain at least four edges [126, 239].

Figure 6.4 illustrates the edge product graph G_E of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. The edge product graph consists of the three vertices (u_1, v_1), (u_1, v_2), and (u_2, v_3) of compatible edge pairs from G_1 and G_2. The edges are compatible, because all edges and the corresponding vertices have identical labels, for example, for the vertex (u_2, v_3) in the edge product graph the vertex H_2 in G_1 is compatible with H_3 in G_2 and the vertex E_3 in G_1 is compatible with E_4 in G_2. In the edge product graph, the vertex (u_1, v_2) is adjacent to vertex (u_2, v_3), since the edges u_1 and u_2 are connected via vertex H_2 in graph G_1, the edges v_2 and v_3 are connected via H_3 in graph G_2, and H_2 in G_1 and H_3 in G_2 are compatible. Consequently, the vertex (u_1, v_1) is not adjacent to vertex (u_2, v_3), since the edges u_1 and u_2 are connected via vertex H_2 in G_1 but the edges v_1 and v_3 are not connected in graph G_2. It is also clear, that vertex (u_2, v_1) does not exist in the edge product graph G_E, because the vertices of edge u_2 in G_1 are not compatible with the vertices of edge v_1 in G_2. The largest clique in G_E is the subgraph consisting of the two vertices (u_1, v_2) and (u_2, v_3). This clique corresponds to the MCES of G_1 and G_2 consisting of the edges u_1, u_2 in G_1 and the edges v_2, v_3 in G_2.

It can be argued that the MCES more adequately describes the structural similarity between two protein graphs than the MCIS does, since the interactions between residues and SSEs are mostly responsible for the overall 3D structure. Surprisingly, most of the methods in bioinformatics or chemoinformatics are searching for MCISs [13,92] instead of MCESs. Since we are basically interested in MCESs, we consider for the rest of this section only the edge product graph.

6.2.3 The Clique Problem

The clique problem is important for a variety of other applications. Therefore, a lot of algorithms have been developed to solve it for arbitrary graphs having exponential runtime [17, 20, 221] or for special graphs with polynomial runtime [77, 89, 107]. For many applications, like the protein graph matching

defined for GANGSTA (see Section 5.2), *all* MCSs in two graphs are required, i.e., all cliques in a graph have to be enumerated. Based on the fastest and most widely used algorithm for the clique problem, the Bron-Kerbosch algorithm [31], Koch *et al.* [125,126,129] developed a modified version that we use to search for all MCESs of two GANGSTA protein graphs as defined in Definition 18. The modified Bron-Kerbosch algorithm considers only those graphs that represent connected components (Definition 10). Thus, runtimes are drastically reduced making it possible to consider large protein graphs using reasonable amount of time and space resources.

The *clique problem* is known to be NP-complete [76]. It can be defined like in [126]:

Definition 31 (The Clique Problem)**.** *Given a graph $G = (V, E)$ and $0 \leq k \leq |V|$ then the clique problem returns true, if there exists a complete subgraph $H = (V', E')$ of G with $V' \subseteq V$ and $E' \subseteq E$ of size k or larger, and false otherwise.*

The definition holds true for complete and maximal complete subgraphs, because if a complete subgraph exists there must exist at least one maximal complete subgraph. Since we are interested in the more complex problem of enumerating all cliques in a graph, we have to define the *all-clique problem*:

Definition 32 (All-Clique Problem)**.** *Given a graph $G = (V, E)$ search for all maximal complete subgraphs $H = (V', E')$ of G with $V' \subseteq V$ and $E' \subseteq E$.*

In contrast to the clique problem which is NP-complete the all-clique problem is NP-hard, because the number of maximal complete subgraph is exponentially high. The enumeration of the all-clique problem of a graph can be done by enumeration algorithms like the Bron-Kerbosch algorithm (*BK-algorithm*), which recursively enumerates all cliques in a graph exactly once. The BK-algorithm is reported as the fastest enumeration algorithm to solve the all-clique problem [126]. First, we shortly describe the original BK-algorithm [31] that serves as basis for the subsequent work by Koch [126], which searches for maximal connected common subgraphs.

The Original BK-Algorithm

The original BK-algorithm finds all cliques in a given graph $G = (V, E)$ only once. It is a recursive tree search using a branch-and-bound search strategy. The search starts at the root vertex with the complete graph. The formation of new vertices in the search tree involves selectively choosing graph vertices that can form potential cliques. Generally, at each vertex of the search tree, the algorithm defines three sets C, P, and S of graph vertices, such that:

- C contains the set of vertices belonging to the currently potential clique.
- P contains all vertices, which are possible candidates for selection and addition to C, because they are adjacent to the currently selected vertex in C.
- S contains all vertices, which are adjacent to the currently selected vertex in C, but which have already appeared in previous cliques and thus must no longer be used for the completion of C. This avoids finding cliques or subgraphs of cliques that have been found previously.

Table 6.1: **BK-algorithm** The variant of the Bron-Kerbosch algorithm [31] that searches all cliques in a given graph, from [126].

