Complete Enumeration of Compact Structural Motifs in Proteins

Bhadrachalam Chitturi
UTSW Medical Center at Dallas
5323 Harry Hines Boulevard
Dallas, Texas 75390
(+1) 214-645-5932
bchitt@mednet.swmed.edu

Doina Bein
Applied Research Laboratory
Pennsylvania State University
University Park, PA 16802
(+1) 814-863-4397
siona@psu.edu

Nick V. Grishin
UTSW Medical Center at Dallas
5323 Harry Hines Boulevard
Dallas, Texas 75390
(+1) 214-645-5952
grishin@chop.swmed.edu

ABSTRACT
The search of structural motifs that specify the spatial arrangement of polypeptide segments is preferred over other methods such as common substructure discovery and structural superposition in comparing protein structures. 3D protein structures can be modeled as graphs whose maximum degree is bounded by a constant. Structural motifs can also be modeled as graphs and a significant percentage of them are trees. Thus, motif search in proteins can be modeled as an enumeration of isomorphic subgraphs where a query tree Q with m nodes is searched in a sparse graph G with n nodes and the maximum degree of any node in G is bounded by a constant ε. We design an efficient divide-and-conquer algorithm that finds all copies of Q in G by partitioning Q using a minimum dominating set. This strategy can be extended to sparse query graphs that can be reduced to trees by deleting a small number of edges.

Categories and Subject Descriptors
I.1.2 [**Symbolic and algebraic manipulation**]: *Algorithms*, *Analysis of algorithms, Nonalgebraic algorithms.* F.2.2 [**Analysis of algorithms and problem complexity**]: *Nonnumerical algorithms and problems – Computations on discrete structures, sorting and searching.* G.2.2 [**Discrete Mathematics**]: *Graph theory – Graph algorithms.*

General Terms
Algorithms, Performance, Design, Experimentation, Theory.

Keywords
Complete subgraph enumeration, divide-and-conquer, dominating set, motif, protein structure.

1. INTRODUCTION
Theoretically well studied problem of subgraph isomorphism has gained large interest in chemical documentation, pharmacology, bio-informatics, pattern matching, and structural biology. Proteins can be represented by a primary structure, *i.e.* a sequence, a secondary structure with secondary structure elements (*SSE*s) made of polypeptide segments, *i.e.* helices, strands, loops etc., or a tertiary structure where the folding of the SSEs is specified. The tertiary structure (a 3D structure) of a protein can be simplified as a matrix consisting of SSEs ordered by their occurrence in the sequence, and interactions for SSE pairs with defined handednesses for SSE triples [16,23,24]. Likewise, a structural motif consists of ordered SSEs with defined interactions and handednesses for SSE pairs and triples respectively. The tertiary structure of a protein or simply a *protein* can be modeled as a graph in which nodes represent SSEs and edges between nodes model interactions where the handedness constitutes supplemental information. The degree of a protein is at most the maximum number of interactions an SSE can have, which is a constant. Similarly, a structural motif can be represented as a graph with additional information. Identifying a given structural motif in a given protein helps in discovering possible functional linkage between proteins. The *compact structural motifs* (see Section 2), *i.e. motifs*, are dictated by biology; they have a limited degree. The common motifs have a small number of SSEs; they are either trees or connected graphs whose number of edges exceeds the corresponding number of nodes by a constant. ProSMoS [23,24] finds all the copies of a motif in individual protein chains; the search process mimics the order of SSEs specified in the motif. For example, given a motif X with k SSEs $(1,2,\ldots,k)$ and a protein P, SSE indices $(i_1,i_2,\ldots,i_k)$ in P are possible candidates for X only if $i_j < i_{j+1}$ $(1 \leq j < k)$. Aung and Li [1] extract sequence independent common motifs (that are cliques) from an input that consists of a set of proteins. In contrast, our proposed system finds all copies of a given motif in P independent of the SSE sequence and chain(s). We note that it is more efficient to impose the desired criteria as constraints for searching instead of filtering the results. SSE types [23,24], order of SSE in the sequence [23,24], interaction type, and handedness are common constraints. SSE and interaction types are to be assigned to nodes and edges respectively, and handedness is to be pre-computed.

A *dominating set* for a graph $H=(V,E)$ [10] is a subset of nodes $V' \subseteq V$ such that any node in V is either a part of V' or adjacent to at least one member of V'. We say that a node in V' dominates its neighbors that are not in V'. We restrict that any node in V-V' is dominated by exactly one node belonging to V', breaking ties by node ID in case a node is dominated by two or more nodes from V'. Thus a dominating set captures the topology of a graph. Let S be a *minimum dominating set* of H, *i.e.* the dominating set of

ISB'10, February 15-17, 2010, Calicut, India.

minimum size. For any $v \in S$, the *dominion* rooted at v, *i.e.* $dom(v) = (\{v \cup \{u_i\}\}, \{(v,u_i)\})$ is the subgraph of H induced by v and u_i (the set of nodes v dominates). Thus, dominions partition H. The notion of dominion is inspired by Problem no. 591 in [21].

Our system searches for all copies of query graph Q in a given graph G. Trees are especially desirable query graphs because the minimum dominating set S of a tree can be computed in linear time and two adjacent dominions obtained from the tree share exactly one edge (otherwise, there is a cycle in Q). Given a query graph Q, we decompose Q into dominions, we search for these dominions in G and we merge the results of the search to obtain copies of Q. Decomposing a model graph is explored in [17,18,19]. Our method works only if we are able to merge the copies of Q's dominions in a time and space efficient manner.

We propose two algorithms that find all copies of Q in G (see Sections 3 and 4). Our algorithms are part of a system that finds all copies of a given motif (which has more edges than a tree that spans it) in proteins, i.e. *PDB* (protein database) (see Section 2). When finding approximate copies of Q in G suffices, we propose solutions for finding all approximate copies (see Section 4.1).

The paper has six sections. We present basic notations used throughout the paper and the system overview in Section 2. Section 3 details the search method using dominions. Section 4 presents a simplified method that uses a dominating set and gives details of a solution for searching approximate copies. In Section 5 we show how our methods can be extended to sparse query graphs. Section 6 gives conclusions and future research directions.

2. TERMINOLOGY AND OVERVIEW

A *motif* can be represented as an interaction matrix of the constituent SSEs and it can be searched in a PDB [16,23,24]. The proposed system consists of Tasks (i)-(iv) as shown in Fig. 1. Our main contribution is Task (iii) as described in Section 3. Task (i) constructs graphs $\{G_i\}$ for all proteins based on PALSSE [16]. This step creates a database offline. Edge and node types indicating the type of interaction {(hydrogen-bond, other-interaction) x (parallel, anti-parallel)} and the type of SSE {helix, strand} are assigned. In Task (ii), given a motif as a matrix, the corresponding graph Q^o is constructed. From Q^o a set of edges E^Q are removed to obtain a tree Q such that $|S|$ is minimum, where S is a minimum dominating set of Q. In Task (iii), $\{G_i\}$ is efficiently searched for the copies of Q. In Task (iv), the copies of Q thus found are processed (for additional edges, *i.e.* SSE interactions) to construct copies of Q^o. The information of the SSEs that constitute the final copies is the output.