BK (C, P, S)
$\triangleright$ enumerates all cliques in an arbitrary graph G
C: set of vertices belonging to the current clique
P: set of vertices which can be added to C
S: set of vertices which are not allowed to be added to C
$N[u]$: set of vertices adjacent to vertex u in G

```
01  Let P be the set {u_1, ..., u_k};
02  if P = ∅ and S = ∅
03    then  CLIQUE_FOUND;
04    else  for i ← 1 to k
05                do  P ← P\{u_i};
06                    P' ← P;
07                    S' ← S;
08                    N ← {v ∈ V|{u_i, v} ∈ E};
09                    BK (C ∪ {u_i}, P' ∩ N, S' ∩ N);
10                    S ← S ∪ {u_i};
11                od;
12  fi;
```

The BK-algorithm (see Table 6.1) is defined by a subroutine BK that has three sets as input parameters. The algorithm starts with the empty set C and S. Initially, $P = V$ includes all vertices of the graph G. The depth-first search is done by forming the sets C, P, and S at each level of the search tree. If P and S are empty, a clique is found and will be reported (line 03). Besides, each vertex of P is considered in a loop (lines 04-11), where the arbitrarily chosen vertex u_i is eliminated from P. P and S are copied into P' and S' (lines 06-07) for the recursion. The neighbors of vertex u_i are generated and stored in the set N (line 08). Then, vertex u_i is added to C and the subroutine call is done with $P' \cap N$ and $S' \cap N$ as new parameters (line 09). BK enumerates without duplication all vertex sets of all cliques of the edge product graph. Figure 6.5 shows the recursion tree for a graph G consisting of two maximal cliques given by the vertices (1,2,3) and (1,2,4,5). The example is taken form [125]. The vertices $u \in P$ are used in increasing order for the completion of C . Here, the cliques were found during the first steps, because the vertex 1 added first to C is a member of all cliques in G. The BK algorithm reports the two maximal complete subgraphs just once.

The Modified BK-Algorithm

A clique in the edge product graph G_E represents not only a pair of identical MCSs but an *automorphism* between this pair.

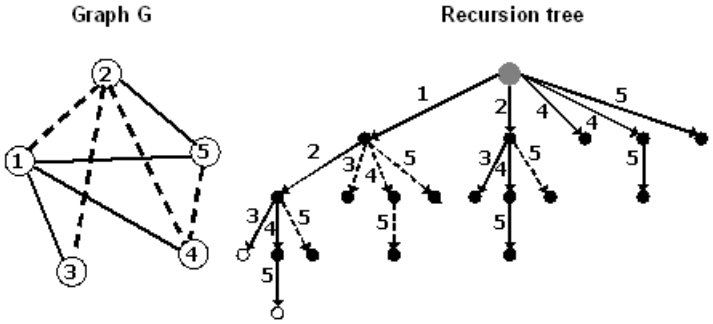

Figure 6.5: **The BK and the BKC algorithm.** The recursion tree for the BK and the BKC algorithm (right) for a Graph G (left). For the BK algorithm there is no distinction of different edge types in G. For the BKC algorithm *c-edges* are drawn as dotted lines. The edges of the recursion tree are labeled by vertex u added to the current set C. The gray vertex is the root of the recursion tree. Paths from the root to a white leaf vertex describe cliques in G. For the BK algorithm all paths from the root to a black leaf vertex describe complete subgraphs in G, which are not maximal, because they can be extended by at least on vertex in the non-empty set S. For the BKC algorithm, only the paths consisting of solid edges have to be considered. Here, paths from the root to a black leaf vertex describe complete subgraphs, which are spanned by *c-edges*, but which are not maximal and therefore do not represent cliques. Example is reproduced from [125].

Definition 33 (Automorphism)**.** *An* automorphism *of a graph is defined as a graph isomorphism with itself, i.e., a mapping from the vertices of the given graph G back to vertices of G such that the resulting graph is isomorphic with G. The sets of automorphisms define a permutation group. The automorphism groups of a graph characterize its symmetries.*

Thus, even though each clique is reported just once, the BK-algorithm output sets of cliques representing the same pair of identical substructures but the vertices in the subgraphs just permutated. Analyzing the symmetries of subgraphs and thereby reporting cliques from the same automorphism groups reduces the size of the output and thereby reduces runtimes. Unfortunately, up to now there is no efficient way to do this. Large automorphism groups exist for MCSs that consist of different disconnected MCSs. When we search for connected MCSs in a connected graph, the problem will be simplified such that the MCSs, which consist of different disconnected subgraphs, must not be considered in the recursion tree during search. For the realization of a BK-algorithm variant that only considers maximal connected common subgraphs Koch [126] divided the edges in the edge product graph G_E into *c-edges* (connected edges) and *d-edges* (disconnected edges). They are labeled according to their division.

Definition 34 (c-Edges and d-Edges)**.** *Let* (e_1, e_2) *and* (f_1, f_2) *with* $(e_1, f_1) \in E_1$ *and* $(e_2, f_2) \in E_2$ *be two edge pairs of the two graphs* PG_1 *and* PG_2 *representing two vertices in the edge product graph* G_E*. An edge of the edge product graph* G_E *between the vertices* (e_1, e_2) *and* (f_1, f_2) *is called* c-edge *if the edges* (e_1, f_1) *in* PG_1 *and* (e_2, f_2) *in* PG_2 *exhibit a common vertex, otherwise the edge*

is called d-edge.

Table 6.2: **BKC-algorithm.** An algorithm for the detection of all c-cliques based on the BK-algorithm (Table 6.1).

BKC (C, P, D, S)
$\triangleright$ enumerates all c-cliques in an arbitrary graph G
C: set of vertices belonging to the current c-clique
P: set of vertices which can be added to C, because they are adjacent to the current vertex u via a c-edge
D: set of vertices which cannot directly be added to C, because they are adjacent to the current vertex via a d-edge
S: set of vertex which are not allowed to be added to C
$N[u]$: set of adjacent vertices of vertex u in G

```
01  Let P be the set {u_1, ..., u_k};
02  if P = ∅ and S = ∅
03    then  CLIQUE_FOUND;
04    else  for i ← 1 to k
05              do  P ← P\{u_i};
06                  P' ← P;
07                  D' ← D;
08                  S' ← S;
09                  N ← {v ∈ V|{u_i, v} ∈ E};
10                  for all v ∈ D'
11                      do  if v and u_i are adjacent via c-edge
12                            then  P' ← P' ∪ {v};
13                                  D' ← D'\{v};
14                            fi;
15                      od;
16                  BKC (C ∪ {u_i}, P' ∩ N, D' ∩ N, S' ∩ N);
17                  S ← S ∪ {u_i};
18              od;
19  fi;
```

Consequently, we search in the edge product graph for so-called *c-cliques*. A clique in the edge product graph specifies a connected common complete subgraph, if for each pair of vertices of the clique, there exists a path that consists only of c-edges. Equivalently, if all d-edges are removed from the clique then the resulting graph is still complete and connected. Therefore, we can define *c-cliques* as follows:

Definition 35 (c-Clique)**.** *A clique in an edge product graph G_E that consists of c- and d-edges is called a* c-clique*, if it is formed by c-edges such that it is connected and acyclic.*

Koch [126] showed that a *c*-clique in the edge product graph is a clique that represents connected MCSs in the graphs PG_1 and PG_2 that form the edge

product graph. Thus the all-clique problem can be reformulated into the *all c-clique problem*:

Definition 36 (All-c-Clique Problem)**.** *Given an edge product graph* $G_E = (V_E, E_E)$ *with* $E_E = C \cup D$*. Let* C *be the set of all c-edges in* G_E*, and* D *the set of all d-edges in* G*. Then the* all c-clique problem *can be defined as to search all maximal complete subgraphs* G'_E *whose c-edges span the graph* G'_E*.*

To solve the all-c-clique problem the BK-algorithm (Table 6.1) has to be modified (see Table 6.2) now having four sets as parameters. The original set P is now divided into two sets P and D such that P contains only vertices, which are adjacent to at least one vertex of C via a c-edge, and D contains only vertices, which are not adjacent to any vertex in C via a c-edge. As mentioned before, C representing the current clique is extended only by those vertices $u \in P$ that are adjacent to at least one vertex of C via a c-edge in G. If a vertex $u \in P$ is selected, the vertices from D are checked whether they are adjacent to vertex u via a c-edge (lines 10-15). In this case, this vertex is eliminated from D and added to P (lines 13-14). In the recursion step the new set D is intersected with all the adjacent vertices of u added to the current clique C in the previous step. A c-clique is reported if P and S are empty, because if P is empty C cannot be extended, and if S is empty no vertex set that contains C is reported so far. The BKC algorithm cannot initially be started with the empty set C. Since each vertex of P has to be adjacent to some vertex in C by some c-edge, the P set would be also empty and leading the algorithm to its termination without starting. Instead, the BKC-algorithm can be started for each vertex $u \in G_E$ with the parameter list $(C = \{u\}, P, D, S = \emptyset)$. Figure 6.5 shows the decrease of the number of steps in the recursion tree by solving the *all-c-clique problem*, because only the paths were traversed consisting of solid edges.

6.3 The ExactGANGSTA Method

The ExactGANGSTA method has two proteins as inputs that are represented as GANGSTA protein graphs (see Definition 18 and Figure 6.1), PG_1 and PG_2. Since the BKC algorithm introduced in the last section works only on connected graphs, we first have to determine all connected components (see Definition 10) of the two protein graphs PG_1 and PG_2 . The resulting connected components are called *folding graphs* and are denoted by FG_i^1 with $1 \leq i \leq n_1$ for PG_1 and FG_j^2 with $1 \leq j \leq n_2$ for PG_2, where n_1,n_2 representing the number of connected components in PG_1 and PG_2, respectively. We have to build $n_1 \times n_2$ edge product graphs G_{ij}^e according to Definition 30 with $i \in (1, \ldots, n_1)$ and $j \in (1, \ldots, n_2)$, because we want to determine all maximal common subgraphs between all connected components of each graph. Each edge product graph G_{ij}^e is built of the constituting folding graphs FG_i^1 and FG_j^2 from PG_1 and PG_2. Then, for each G_{ij}^e the BKC algorithm searches all cliques C_k^{ij} with $1 \leq k \leq l_{ij}$ and l_{ij} the number of cliques for edge product graph G_{ij}^e. It may be desirable to consider only larger cliques, as there are often many cliques of small sizes that do not convey any great structural significance. The method finds all cliques located from size two upwards, i.e., any common subgraphs containing at least two vertices. If a minimum clique size of, for instance, six has been specified, then no cliques of size less than six will be output.

To build a maximal GANGSTA-SSE-alignment defined in Definition 23, i.e., a GANGSTA-SSE-alignment with the maximal possible number of aligned SSEs, the cliques have to be transformed into the maximal common subgraphs within their constituting folding graphs. Each clique can also be interpreted as a sub-alignment or a sub-mapping:

Definition 37 (Sub-Alignment)**.** *A* sub-alignment *is defined as a clique* $C_k^{ij} \subseteq G_{ij}^e$ *in the edge product graph* G_{ij}^e *that represents the two maximal common subgraphs* $FG_i'^1 = (V_i'^1, E_i'^1) \subseteq FG_i^1 \subseteq PG_1$ *and* $FG_j'^2 = (V_j'^2, E_j'^2) \subseteq FG_j^2 \subseteq PG_2$*. Then there exist a bijective mapping* $m_s : V_i'^1 \rightarrow V_j'^2$ *with the property that, for every* $v_j'^2 \in V_j'^2$*, there is exactly one* $v_i'^1 \in V_i'^1$ *such that* $m_s(v_i'^1) = v_j'^2$*.*