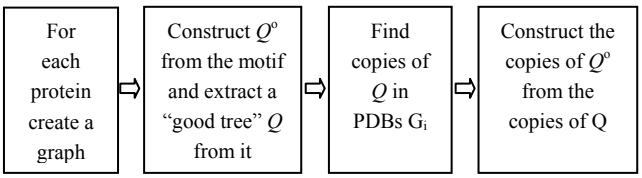

Figure 1. System Overview

We design an efficient algorithm for Task (iii) (critical step) in Section 3. Due to a small number of SSEs in a motif, the solution for Task (ii) described in Section 4 suffices (it does not increase the overall complexity). Task (iv), described in Section 5, due to the small number of deleted edges has a low time complexity.

The common types of *compact structural motifs*, *i.e. motifs*, are β-sheets, helical bundles, and β-sheet interacting with helices (bundles) or other β-sheets. In a β-sheet with interacting helices, each helix interacts with at least two strands of the sheet for the motif to be *compact*; it is modeled by a degenerate tree. If a structural motif is not connected or it has an SSE with only one interaction (non hydrogen bonded) then it is not compact. A manual survey of common motifs reveals that a motif with m SSEs has a spanning tree with at most $\lceil m/3 \rceil$ dominions (see Claim 1). Past results show that the number of copies of a given motif in a given protein with n SSEs is $O(n)$ (see Claim 2).

Claim 1. A motif with m SSEs has an underlying spanning tree with at most $\lceil m/3 \rceil$ dominions.

Claim 2. The number of copies of a motif in a given protein with n SSEs is $O(n)$.

Given two graphs G_1 and G_2, determining if G_1 contains a subgraph isomorphic to G_2, i.e. *subgraph isomorphism*, which we denote by $I_S(G_1,G_2)$ or I_S, is NP-complete [10]. It remains NP-complete even if G_2 is a tree [10]. Finding all copies of Q in G (a complete enumeration) which we denote as $E(G,Q)$ is clearly at least as hard as $I_S(G,Q)$ (a decision problem). The brute-force method for $E(G,Q)$ requires enumeration of all ${}^nC_m$ subgraphs $\{g_i\}$ induced by combinations of m nodes, mapping the nodes of Q on to the nodes of each $\{g_i\}$ in all possible (m!) ways, a total of (${}^nC_m$ m!) mappings, and for each mapping testing if an embedding of Q exists (in $O(m)$ time as Q is a tree with m-1 edges). Thus, the time complexity is $O(n^m\ m)$. In [7,20] I_S was reduced to clique detection by constructing an association graph. Messmer et al. [17] pre-process G (model graph) to obtain a possibly exponential decision tree, then in $O(m^2)$ time they decide whether an instance of query graph is present in a set of graphs. Messmer et al. propose a network-based approach to I_S [18]; they decompose the model graphs into a library of subgraphs in [19] with good performance in practice despite the exponential worst case time.

A related problem, graph isomorphism, determines if G_1 and G_2 are isomorphic; we denote it by I_G. Its complexity is unknown. I_G was mainly studied under group theoretic and practical approaches [17]. Practical approaches primarily use pruning and backtracking on the state-space search. Types of restrictions on the problem definition yielded polynomial algorithms in the group theoretic approach [2,3,9,11,12]. Luks [15] gives an $O(n^{O(n^{2/3})})$ algorithm for I_G based on [2,4,26]. As practical approaches [6,25] give the first results. A survey of earlier methods are in [8,22].

A *burst* B or $B(u)$ where u is the root of the burst is defined as follows. (i) An isolated node u is a burst with degree 0, *i.e.* $deg(B)=0$; u itself is the root, *i.e.* $r(B)=u$. (ii) Two nodes connected by an edge form a burst with degree one, *i.e.* $deg(B)=1$; either node can be the root (leaf). (iii) If some node u has j-1 neighbors (j>1) which are leaves with degree one then $deg(B)=j$-1 (see Fig. 2e). We denote a leaf and the i^{th} leaf of B respectively by $l(B)$ and $l_i(B)$. Leaves of Q and Q_s are denoted similarly. The cardinality of B, $|B|=deg(B)+1$. In rooted Q and Q_s, $parent(u)$ represents the parent of the node (supernode) u. A *dominion* and a *starburst* are bursts obtained in different ways. A supernode of Q_s is a dominion in Section 3 and a starburst in Section 4.

A tree of p nodes is called *degenerate* if p-2 nodes have a degree of two and two nodes have a degree of one (see Fig. 2a). A

degenerate tree with odd number of nodes is called an *odd degenerate tree*. An even degenerate tree is defined similarly. A tree with p nodes is called a *match stick tree* if it has $p/2$ internal nodes called *feet* and $p/2$ leaf nodes called *heads* where only two feet have a degree of two and are connected to one foot and one head, and the rest of the feet have degree three and are connected to exactly one head and two feet; the set of feet constitutes the minimum dominating set S (see Fig. 2b). We will show that an odd degenerate tree is a worst case for starbursts (see Section 3) and a match stick is the worst case for dominions and also a worst case for starbursts. For a cart-wheel tree (see Fig. 2c), the center node can be absorbed into a dominion of any of the white nodes. An hour-glass (see Fig. 2d) is formed by joining two centers c_1 and c_2 of two cart-wheels via a new dominion of degree zero, *i.e. r*; ε cart-wheels each with up to ε-1 spokes can be joined this way.

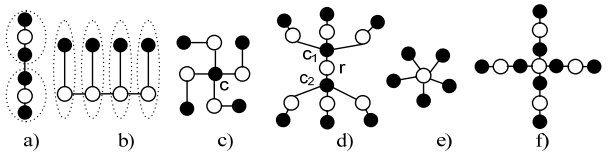

Figure 2. Types of trees. a) degenerate, b) match stick, c) cart wheel, d) hour-glass, e) burst, and f) 3-burst-4

A *component* is an improper subgraph of Q and can be recursively defined as either a burst (dominion in Section 2 or starburst in Section 3) or a merger of two components. Two components are merged into a larger component by an edge joining them.

Trees can be easily partitioned into dominions. If a node u has degree d then it has dominion(s) of degree at most d. For a query tree Q, if each dominion is collapsed into a *supernode*, then one still obtains a tree, Q_s. We call Q_s a *succinct tree*. We note that Q_s is critical in assembling copies of Q from the copies of dominions.

We find copies of dominions and merge them to obtain copies of Q. Only the copies whose edge connectivity mimics the corresponding components in Q_s are merged, forming larger copies (of a larger component). When G is not a clique, the number of copies of a component with k nodes that are formed will be less than the brute-force enumeration of k nodes. Also, each combination of nodes chosen by brute-force enumeration induces a graph with m nodes in which one has to find copies of Q. Hence, our method enumerates fewer trees with m nodes than a brute-force enumeration of all combinations of m nodes and at the end of the process, no test for subgraph isomorphism is necessary.