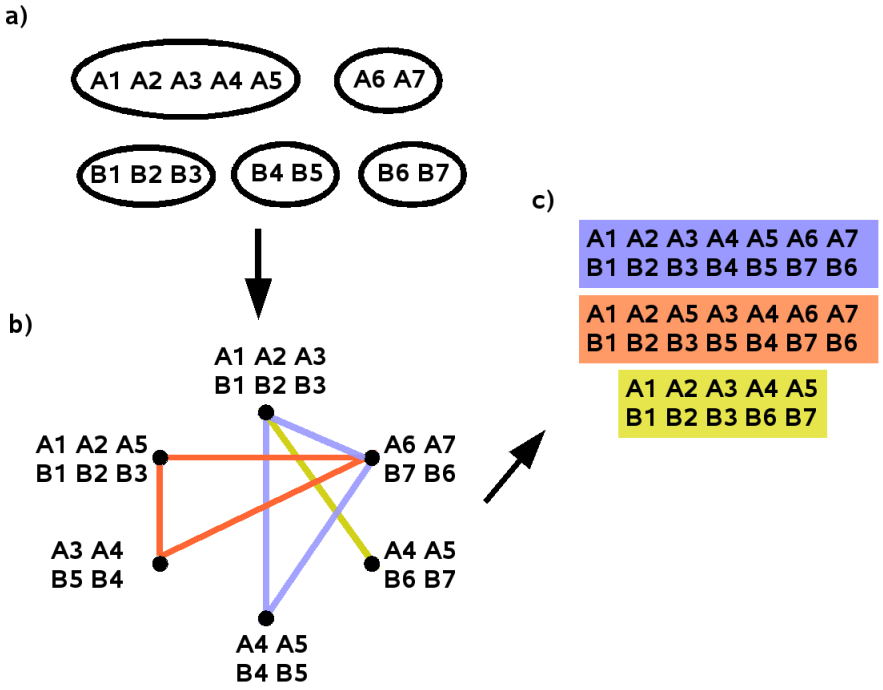

Figure 6.6: **The ExactGANGSTA method: an overview.** a) Connected components of two protein graphs A and B are illustrated in ellipses (only the vertices representing the SSEs are shown). SSEs are numbered from the N-to C-terminus. b) Consistency graph: Each vertex represents one sub-alignment. An edge is drawn if two sub-alignments are consistent. Cliques are indicated by different colors. c) Each clique represents one possible maximal GANGSTA-SSE alignment.

Then, the task is to generate all maximal GANGSTA-SSE-alignments by combining all sub-alignments. This can be done by searching cliques in a consistency graph defined for all sub-alignments. Therefore, we have to define first the consistency of two sub-alignments:

Definition 38 (Sub-Alignment Consistency)**.** *Given two sub-alignments consisting of the two maximal common subgraphs* $FG_i'^1 = (V_i'^1, E_i'^1) \subseteq FG_i^1 \subseteq PG_1$ *and* $FG_j'^2 = (V_j'^2, E_j'^2) \subseteq FG_j^2 \subseteq PG_2$ *for the first sub-alignment and* $FG_k''^1 =$

$(V_k''^1, E_k''^1) \subseteq FG_k^1 \subseteq PG_1$ *and* $FG_l''^2 = (V_l''^2, E_l''^2) \subseteq FG_l^2 \subseteq PG_2$ *for the second sub-alignment. Two* sub-alignments *are said to be* consistent *if and only if there exists bijective mapping* $m_c : V_i'^1 \cup V_k''^1 \rightarrow V_j'^2 \cup V_l''^2$.

Note that, for example, the two maximal common subgraphs $FG_i'^1$ and $FG_k''^1$ can come for $(i = k)$ from the same connected component $FG_i^1 = FG_k^1$ in PG_1. Afterwards, we can calculate the pairwise consistencies for all sub-alignments and store these consistencies in the so-called *consistency graph*:

Definition 39 (Consistency Graph)**.** *A* consistency graph G_c *includes the vertex set* V_c *consisting of all sub-alignments for the two protein graphs* PG_1 *and* PG_2*. An edge between two vertices* $v_i, v_j \in V_c$ *with* $i \neq j$ *exists if the two sub-alignments* v_i, v_j *are consistent according to Definition 38.*

Then, a maximal GANGSTA-SSE-alignment is represented as a clique in the consistency graph G_c for the two protein graphs PG_{src} for the *source* protein and PG_{targ} for the *target* protein and additional gaps for not aligned vertices from PG_{src}:

Definition 40 (Maximal GANGSTA-SSE-Alignment)**.** *A* maximal GANGSTA-SSE-alignment *for* $PG_{src} = (V_{src}, E_{src})$ *and* $PG_{targ} = (V_{targ}, E_{targ})$ *consists of a maximal complete subgraph* $CL = (V_{CL}, E_{CL}) \subseteq G_c = (V_c, E_c)$ *in the consistency graph for the* PG_{src} *and* PG_{targ} *and, additionally, the gap mapping* $m_g : V_{src} \setminus V_{CL} \rightarrow \ '-'$.

All maximal GANGSTA-SSE-alignments can be found solving the all-clique problem for the consistency graph using the BK-algorithm (see Table 6.1). Every maximal GANGSTA-SSE-alignment is evaluated using the objective function (Equation 5.11). Then, the best n alignments according to the objective function are passed on to the next level, the contact map alignment level, like shown in Figure 6.1. From here on, the ExactGANGSTA method is identical to the GANGSTA method as described in the previous chapter.