Two adjacent dominions D_1 and D_2 are connected by a unique edge $(u,v)\in Q$ where $u\in D_1$ and $v\in D_2$. We call (u,v) a *bridge*. Similarly, (u_c,v_c) is a *bridge* where u_c and v_c are copies in G of u and v respectively. Depending on whether u is a leaf or a root and whether v is a leaf or a root there are four types of bridges: a) leaf-root (*l-r*), b) root-root (*r-r*), c) leaf-leaf (*l-l*), and d) root-leaf (*r-l*). If $deg(D_1)=0$ then D_1 can have only *r-l or r-r* bridges.

Given two dominions D_1 and D_2, let x and y be some copies of D_1 and D_2 respectively in G. If (i) x and y are joined by a bridge (same type joining adoms D_1 and D_2) $(u,v)\in G$ s.t. $u\in x$ and $v\in y$, and (ii) they are node disjoint then x and y can be merged. The result is a larger copy. Let C_1 and C_2 be two components in Q_s and let x and y be some copies of C_1 and C_2 in G. If (i) x and y are joined by a bridge (same type as adoms) $(u,v)\in G$ s.t. $u\in adom$ of x and $v\in$ *adom* of y, and (ii) they are node disjoint then x and y can be merged. We call these as *basic criteria* in selecting eligible copies of components (hence dominions) in G. Criterion (i) can be verified in $O(1)$ time and Criterion (ii) can be checked in $O(|x|+|y|)$ akin to merging sorted lists by having a sorted list of indices within a copy. It suffices to check that x and y are node disjoint only once (for all feasible ways of merging x and y).

Let $D_1\in C_1$, $D_2\in C_2$, be two dominions such that D_1 and D_2 are joined by a bridge. We call D_1, D_2 as *active dominions or adoms* of components C_1 and C_2 respectively. Let c_{a1} be a copy of D_1 in c_1. Let c_{a2} be a copy of D_2 in c_2. When the bridge between C_1 and C_2 is of the type *l-l, or l-r* the following notation is applicable. Let $w\in c_{a1}$ be a leaf. We say that w is a *free leaf* of c_{a1} if merging c_1, c_2 using $(w,x(\in c_{a2}))$ as a bridge creates a of copy of a subgraph of Q. It must be noted that each *l-l* or *l*-r merge involving an adom of a copy affects the number of adom's free leaves. The number of free leaves of a copy of an adom is function of a) the past merges that the adom participated in, and b) the current merge.

In Q_s, a dominion D with $deg(D)>0$ can have *l-l* and *l*-r bridges. A leaf $u\in D$ can belong to: A) zero, B) one, or C) greater than one number of bridges. Say there are k_1, k_2, and k_3 leaves of types A), B) and C) respectively in D. Consider the scenario after the last merge with D as an adom. Let c be a copy thus formed. Let c_D be the copy of D in c. If c_D has k_1 leaves with no bridges then c meets the requirement for type A) bridges. If there are k_2 leaves of c_D that form exactly one bridge each with a specified dominion then c meets requirement for type B) bridges. If there are k_3 leaves of c_D that form specified number of bridges with specified dominions then c meets requirement for type C) bridges.

3. DOMINION METHOD

The following sequence of steps extract all copies of Q in G. 1) *Dom_Set* computes a minimum dominating set S of Q. 2) Algorithm *Dom_Ex* partitions Q into $|S|$ dominions $\{D_i\}$, i=1…$|S|$, and builds a *succinct tree* Q_s with these dominions as supernodes. We note that Q_s carries all the information of Q. 3) *Select_Root* picks a root for Q_s; we specify the *merging sequence* of $\{D_i\}$ to construct copies of Q from the copies of $\{D_i\}$. 4) We search G for copies of dominions $\{D_i\}$. (If $\deg(D_i) > \varepsilon$ for any D_i then we conclude that there are no copies of Q in G.) 5) We execute *Merge_Comp* on copies of $\{D_i\}$ as per merging sequence from Step 3. Thus, copies of Q are formed.

Step 1 (*Dom_Set* algorithm) is described next. S is initialized to empty set. The following steps are repeated until no node is left. In case a forest is obtained after any iteration, each tree in the forest is processed like the original tree.

1. Let V be the set of nodes with at least one adjacent leaf. For an isolated edge (u,v) either node can be designated as a leaf.
2. For all $v\in V$, delete v and all u_i adjacent to v unless $u_i\in V$.
3. Update $S = S\cup V$.

Step 2 (*Dom_Ex* algorithm) is as follows:

1. For each $u\in S$, include all the neighbors $\{v_i\}$ (that $\notin S$ and are not dominated by $w(\neq u)\in S$) of u and (u,v_i) in $dom(u)$.
2. If a node x can be included both in $dom(u)$ and $dom(v)$ then x (and (u,x)) is assigned to $dom(u)$ if $u < v$ and vice versa.

Thus, Q is partitioned into several dominions $\{D_i\}$ and the *succinct tree* Q_s is formed with $\{D_i\}$ as nodes. The edges of Q_s are defined as follows. For any two D_i, $D_j\in Q_s$, D_i and D_j are connected by an edge in Q_s (D_i and D_j are *adjacent*) if there exist two nodes u and v in Q such that $u\in D_i$ and $v\in D_j$ and $(u,v)\in Q$.

For Step 3, part 1, *Select_Root* algorithm assigns a root to Q_s:

1. The diameter d of the tree Q_s is computed.
2. We select u and v to be a pair of supernodes that are at distance d apart.
3. We choose the middle supernode on the shortest path between u and v as the root of Q_s. If d is even then there is a single choice. Otherwise, there are two choices r_1 and r_2, in this case, the two trees formed by deleting the edge (r_1, r_2) are examined. If the first tree containing r_1, has more nodes then r_1 is chosen; otherwise r_2 is chosen. In case of a tie, the node with a lesser index is chosen as the root.

For Step 3, part 2, we define the following inorder traversal of the rooted tree Q_s. Repeat until no child is left: a) traverse the leftmost unvisited child, and b) visit the root. A node will be visited after each child. The *merging sequence* is specified as follows: if two nodes are visited consecutively, the *components* and their copies (See Section 1) must be merged.

Step 4 searches for copies of all dominions in G; it is a straight forward enumeration. In order to find a dominion of degree s in a graph G, one finds all the nodes in G with the degree $d \geq s$. If the degree d of node x is s, then there is one copy of the dominion rooted at x. Otherwise, there are ${}^{d}C_s$ copies at x. Since d is $O(1)$, all copies of dominions rooted at a given node are found in $O(1)$.

A copy of a dominion is represented as a sequence where the leftmost entry is the root and the rest are leaves. Copy {3,4 5} with root 3 is different from {4,3 5} with root 4. However, {5,3 4} and {5,4 3} are the same dominions, with root 5. Likewise, if a copy of a tree is x={1,2 3 4,5 6,7, 8}, then the nodes 2, 3, 4 are children of node 1, the nodes 5, 6 are children of node 2, the node 7 is child of node 3, and the node 8 is a child of node 4. In x, 1 is root; 5, 6, 7, and 8 are leaves.

Let t_1={5,3 4} and t_2={5,4 3} be copies of a tree. By permuting the order of subtrees rooted at 5 *in* t_1, t_1 transforms into t_2. Thus t_1 and t_2 are *repeats* of the same tree. Our dominion method will find repeats if Q_s has symmetry. By Claim 2, the number of unique copies is $O(n)$. The number of repeats of a copy is a function of number of non-leaf nodes with more than one child in the rooted succinct tree Q_s. This number is relatively small because $m << n$. In linear time one can determine whether t_1 is a repeat of t_2. Thus, elimination of repeats takes limited time.