Figure 6.6 gives an illustration of the ExactGANGSTA method. Here, the first protein graph A contains seven vertices and two connected components, $(A1, A2, A3, A4, A5)$ and $(A6, A7)$, and the second protein graph consists of seven SSEs in three connected components, $(B1, B2, B3)$, $(B4, A5)$, and $(B6, B7)$ (Figure 6.6a). Using the BKC-algorithm we get six different sub-alignments representing the vertices of the consistency graph. For example, the sub-alignment $(A1, A2, A5) \rightarrow (B1, B2, B3)$ is consistent with $(A3, A4) \rightarrow (B5, B4)$, because there exists a bijective mapping of $m_c : \{A1, A2, A5\} \cup \{A3, A4\} \rightarrow \{B1, B2, B3\} \cup \{B4, B5\}$. Then, a clique in the consistency graph represents a maximal GANGSTA-SSE-alignment. The consistency graph in Figure 6.6b contains three cliques of minimum size 2. The corresponding maximal GANGSTA-SSE-alignments are shown in Figure 6.6c.

6.4 Results

The ExactGANGSTA method is together with the BK and the BKC algorithm part of the GANGSTA C++ implementation. Here, we report on results obtained by running ExactGANGSTA in comparison to the GA for the second stage of the GANGSTA method (see Figure 6.1). Additionally, we analyze some general properties of protein graphs and product graphs on different datasets.

6.4.1 Protein Graph Properties

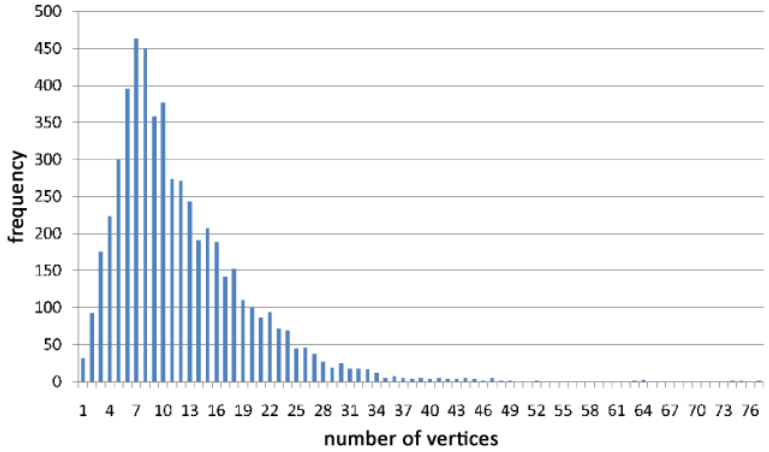

Figure 6.7: **Protein graph vertex distribution.** Columns give the number of vertices of different sizes in protein graphs for ASTRAL Scop40 dataset. SSEs defined by Stride [74]

To analyse general properties of protein graphs we use two different graph-theoretical parameters. First of all, there is the size the graph $G = (V, E)$, given by the number of its vertices $|V|$ and edges $|E|$. Second, there is the density of a graph given by the ratio of the number of edges and the number of vertices:

Definition 41 (Graph Density)**.** *The* graph density *for an undirected graph* $G = (V, E)$ *is defined as* $D = \frac{2|E|}{|V|(|V|-1)}$ *, where* $|E|$ *denotes the number of edges and* $|V|$ *the number of vertices (this definition only works if* $|V| > 1$*; we define* D *to be* 0 *if* $|V| \leq 1$*).*

An undirected graph can have at most $|V|(|V| - 1)/2$ edges, and hence the maximal density is 1. Graph G is a *sparse* graph, when $|E| \approx |V|$, and G is called *dense* if $|E| \approx |V^2|$ [187]. Graphs are commoly called dense, if $D > 0.5$.

Table 6.3: **Protein graph properties.** We show the maximal vertex degree, the maximal number of edges, and the mean graph density (Definiton 41) for the five contact types for all domains of the ASTRAL Scop40 dataset.

contact type	max vertex degree	max $\|E\|$	D
ca	8	235	0.57
cb	9	273	0.62
all	10	437	0.73
vor	6	208	0.50
vdW	6	182	0.43

We have investigated the ASTRAL Scop40 dataset (see Appendix D.1) using Stride [74] for SSE identification and the five different contact types defined in Section 2.7.3. The results are shown in Table 6.3 and Figures 6.7 and 6.8.

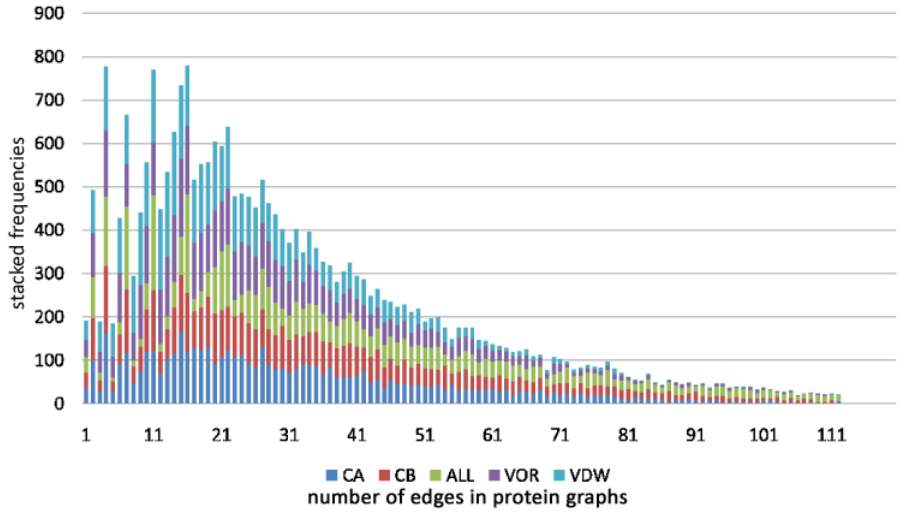

Figure 6.8: **Protein graph edge distributions.** Stacked columns for the five contact types as defined in Section 2.7.3. The number of edges in protein graphs is shown for edge sizes ranging from 0 to 110 edges per protein graph for domains of the ASTRAL Scop40 dataset (see Appendix D.1).

Protein graphs are limited in their size by nature. Single protein chains in the PDB [22] contain at most about 1000 SSEs. Here, we used protein domain definitions from SCOP [169]. Protein domains can be defined as compact portions of proteins, i.e., they can span whole protein chains or only parts of it. The vertices of the protein graphs for the ASTRAL Scop40 dataset represent the SSEs in the domains. The number of SSEs varies between 1 and 78 vertices, with a single peak at 6, and a long tail (Figure 6.7). We have excluded all domains from the ASTRAL SCOP40 datasets that showed no SSEs. The majority of the graphs have between 7 and 12 vertices; only few very large domains have over 30 SSEs. The maximal vertex degree, i.e. the number of incident edges, is 10. The edge distributions for the different contact types are given in Figure 6.8. Despite the *ca* contact type defines much more contacts than the *all* contact type, *all* contacts define more graphs with more edges. This contact type exhibits also the graph with the highest number of vertices, 437, for the domain $1muk A\_$, a RNA-dependent RNA polymerase, containing 1256 residues. The graph density is for all contact types relatively high, but most of the graphs have about 15 to 25 edges.

6.4.2 ExactGANGSTA versus GA

We compared the quality of pairwise GANGSTA alignments using the two SSE alignment methods, GA and ExactGANGSTA, for 70 pairwise alignments from the Fischer dataset [70] as described in the Appendix D.9. We considered for each $reference$ structure the first $target$ structure in Table D.4. The alignments were produced using the *ca* contact type and Stride [74] for SSE assignment. Only for 35 of the 70 pairwise structure alignment the ExactGANGSTA method could find a solution within 15 hours (44,000s). The results for these 35 suc-

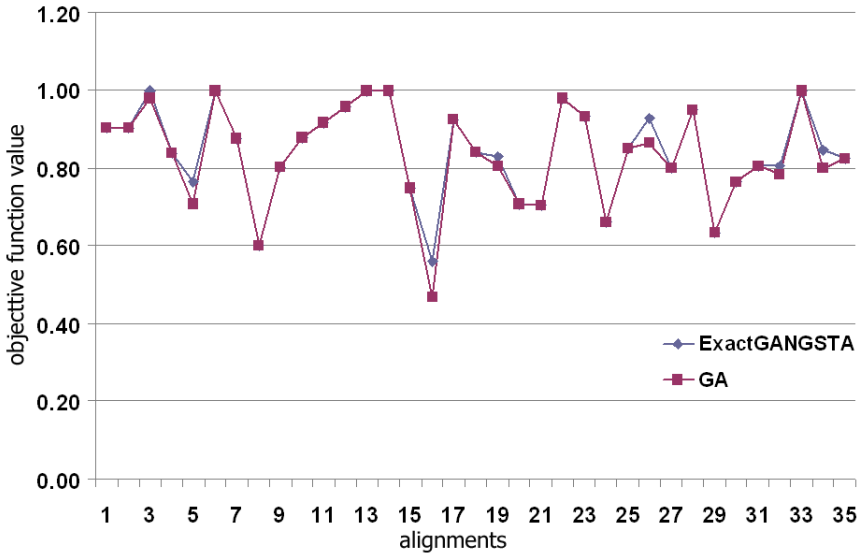

Figure 6.9: **ExactGANGSTA versus GA: Objective function.** The objective function (Equation 5.11) distribution from the 35 successful pairwise comparisons using the ExactGANGSTA and the GA method for the second stage of the GANGSTA method.

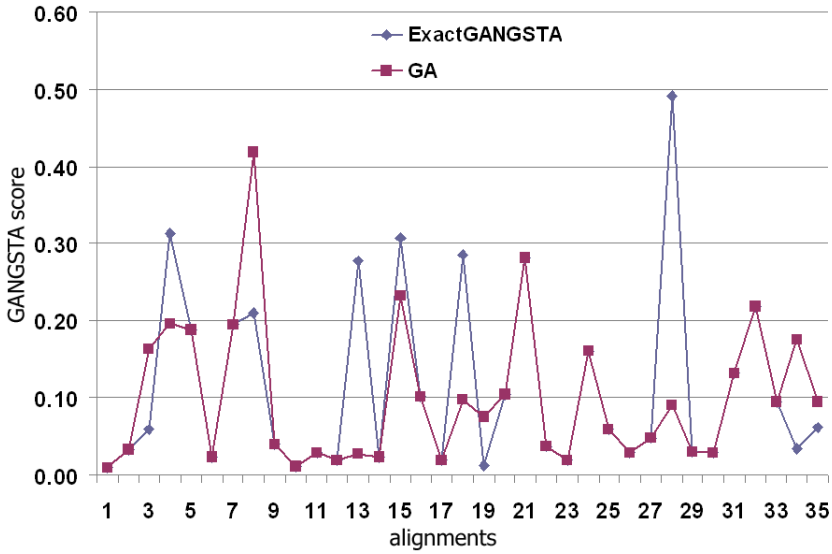

Figure 6.10: **ExactGANGSTA versus GA: GANGSTA *score*.** GANGSTA *score* (Equation 5.16) from the 35 successful pairwise comparisons using the ExactGANGSTA and the GA method for the second stage of the GANGSTA method.