In Step 5, *Merge_Comp* algorithm merges the copies enumerated in Step 4 following the merging sequence of Step 3, part 2 to obtain copies of Q. Let C_1 and C_2 be two components of Q_s that are merged by *Merge_Comp* where C_1 is parent of C_2; *i.e.* if C_1 and C_2 are collapsed into supernodes then C_1 is the parent of C_2. Let A be the set of all copies of C_1 and B be the set of all copies of C_2 in G. *Merge_Comp* pairs copies from A and B which are connected by one of the bridge types a)-d) and then filters the feasible pairs by the *basic criteria* defined in Section 2. Below, we expand on the merging process for each type of bridge.

In Type a) bridge (l-r) each free leaf u of each copy $a \in A$ is paired with the root v of the adom of each copy $b \in B$. This is a many-to-many mapping. The root u of an adom of a copy in B can be paired to any free leaf (of any copy) of A. In Type b) bridge for each copy $a \in A$, the root of its adom u is paired to the root v of the adom of each copy $b \in B$. In Type c) bridge for each copy $a \in A$, every free leaf of its adom u is paired to every free leaf of the adom v of each copy $b \in B$. In Type d) bridge for each copy $a \in A$, the root of its adom u is paired to every free leaf v of the adom of each copy $b \in B$. Type a) bridge (l-r) and Type d) bridge (r-l) are symmetric. More details are given below.

Let C_1 and C_2 be the components being merged as per merging sequence. Let D_1 and D_2 be the adoms of C_1 and C_2 respectively. Let (w,x) be the bridge with $w \in D_1$ and $x \in D_2$. Let c_1 and c_2 be some copies of C_1 and C_2 respectively in G and $a_1 \in c_1$ and $a_2 \in c_2$ be the copies of D_1 and D_2 respectively joined by a bridge (y,z) where $y \in a_1$ *and* $z \in a_2$. Basic criteria are applied to c_1, c_2 and (y,z).

For Type a) bridge, $r(a_2)$ is the only candidate for z. We have two sub-cases: Subcase a1) (y,z) is the only bridge incident on y; any $l(a_1)$ that has no bridges qualifies. Subcase a2) other bridges are incident on y. Sub-case a2) can be further divided into: Sub-case a2-i) the current merge is the first merge via a bridge incident on y. Any $l(a_1)$ without an incident bridge qualifies. Sub-case a2-ii) there was a previous merge with a bridge incident on y. Let the prior merges of a_1 with a bridge (y, z_i) be with $\{a_j,..,a_k\}$ where $z_i \in a_i$ and a_i is a copy in G of D_i. Select $l_j(a_1)$ that has specified l-l or l-r bridges to a_i ($i=j \ldots k$). Note that such $l_j(a_1)$ is unique.

For Type b) bridge, $(r(a_1),r(a_2))$ is the only option for (y,z).

For Type c) bridge we have four sub-cases. Subcase c1) (y,z) is the only bridge incident on y or z; any $l(a_1)$ and any $l(a_2)$ with no incident bridges qualify. Subcase c2) (y,z) is the only bridge incident on y and z has other incident bridges. Subcase c3) (y,z) is the only bridge incident on z and y has other incident bridges. Subcase c4) both z and y have other incident bridges. Subcases c2) and c3) are equivalent. Subcase c2) can be classified into: Subcase c2-i) this is first merge involving a bridge incident on z. Here any $l(a_1)$ and any $l(a_2)$ with no incident bridges qualify. Subcase c2-ii) There was a previous merge involving a bridge incident on z. Let $(a_j,..,a_k)$ be the copies of dominions in G with which a_2 was merged by bridges (p, q_i) where $p \in a_2$, $q_i \in a_i$ and a_i is a copy in G of D_i, $i=j \ldots k$. Select any $l(a_1)$ that has no incident bridges and $l_j(a_2)$ that has l-l or l-r bridges as per Q_s to a_i, $i=j \ldots k$. Note that there can be only one such $l_j(a_2)$. Subcase c4) can be divided into Subcase c4-i) This is first merge involving a bridge incident on y or z; any $l(a_1)$ and $l(a_2)$ with no incident bridges qualify. Subcase c4-ii) This is first merge involving a bridge incident on y and z has a prior merges via bridges incident on it, Subcase c4-iii) This is first merge involving a bridge incident on z and y had a prior merges with bridges incident on it, and Subcase c4-iv) Both z and y had a prior merges with bridges incident on them. Subcase c4-ii) is similar to Subcase c2-ii) and Subcase c4-iii) is similar to Subcase c4-ii). Subcase c4-iv): Let $(a_f,..,a_g)$ be the copies of dominions in G with which a_1 was merged by bridges (y, z_i) where $y \in a_1$, $z_i \in a_i$ and a_i is a copy of D_i in G, $i=f \ldots g$. Select $l_j(a_1)$ that is connected to a_i for $i=f \ldots g$; such $l_j(a_1)$ is unique. Likewise, let $(a_j,..,a_k)$ be the copies of dominions in G with which a_2 was merged by bridges (p, q_i) where $p \in a_2$, $q_i \in a_i$, and a_i is a copy of D_i in G, $i=j \ldots k$. Select $l_j(a_2)$ that has l-l or l-r bridges as per Q_s to a_i, $i=j \ldots k$. Note that such $l_j(a_2)$ is unique.

Lemma 1. *Dom_Ex* algorithm, applied to Q, $|Q|=m(>2)$, produces a tree Q_s with $|Q_s| \leq m/2$.

Proof: If all supernodes of Q_s have cardinality more than one then $|Q_s| \leq m/2$. Let $u \in Q_s$ and $|u|=1$. Thus, $r(u)$ is the only member of u. Also, if $r(u)= l(Q)$, it will be found as a leaf in the first iteration of *Dom_Set* and $|dom(r(u))|>1$ (contradiction). (Note that Q is not rooted, the following argument holds for any u). So, $r(u) \neq l(Q)$

and u has bridges to at least two dominions. In any iteration of *Dom_Set* $r(u)$ cannot be exposed as a leaf (connected to other node). If $|u|=1$ then *Dom_Set* finds $r(u)$ as an isolated node. So, at least two dominions v and w adjacent to u are formed simultaneously before $r(u)$ is found as an isolated node. If $r(v)$ is adjacent to $r(u)$ then v absorbs $r(u)$. So $l_x(v)$ is adjacent to $r(u)$. Moreover, $l_y(v)$ ($x \neq y$) must have been exposed as a leaf in order to form v because v forms before u. Thus $|v| \geq 3$ (similarly, $|w| \geq 3$). On average, u,v, and w have at least (1+3+3)/3=7/3 (>2) nodes. For example, a degenerate tree with seven nodes has three dominions u,v, and w where $|u|=|v|=3$ and $|w|=1$. Thus, the number of dominions is at most $m/2$.•

Corollary 1. Among all trees of m nodes, the largest number of dominions occurs in a match stick tree.

Proof: In a match stick tree, each node in S dominates exactly one other node. Therefore, the number of nodes is $m/2$ which is the maximum.•

Lemma 2. The number of copies in G of a dominion of any degree d is $O(n)$.

Proof: The maximum degree of any node in G, ε, is a constant ($O(1)$). Let D be a dominion of some degree d ($\leq \varepsilon$). The number of copies of D rooted at any node in G is bounded above by ${}^{\varepsilon}C_d$ which is also a constant. Thus the total number of dominions of degree d in G is bounded above by $n\,{}^{\varepsilon}C_d$ which is $O(n)$.•

If two adoms D_1 and D_2 are connected by an *r-r* bridge then their corresponding copies, c_1 and c_2, can be connected only by $(r(c_1),r(c_2))$. Thus, if $|Q_s|=k$ and Q_s has only *r-r* bridges then our method is $O(mn^k)$. However, an *l-l* or *l-r*/*r-l* (which we call *lr-rl*) bridge implies that the copies connected by them can be merged in several ways and hence introduce a multiplicative factor to the complexity which we call a *factor*. Lemma 3 shows that *l-l bridges* do so by reducing $|Q_s|$. For an *l-r* bridge from D_1 to D_2 to have $q>1$ feasible merges, D_1 must have q leaves which also reduces $|Q_s|$. Thus, non *r-r* bridges that contribute a factor (>1) and make several merges feasible, do so by reducing $|Q_s|$. Because $|Q_s|$ determines the exponent of n, overall, they reduce the complexity. This relationship is further analyzed below.

We define a *threshold* of two nodes per dominion. For a dominion D, if $|D|=1$ then D has no leaves. If $|D|=k$ (≥ 2) then we say that D is (k-2)*-over* the threshold (it has k-2 nodes over *threshold*). Similarly, if $|Q| = k$ and $|Q_s| = p$ then Q is (k-2p)*-over.*

In case of *lr-rl* bridges we establish the relation between the *factor* and $|Q_s|$. For trees with less than four nodes, $|Q_s|=1$. Only a degenerate tree with four nodes has $|Q_s|=2$ with one *r-r* bridge. For five nodes, there are four possible trees and only the burst with degree 4 has $|Q_s|=1$. The rest are 1-over with $|Q_s|=2$. Out of these, only degenerate tree has an *lr-rl* bridge with factor of two. The following trees are 1-over: a cart-wheel (see Fig. 2c) has one *lr-rl* bridge per spoke except the spoke that contains the center, cart-wheel-(ε-1) has ε-1 *lr-rl* bridges and an hour-glass-ε-1 (see Fig. 2d) has 2(ε-1) *lr-rl* bridges, the maximum number we could find. However, their respective factors are only two and four. If several roots have *r-l* bridges to a particular leaf l^*, only the first root has a factor; the rest are forced to choose l^*. For example, the cart-wheel has a dominion that is 1-over and a factor of 2 for the first dominion that is merged with it; the subsequent merges have a factor of one. An hour-glass has two dominions that are 1-over, each with a factor of 2 (total 4). Similarly, in 3-burst-k (see Fig. 2f) every spoke is 1-over each with a factor of 2. Thus, it is more relevant to find the maximum factor per 1-over.

Let D be a dominion with k leaves. The maximum factor for D with *l-r* bridges to other dominions is $k!$. Thus, per a node over threshold of two the maximum factor is $f = (k!)^{1/(k-1)}$ which is bounded by $(\varepsilon/e)^{\varepsilon/(\varepsilon-1)}$ for large ε (Sterling's approximation) where e is the base of natural log. Since ε is limited, for ε=5, f=3.31 and for ε=10, f=5.36. A similar analysis holds for *l-l* bridges where the extra leaves are present in both dominions to be merged. So, the maximum factor per 1-over is $O(1)$. However, the complexity decreases by $O(n^{1/2})$ for every 1-over. Thus, *lr-rl* and *l-l* bridges reduce the overall complexity. Two *l-l bridge*s reduce the maximum value of $|Q_s|$ by one and the complexity by $O(n)$ which is asymptotically larger than the maximum contribution of factor, *i.e.* f^2. Let Q_s, where $|Q_s|=k$, have only *l-l bridge*s (k-1 of them). If $(m-(k-1)/2=k$ then k is maximized to $(m+1)/3$, which yields $O(n^{(m+1)/3}\, f^{(m+1)/3-1})$. The maximum complexity with only *r-r* bridges is $O(n^{m/2})$ which is greater by $O(n/f^2)^{m/6}$.

Let $\{D_1, D_2,\ldots,D_k\}$ be the set of all dominions with at least two leaves. Let l_i be the number of leaves of D_i and b_i ($\leq l_i$) be the number of unique leaves of D_i with incident bridges. The factor f_i of a dominion D_i is given by $\Pi(l_i - j)$ (j=0 …b_i-1) and the total factor for all dominions, $F = \Pi(f_i)$. If l_x=6, D_x has 4 bridges incident on its leaves, and b_x=3 then f_x= (6.5.4) because b_x=3 even though the total number of incident bridges on the leaves are four.

Lemma 3. If the number of *l-l* bridges in Q_s is p then $|Q_s| \leq (m-p)/2$.

Proof: We show that a) an *l-l* bridge adds an extra node to some dominion (forces 1-over), and b) each additional *l-l* bridge, even if it is incident to a dominion that is j-over (j>0), forces 1-over.

We examine the process of forming dominions. Step 1 of dominion method finds all nodes in $S = \{v_i\}$. A node v_i is discovered because $u_j \in dom(v_i)$ is found as a leaf of a tree (Q or any tree in the forest formed by *Dom_Set* algorithm) in an iteration of *Dom_Set*. Thus, there is a temporal relation between the discoveries of dominions. Let $i(v_i)$ be the iteration in which $r(v_i)$ is identified (v_i is formed). Step 2 simply breaks the ties if they exist. We use this causal relationship in our proof.

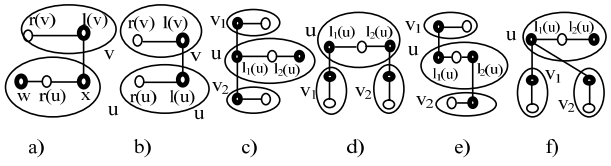

Figure 3. *l-l* bridges; roots are white. a) $u=l(Q_s)$, b) $u \neq l(Q_s)$, c) Case A), d) Case B, e) same as d), and f) same as c).

First we prove a). Consider a pair of dominions u, $v \in Q_s$ joined by *l-l* bridge. So, $(l_i(u),l_j(v)) \in Q$. This gives rise to the cases shown in Fig. 3a and Fig. 3b respectively. First, consider the case where u is connected only to v (Fig. 3a). This implies that a $l(u)$ say w, is $l(Q)$. Clearly, if $(l_j(v), w) \in Q$ a cycle is formed. So, $|u|>2$. Thus, u is at least 1-over. In the remaining case u and v are 0-over and $(l(u), l(v)) \in Q$. Neither $r(u)$ nor $r(v)$ can have additional leaves. The following argument holds for v also. For $r(u)$ to be the root of u there are two possibilities: (i) $l(u)$ is found as a leaf in one of the iterations of *Dom_Set* (before $r(u)$); (ii) The isolated edge ($l(u)$,

$r(u)$) is found in one of the iterations of *Dom_Set*, thereby the assignment of leaf and root are arbitrary. Case (i) is not possible because of the edge ($l(u)$,$l(v)$). Case (ii) implies that isolated ($l(v)$, $r(v)$) should be found before isolated ($l(u)$, $r(u)$) can be found (otherwise ($l(u)$, $l(v)$) prevents ($l(u)$, $r(u)$) from being isolated) and isolated ($l(u)$, $r(u)$)should be found before isolated ($l(v)$, $r(v)$) can be found (contradiction).