Table 6.4: **Comparison GA versus ExactGANGSTA for the 35 successful pairwise alignments of the Fischer dataset [70] using *ca* contact type.** We show the mean values for the objective function (Equation 5.11), the GANGSTA *score* (Equation 5.16), and the runtime. Additionally, for the ExactGANGSTA method the mean number of cliques and the mean number of SSE alignments is shown.

alignment type	obj	GANGSTA *score*	runtime $[s]$	cliques	alignments
GA	0.8116	0.0973	1.57	-	-
ExactGA	0.8207	0.1077	2,207.89	118,866	105,423

cessful pairwise alignments are shown in Table 6.4 and Figures 6.9 and 6.10. As expected, ExactGANGSTA found for these alignments objective function values (Equation 5.11) that are better or equal to the values reported from the GA method demonstrating that the implementation of the ExactGANGSTA method is correct. In the GANGSTA method (see Section 5.2.3 and Figure 6.1) the objective function is the criterion to rank the best SSE alignments from the first level of the GANGSTA method. The best m SSE alignments according to the objective function are then evaluated in the second step of the hierarchy, the contact map overlap optimization step. Since the objective function is only a crude but fast heuristic for the quality of a GANGSTA structure alignment it shows imperfect correlation with the GANGSTA *score* (Equation 5.16) and the contact map overlap (Equation 17), i.e., the best SSE alignment according the objective function does not have to correspond to the best SSE alignment according to the GANGSTA *score*. For five of the 35 successful alignments, the ExactGANGSTA method reports SSE alignments corresponding to lower GANGSTA *scores* than the original GA method. Totally, the mean value of the GANGSTA *scores* is slightly worse for the ExactGANGSTA method than for the GA method. Within the GANGSTA hierarchy only the best m SSE alignments according to the objective function are further processed, i.e., the GA method can find SSE alignments corresponding to better GANGSTA *score* values but lower or equal objective function values than the SSE alignments found from the ExactGANGSTA method. A reason for this could be that we have designed the genetic operators of the original GA method to produce individuals that cover a wide range of the whole search space, i.e., they produce very distant, but still reasonable SSE alignments, whereas the ExactGANGSTA method only reports SSE alignments that correspond to the exact graph-theoretical solution. The exact graph-theoretic solution itself must not correspond to the best solution in terms of RMSD or contact map overlap that are the mean ingredients of the GANGSTA *score* calculation after the second stage of the GANGSTA method.

The runtime of both methods relative to the edge size of the product graph is illustrated in Figure 6.11. All runtime experiments were done on a Linux AMD Opteron 242 system, using one thread for the entire program including all initializations. For several pairwise alignments the ExactGANGSTA method found

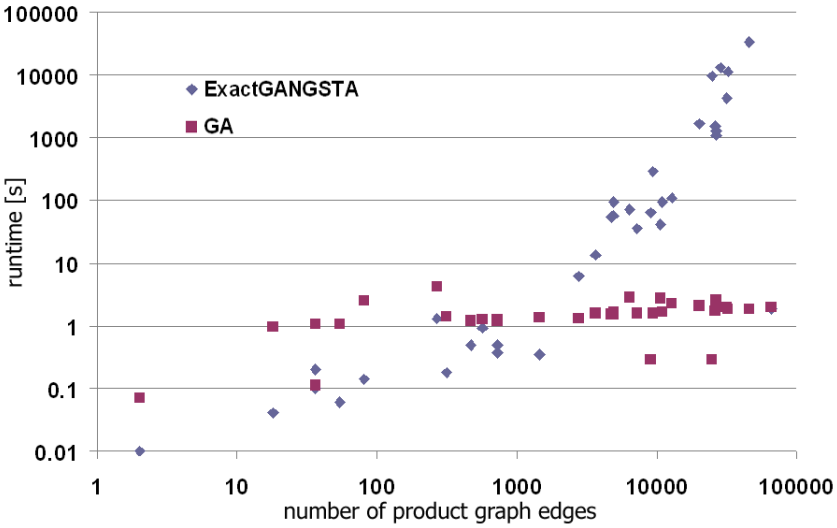

Figure 6.11: **ExactGANGSTA versus GA: edge number vs. runtime.** Runtime ([s]) as a function of the number of edges in the product graph. Data from the 35 successful pairwise comparisons using the ExactGANGSTA method. The runtime for the GA method is given for the corresponding alignment task. Both axis are scaled logarithmic.

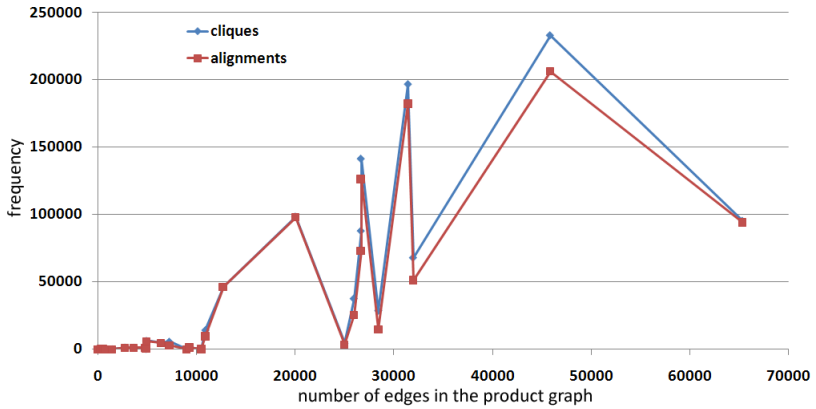

Figure 6.12: **ExactGANGSTA: cliques and alignments.** Numbers of cliques and number of alignments resulting from the 35 successful clique searches in the product graphs from the Fischer dataset as a function of edges in the edge product graph.