Here we prove b). If dominions v_1, v_2 where $|v_1|=|v_2|=2$ have *l-l bridge*s with a dominion u ($|u|=3$) without forcing additional 1-over then on average an *l-l bridge* does not force 1-over. We show that this is not possible by considering rooted succinct tree Q_s. The scenario considered above is critical. The proof for this scenario holds even if more than two dominions with cardinality two have an *l-l bridge* with u and/or $|u|>3$. This yields Case A): Fig. 3c) and Case B): Fig. 3d). The cases shown in Fig. 3f) and Fig. 3e) are equivalent to Case A) and Case B) respectively.

∗ Case A): $\exists x_2$ s.t. $(r(v_2),x_2)\in Q$ and $x_2\neq l(v_2)$; otherwise, $r(v_2)=l(Q)$ which implies that $r(v_2)$ would have been $l(v_2)$ in the first iteration (contradiction). Likewise, $\exists x_1$ s.t. $(r(v_1),x_1)\in Q$ and $x_1\neq l(v_1)$.

The following argument holds for the pair v_2 and x_2 also. If v_1 was formed before or in the same iteration as u then $r(v_1)$ would have been found as $l(v_1)$ (contradiction). Thus, u was formed before v_1. Let x_1 belong to dominion w_1 and $i(w_1) \leq i(v_1)$ where $i(w_1)$ is the maximum value for all such dominions. If such w_1 does not exist then $x_1\in v_1$ and $|v_1|=3$ (contradiction). Also, if $i(w_1)<i(v_1)$ then $r(v_1)$ would have been found as $l(v_1)$ unless v_1 is found as an isolated dominion-(i) (the root and leaf assignment in such a case is arbitrary). If w_1 forms before u then $r(v_1)$ would have been found as $l(v_1)$. Thus, w_1 forms along with u or after it -(ii). Conditions (i) and (ii) above yield two scenarios: (I) $i(w_1) = i(v_1)$ and $i(u) < i(w_1)$ or (II) $i(w_1) = i(u)$ and $i(w_1)<i(v_1)$.

In scenario (I), if $|w_1|>2$ then we are done. Assume that $|w_1|=2$. In iteration $i(w_1)$-1, $l(w_1)$ has a bridge to at least one other dominion say w'. So w' forms before w_1 to expose $l(w_1)$ as leaf and w_1 does not absorb $l(w_1)$. Thus, $r(w')$ is not adjacent to $l(w_1)$ and the leaf of w' that was exposed is not adjacent to x_1. So, w' has leaf adjacent to x_1 and another leaf. So $|w'|>2$; *i.e.* w' is 1-over. In scenario (II), w_1 forms before v_1 and does not absorb $r(v_1)$. The same argument used above yields $|w_1|>2$. Thus w_1 is 1-over.

∗ Case B): if v_1 or v_2 are formed before u then $r(v_1)$ and $r(v_2)$ would have been found as leaves of v_1 and v_2 (contradiction). Because both leaves of u are connected to $l(v_1)$ and $l(v_2)$; $l_1(u)$ and $l_2(u)$ do not get exposed as leaves before v_1 and v_2 are formed. So, other cases are not possible. Thus, in all cases an *l-l bridge*, ($l(v_1)$,$l(u)$) or ($l(v_2)$,$l(u)$), forces 1-over.•

Corollary 2. If the number of *l-l bridge*s in Q_s is p then our method is $O(n^{(m-p)/2}mF)$ where F is the total factor.

Theorem 1. Our method finds all copies of Q in G.

Proof: Dominions are the simplest components. For each dominion, all its copies are enumerated. So, before the first merge we have all copies of all dominions. After the first merge all the copies of component that is formed by combining two dominions with a bridge are enumerated. Subsequently, the larger components are similarly formed by emulating the subtrees of Q and their copies are also enumerated. So, all possible combinations of copies are considered and all the feasible ones are retained. Thus, all copies of Q are found.•

Theorem 2. Our method is $O(mn^{m/2})$.

Proof: Let A and B be two sets of copies of components that are to be merged to form components with k nodes. Let $a\in A$ and $b\in B$. The basic criteria for merging a and b by an edge (u,v) where $u\in a$, $v\in b$ are: (i) $(u,v)\in G$ and (ii) a and b are node disjoint. Criterion (i) can be verified in $O(1)$ and Criterion (ii) can be checked in $O(k)$, where $k= |a|+|b|$ akin to merging sorted lists by maintaining a sorted list of node indices within a copy. For any type of bridge joining *adoms* of a and b, they need to be deemed as node disjoint only once (even *l-l* and *lr-rl*).

Consider the worst case where $|Q_s|=m/2$. In the merging sequence, several pairs of sets (A and B) are merged to form sets of larger copies. A copy $a\in A$ must be paired with every copy of B to determine feasible merges. Thus, there are $|A|\cdot|B|$ such pairings. For each pairing basic criteria takes $O(1)+O(k) = O(k)$. Thus the total complexity is $O(|A|\ |B|\ k)$. In the final merge $k= |a|+|b|= m$ and $|A||B|= O(n^{m/2})$ yielding $O(mn^{m/2})$. The sum of complexities of all prior merges is asymptotically less than the final merge.•

$O(mn^{m/2})$ is comparable to the square root of the complexity of brute-force search. By Claim 1 we have that $|Q_s|\leq\lceil m/3\rceil$ which yields Corollary 3.

Corollary 3. For motif search our method is $O(mn^{\lceil m/3\rceil})$.

4. SIMPLIFIED IMPLEMENTATION WITH STARBURSTS

Certain modifications are done to the method presented in Section 3 for the ease of implementation. The major difference is that we find *starbursts* (see next paragraph) instead of dominions. The term starburst was used in [5]. We assign a root to Q instead of Q_s by the *Select_Root* and then partition Q into starbursts B_i as per *Burst_Ex*. We build *succinct tree* Q_s whose nodes are B_i. Q_s specifies the *merging sequence* of B_i to reconstruct Q. We then search G for copies of B_i and form copies of Q from copies of B_i.

Select_Root assigns a root to Q. As a consequence the starburst $r(Q_s)$ contains $r(Q)$. *Burst_Ex* is as follows. Repeat the process until only the starburst containing $r(Q)$ is left: (i) among the nodes still connected to Q choose a node u that is farthest from $r(Q)$; (ii) pick v which is parent(u) and all children of v (including u) with incident edges as $B(v)$ and delete $B(v)$ from Q by deleting (v, *parent*(v)). The algorithm terminates when the final starburst $B(r(Q))$ is found (which can have a degree of 0).