Table 6.5: **Product graph properties.** Comparison GA versus ExactGANGSTA for all 70 pairwise alignment tasks of the Fischer dataset [70]. The *max*, *mean*, and *min* numbers for the vertex size ($|V|$) and edge size($|E|$), respectively; *mean D* gives the mean value of the product graph density.

property	value
min $\|V\|$	4
mean $\|V\|$	3,567
max $\|V\|$	9,015
min $\|E\|$	2
mean $\|E\|$	1,757
max $\|E\|$	9,015
mean D	0.44

the best alignment according to the objective function in reasonable time: for product graphs of size 1,000 or smaller it clearly outperforms the GA method, and for product graphs of sizes of at most 10,000 vertices the ExactGANGSTA method is able to find the best alignment within 100 seconds. While the GA method runtime is nearly constant or slowly increasing for increasing problem sizes, the runtime for the ExactGANGSTA method increases nearly exponentially with respect to the logarithmic increase of the edge number. For large problem cases the GA method outperforms the exact solution drastically in runtime. The more the number of vertices and edges in the two input protein graphs increases, the more the number of vertices and edges in the resulting product graphs grows, resulting in runtime that exceed the range of 15 hours. The properties of the product graphs for all 70 alignment tasks are given in Table 6.5. Their size given as the number of vertices varies from 4 to 9,015 with a mean of 3,567 vertices. Edge product graphs for protein domains can get very large, because protein graphs are highly dense (see Table 6.3 and Figure 6.8), i.e., that they have often a large number of edges. As the size of the product graphs increases, the number of cliques and valid maximal GANGSTA-SSE-alignments (Definition 40) is also increasing (see Figure 6.12).

All protein graphs of the 35 successful pairwise alignments consisted of at most three connected components (Definition 10). In all these cases, the input protein graphs consisted only of one large connected component and one or two connected components of only a single vertex. Therefore, the consistency graph (see Figure 6.6) had only to combine subalignments from a single pair of connected components from each protein graph. The reason for this is that connected components of size one will not be used in the edge product graph, because there exist no edges.

Figure 6.12 shows the number of cliques and alignments the ExactGANGSTA method found for the 35 successful pairwise structure alignments. Since the different cliques can contain vertex sets from the same automorphism group (Definition 33), here, the number of unique optimal alignments is slightly lower than the number of cliques in the product graph (see Table 6.4 and Figure 6.12). Every exact alignment, also called a maximal GANGSTA-SSE-alignment (Definition 40), of the ExactGANGSTA method represents an exact solution for the MCS problem (see also Figure 6.6). The high number of exact solutions is due to

the relative abstract representation of proteins using protein graphs with SSEs as vertices and contacts between SSEs as edges and some additional constraints, so that the search space of valid SSE alignments combinatorial explodes. This is also another reason for the high runtime of the ExactGANGSTA method, because for all the maximal GANGSTA-SSE-alignments the objective function value has to be calculated to rank the alignments and to determine the m best alignments for the second stage of the GANGSTA method.

6.5 Discussion

We presented an exact graph-theoretical method called ExactGANGSTA, which searches maximal common substructures of two protein structures by solving the MCS problem for two GANGSTA protein graphs. The method solves the MCS problem by transforming it into the maximal clique search problem in edge product graphs. The resulting maximal common subgraphs are transformed into valid SSE alignments afterwards. Similar methods have been proposed before for protein structure alignment [125, 162] but either they have used the vertex product graph only [162] or they searched only for the presence of local structural motifs [125] without constructing a valid structural alignment in 3D. The ExactGANGSTA method is the first implementation of an algorithm to find the exact solution for the MCS problem in protein graphs providing global SSE mappings and using the edge product graph. For the application in protein structure alignment this approach is able to detect the exact optimal solution for arbitrary pairs of protein structures given sufficient runtime. The heuristic solution for the same graph-theoretical problem, the GA method, as described in the previous chapter, runs in nearly constant runtime for all problem sizes and produces SSE alignments with objective function values that are for most of the problem instances equal or slightly worse than the exact solutions from the ExactGANGSTA method. As the size of the protein graphs and the considered product graphs increases, the GA method clearly outperforms the ExactGANGSTA method in terms of runtime. Therefore, the ExactGANGSTA method will only be applicable to small or medium range problem instances. For those problem instances, the ExactGANGSTA method outperforms the GA method in runtime and is therefore the better choice as search algorithm. It may be possible to significantly improve the performance of the ExactGANGSTA method by incorporating more constraints on the product graph generation or on pruning the search tree of the BKC algorithm. Additionally, the combinatorial large number of valid exact solutions could be reduced. These possibilities remain for future work on the algorithm, as does its application to more difficult problem instances. The demonstration of the efficiency of the heuristic GA algorithm with its ability to ignore the sequence order of SSEs gives it immense advantages in routine scanning of protein structure databases. Therefore, the GA method within the overall GANGSTA structure alignment method provides a unique tool for the investigation of non-trivial structural resemblances in proteins, as demonstrated in the previous chapter.