Lemma 4. *Burst_Ex* produces no more than $\lceil m/2\rceil$ starbursts.

Proof: *Burst_Ex* produces Q_s (with starbursts, B_i as nodes) from Q where $|Q| = m$. In a degenerate tree, every starburst except $B(r(Q))$ has degree one. $B(r(Q))$ has degree of: zero if $m=4k+1$, one if m is even, and two if $m=4k+3$ ($k\in 1^+$). (If $deg(B_i)=x$ then $|B_i|=x+1$). Thus, if $m=4k+1$ then $|Q_s| = 2k+1= (m+1)/2$. If $m=4k+3$ then $|Q_s| = (4k+3-3)/2+1 = 2k+1= (m-1)/2$. If m is even then $|Q_s|= (m-2)/2+1= m/2$. For match stick tree (and degenerate tree with even number of nodes) $|Q_s| = m/2$. If the tree is not degenerate (or match stick) then the degree of one or more of the corresponding B_i will increase, reducing $|Q_s|$. Thus, in a degenerate tree where $m = 4k+1$, $|Q_s|$ maximizes to $\lceil m/2\rceil$.•

The modification to the dominion method has several effects. *Burst_Ex* limits the bridge types between the nodes of Q_s to *r-r*

and *l-r*. This simplifies the implementation. For a given Q, $s_Q \geq d_Q$ where s_Q, d_Q are the number of starbursts and dominions respectively. For example, in Fig. 4a)-4b), Q yields 3 dominions {4, 1 7 8 9 10}, {3, 6} and {5, 2 11 12 13} or 4 starbursts {4, 7 8 9 10}, {5, 11 12 13}, {3, 6} and {1, 2}. The degenerate tree is the worst case, where $s_Q=(3/2)d_Q$. Thus for some types of trees, starburst method has higher complexity. Similar to active dominions *active starbursts* are defined. The concept of *free leaves* holds here also. Lemma 5 directly follows from Lemma 2.

Lemma 5. The number of copies of a starburst of a given degree d in G is $O(n)$.

Lemma 6. For any tree, the ratio of the number of starbursts over the number of dominions is at most 3/2.

Proof: Let s_T be the number of starbursts in a given tree T and d_T be the number of dominions. A match stick tree has the largest number of dominions. However, for a match stick tree, $d_T=s_T$. A degenerate tree with m nodes has $s_T = m/2$ and $d_T = m/3$, thus $s_T/d_T=3/2$. We show that this ratio is the maximum possible value.

First we examine the *Burst_Ex* algorithm. Let L_i be the set of leaves at the maximum distance from the root in the i^{th} iteration. Let P_i be the set containing unique parents of L_i and $|P_i|=k_i$. After the first iteration, the algorithm chooses the j^{th} parent u_j as the root and all its children as leaves of the j^{th} starburst; a total of k_1 starbursts. However, all p_j $(\in P_1) \in S$, the minimum dominating set. In case a dominion, $dom(u)$, has maximum cardinality then it covers all children of u (leaves adjacent to u), and parent(u). The least a starburst can cover is u and all its children. This argument holds for all iterations. The ratio is maximized when the number of leaves of u is minimum, *i.e.* one. Thus, the upper bound is 3/2.•

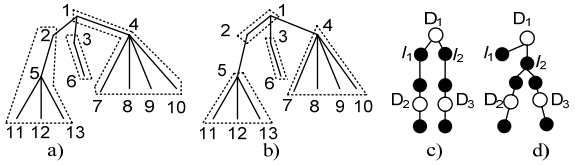

Figure 4. a) dominions, b) starbursts, c) *l-l* Case 1, and d) *l-l* Case 2.

Fig. 4c) and 4d) depict types of *l-l* bridges; roots are white. Let the merge sequence be *merge*1 (D_1, D_2) followed by *merge*2 ($C_{1\text{-}2}$, D_3) ($C_{1\text{-}2}$ is a component composed of D_1, D_2, and the bridge joining them). In Fig. 4c), if *merge*1 uses l_1 then *merge*2 uses l_2 and vice versa. Thus, for a copy of D_1, for *merge*1, both leaves are free leaves. However, for *merge*2 only the leaf that did not form a bridge in *merge*1 is a free leaf. In Fig. 4d) one leaf of D_1 has *l-l* bridges to D_2 and D_3; *merge*1 can use any leaf of D_1 but *merge*2 has to use the same leaf. Thus, for a copy c of D_1, *merge*1 has two free leaves; however, *merge*2 has only one. This is derived from the following principle: if $\{r_1, r_2, \ldots, r_k\}$ are children of r then the subtrees rooted at $\{r_1, r_2, \ldots, r_k\}$ can be rotated around r but cannot be split or merged. This applies to the merging of copies of any type of components (composed of either dominions or starbursts).

Lemma 7. The starburst implementation consumes $O(mn^{m/2})$ time.

Proof: The complexity is given by $O(mn^{m/2-k}\ F)$; where F is the total factor. The worst case occurs when there are $m/2$ starbursts, *i.e.* k=0 and F=1, yielding a complexity of $O(mn^{m/2})$.•

Clearly, the complexity is better than the brute-force search and is comparable to the dominion method. The copies specified by merging sequence are merged with *merging algorithm* applied to starbursts (Section 3). Each merge creates larger copies of a larger component, eventually forming the copies of Q in G.

4.1 Solutions for Approximate Copies

Given two components to be merged subject to having an edge in G, we present two simple approximation types for their copies. The first type, say a_1, allows the merge of copies even if one of the copies is an approximation of its corresponding component by having exactly one less edge (thus, node) than required. However, the corresponding node of the missing node in the component should not have an incident bridge. The second type, say a_2, allows the copies to be merged even if they are connected by an additional intermediate edge instead of a direct bridge.

The approximation criteria are applied to a copy because different copies require approximations (if at all) in different merges. One or more criteria can be chosen. Each edge that is in Q but absent in a given copy contributes one a_1 approximation. Each edge that is present in a given copy but is absent in Q contributes one a_2 approximation. Each copy will accrue its corresponding approximations. Let $\#a_x$ denote the number of a_x approximations. For three given constants k, k_1, and k_2, the possible schemes are: (i) $\#a_1 < k$, (ii) $\#a_2 < k$, (iii) $\#a_1 + \#a_2 < k$, (iv) $\#a_1 < k_1$ and $\#a_2 < k_2$. Methods given by Kao et al. [13] can be used to estimate the quality of the enumerated approximate copies.

5. SPARSE GRAPHS

A tree with m nodes has only m-1 edges. Let us consider a sparse connected graph Q^o with m nodes and m-1+p edges where p is bounded by a constant. (Otherwise, the number of unique spanning trees of Q^o is exponential.) In order to find all copies of Q^o in G, *i.e.* $E(G,Q^o)$, the following tasks are executed: (a) we pick a tree $Q \subset Q^o$ which has minimum value for |S| where S is the minimum dominating set of Q, (b) we find copies of Q in G, and (c) from copies of Q we find copies of Q^o in G. Task (a) and Task (c) which are absent in $E(G,Q)$ constitute *sparse graph overhead*.

A direct method for Task (a) is to delete each of ${}^{m-1+p}C_p$ combinations of p edges from Q^o. Each such deletion from Q^o can result either in a tree, or a set of trees and graphs with cycles. We pick the deletions that yield a single tree: *i.e. DFS/BFS* from any node reaches all the other nodes. Among all such trees, a tree Q with minimum value for $|S|$ is chosen (S can be computed in linear time). Thus, for a fixed p, Q can be obtained in time polynomial in m. Liu and Wang [14] give an efficient way of enumerating simple cycles. Their method might yield a better strategy to enumerate all trees, thus reducing the complexity.

Task (c), which we call *garnering,* determines whether a copy of Q can have additional edges in G to be a copy of Q^o. The best case for garnering is $O(p)$ time when all the deleted edges of Q^o are incident on the nodes of the copies that can be uniquely identified. There are two factors that make identification of a node difficult, hence increasing the complexity of garnering: 1) a leaf and its siblings in a copy are equivalent, and 2) there is a rotational symmetry in Q_s. As a consequence, given a copy c of Q in G, the number of copies of Q^o in G constructed from c (with c as a subcopy) can be greater than one, and given copies c_1 and c_2 of Q in G, the copies of Q^o in G constructed from c_1 and c_2 as

subcopies can be repeats of one another. Thus, in certain cases the complexity is greater than $O(p)$.

In motif search, the node and the edge types, and the handedness constraints reduce the ambiguity in identifying a node thus decreasing the complexity of Task (c). Finally, due to a small value of p, sparse graph overhead does not increase the complexity of $E(G,Q^o)$ which remains bounded by the complexity of $E(G,Q)$. (Section 3 provides an efficient algorithm).

6. CONCLUSIONS

Sequence independent search of a specified motif in proteins, a complex 3D problem, is modeled as a subgraph (tree) enumeration and an efficient algorithm is designed for it. In prior research, the search was either sequence dependent [23,24] or focused on finding common motifs that are cliques among a given set of proteins [1]. In our algorithm, copies of trees are constructed, bypassing the need for subgraph isomorphism test. It can be adapted to any application. The time complexity is $O(mn^{\lceil m/2 \rceil})$ for general trees and $O(mn^{\lceil m/3 \rceil})$ for motif trees.

The case where the maximum degree of any node in G is not bound by a constant is open. In the proposed system, we assume that a given sparse connected graph Q^o with m nodes has at most m-1+p edges where p is a constant. Thus, Task (a) and Task (c) of Section 5 do not increase the overall complexity and efficient solutions for them remain open. It is of future interest to find the threshold value of p where the complexity of these tasks starts to dominate the overall complexity, thus showing the limit of extending our basic method (for query graphs that are trees) towards sparse query graphs.

7. ACKNOWLEDGEMENTS

This research has been funded in part by the Welch Foundation Grant I1505 to Nick V. Grishin.

8. REFERENCES

[1] Aung, Z., and Li, J. Mining super-secondary structure motifs from 3D protein structures: a sequence order independent approach. *Genome Informatics*, 19 (2007) 15-26, PMID: 18546501.

[2] Babai, L. Moderately exponential bound for graph isomorphism. In *Proceedings of FCT*, Springer Verlag, 1981, LNCS 117, 34-50.

[3] Babai, L., Grigoryev, D.Y., and Mount, D.M. Isomorphism of graphs with bounded eigen value multiplicity. In *Proceedings of STOC*, 1982, 310-324.

[4] Babai, L., and Luks, E.M. Canonical labeling of graphs. In *Proceedings of STOC*, 1983, 171–183.

[5] Bein, D., Morales, L., Bein, W., Shields Jr., C.O., Meng, Z., and Sudborough, I.H. Clustering and the biclique partition problem. In *Proceedings of HICSS*, 2008, 235-242.

[6] Corneil, D.G., and Gotlieb, C.C. An efficient algorithm for graph isomorphism. *J. ACM* 17, 1 (1970) 51-64.s

[7] Falkenhainer, B., Forbus, K.D., and Gentner, D. The structure mapping engine: algorithm and examples. *Artificial Intelligence* 41, 1 (1989) 1-63.

[8] Gati, G. Further annotated bibliography on the isomorphism disease. *J. Graph Theory* 3, 2 (1979) 95-109.

[9] Filotti, I.S., and Mayer, J.N. A polynomial-time algorithm for determining the isomorphism of graphs with fixed genus. In *Proceedings of STOC*, 1980, 236-243.

[10] Garey, M.R., and Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[11] Hoffman, C.M. *Group Theoretic Algorithms and Graph Isomorphism*. Springer Verlag, LNCS 136, 1982.

[12] Hopcroft, J.E., and Wong, J.K. Linear time algorithm for isomorphism of planar graphs. In *Proceedings of STOC*, 1974, 172-181.

[13] Kao, M.Y., Lamb, T.W., Sung, W.K., and Ting, H.F. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *J. Algorithms* 40, 2 (2001) 212-233.

[14] Liu, H., and Wang, J. A new way to enumerate cycles in graph. In *Proceedings of AICT-ICIW*, 2006, 57.

[15] Luks, E.M. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *Proceedings of FOCS*, 1980, 42-49.

[16] Majumdar, I., Krishna, S.S., and Grishin, N.V. PALSSE: a program to delineate linear secondary structural elements from protein structures, *BMC Bioinformatics* 6 (2005) 202.

[17] Messmer, B.T., and Bunke, H. Subgraph isomorphism detection in polynomial time on preprocessed model graphs, chapter in *Recent Developments in Computer Vision* (1996) Springer Verlag, LNCS 1035, 373-382.

[18] Messmer, B.T., and Bunke, H. Error-correcting subgraph isomorphism using decision trees. *Intl. J. Pattern Recognition and AI* 12, 6 (1998) 721-742.

[19] Messmer, B.T., and Bunke, H. Efficient subgraph isomorphism detection: a decomposition approach. *IEEE Trans. on Knowledge and Data Engineering* 12, 2 (2002) 307-323.

[20] Myaeng, S. H., and Lopez-Lopez, A. Conceptual graph matching: a flexible algorithm and experiments. *J. of Experimental and Theoretical AI* 4, 2 (1992) 107-126.

[21] Parberry, I. *Problems on Algorithms*. Prentice Hall, 1995.

[22] Read, R.C. and Corneil, D.G. The graph isomorphism disease. *J. Graph Theory* 1, 4 (1977) 339-363.

[23] Shi, S., Chitturi, B., and Grishin, N.V. ProSMoS server: a pattern-based search using interaction matrix representation of protein structures. *Nucleic Acids Research* 37 (2009) (Web Server issue):W526-31, PMID: 19420061.

[24] Shi, S., Zhong, Y., Majumdar, I., Krishna, S.S., and Grishin, N.V. Searching for three-dimensional secondary structural patterns in proteins with ProSMoS. *Bioinformatics* 23, 11 (2007) 1331-1338.

[25] Ullman, J.R. An algorithm for subgraph isomorphism. *J. ACM* 23, 1 (1976) 31-42.

[26] Zemlyachenko, V.N., Korneenko, N.M., and Tyshkevich, R.I. Graph isomorphism problem. *J. Mathematical Sciences* 29, 4 (1985) 1426-1481.