Tel-Aviv University
Raymond and Beverly Sackler
Faculty of Exact Sciences

Topology-Free Querying of Protein Interaction Networks

Thesis submitted in partial fulfillment of the requirements for
the M. Sc. degree in the Blavatnik School of Computer Science,
Tel-Aviv University

by

Sharon Bruckner

The research work for this thesis has been carried out at Tel-Aviv University
under the supervision of
Prof. Ron Shamir and **Prof. Roded Sharan**.

August 2009

Abstract

In the network querying problem, one is given a protein complex or pathway of species A and a protein–protein interaction network of species B; the goal is to identify subnetworks of B that are similar to the query in terms of sequence, topology, or both. Existing approaches mostly depend on knowledge of the interaction topology of the query in the network of species A; however, in practice, this topology is often not known. To address this problem, we developed a topology-free querying algorithm, which we call Torque. Given a query, represented as a set of proteins, Torque seeks a matching set of proteins that are sequence-similar to the query proteins and span a connected region of the network, while allowing both insertions and deletions. The algorithm uses alternatively dynamic programming, integer linear programming, and fast heuristics. We tested Torque with queries from yeast, fly, and human, and with queries from less studied species, where only topology-free algorithms apply. Torque detects many more matches than previous approaches, while giving results that are highly functionally coherent. Our implementation is available to the public via a web interface.

Acknowledgements

First and foremost, I would like to thank my advisors, Prof. Roded Sharan and Prof. Ron Shamir. Their involvement and instruction at every step of the way were invaluable. I had the privilege of learning from both of them, and their combined wisdom, patience, and thoroughness contributed greatly to this work and to my own evolvement as a scientist.

I would also like to thank my collaborators, Dr. Falk Hüffner and Prof. Richard Karp, for their valuable insights and contributions to this work. I am grateful to Igor Ulitsky and Nir Yosef for their help in navigating the maze of bioinformatics data and their helpful advice. Also, I'd like to thank the others in the lab: Chaim Linhart, Adi Maron-Katz, Seagull Shavit, Ofer Lavi, Guy Karlebach, and Michal Ozery-Flato for their great support, fun conversations, and friendship. Last but not least, I'd like to thank my family: my parents, my sister, and my brother for their love and support, and also to my wonderful boyfriend.

This study was supported in part by the Israel Science Foundation (Grant no. 385/06) and by the German-Israeli Foundation.

Contents

Chapter 1

Introduction and summary

Sequence-based searches have revolutionized modern biology, serving to infer gene function, homology relations, protein structure, and more. In the last few years, there has been an effort to generalize these techniques to the network level. In a *network querying* problem, one is given a small subnetwork, corresponding to a pathway or a complex of interest. The goal is to identify similar instances within a large target network, where similarity is measured in terms of sequence or interaction patterns. The matched instance is then assumed to correspond to a protein complex or pathway in the target network, and its properties and function may be inferred from our knowledge about the query. The focus thus far has been mostly on methods that require an exact match of edges in the query and target subnetworks. A limitation of these approaches is that they rely on precise information on the interaction pattern of the query pathway. However, this information is often missing. For example, hundreds of protein complexes have been reported in the literature for yeast [58], human [55], and other species, but for most of these complexes no information exists on their interaction patterns [72], motivating a topology-free approach for the querying problem.

This work introduces Torque (TOpology-free netwoRk QUErying), a novel approach for network querying that does not rely on knowledge of the query topology. The input to our method is a set of proteins, representing a complex or pathway of interest and a protein–protein interaction (PPI) network in which the search is to be conducted. The goal is to find matching sets of proteins that span connected regions in the network. The corresponding theoretical problem that we study is searching a colored graph for connected subgraphs whose vertices have distinct given colors. We provide dynamic programming algorithms that are polynomial when the complex size is bounded by a constant. Some of our algorithms utilize the color-coding paradigm [2]. In addition,

we provide an integer linear programming formulation of it. This formulation includes a novel way to describe subgraph connectivity constraints, which can be useful in other problems as well. The methods can handle edge weights, insertions of network vertices (that do not match any query protein), and deletions of query nodes. We also develop a fast heuristic approach to the problem. By using a combination of the three approaches, we can solve most query complexes of practical sizes within current networks in reasonable time.

We applied Torque to query about 600 known complexes of size 4–25 from a variety of species in the PPI networks of yeast, fly and human. We tested our algorithm both on queries from species for which a PPI network is available, where we compared it to the QNet [17] topology-based approach, and on queries from less studied species, where only topology-free algorithms apply, where we also compared to the methods of Lacroix et al. [37] and their MOTUS [36] software. Torque detected many more matches than QNet and MOTUS, while giving results that receive high functional coherence.

An extended abstract based on this thesis appeared in the proceedings of the 13th RECOMB conference [12]. An additional paper describing the Torque web-server appeared in [13].

The thesis is organized as follows: in Chapter 2 we present the necessary biological and theoretical background, review some of the relevant literature, and briefly introduce our model and its extensions. In Chapter 3 we present an exact algorithm for network querying based on dynamic programming, and an integer linear programming formulation of the problem. In Chapter 4 we describe several fast heuristics for different variants of the problem. In Chapter 5 we explain our implementation, expanding upon the preprocessing stage and highlighting the Torque web-server. In Chapter 6 we present and discuss our results for querying more than 600 in various PPI networks. Finally, in Chapter 7 we discuss the limitations of our method and suggest some possible future directions. Some technical details and experimental results are described in the Appendix.

Chapter 2

Preliminaries and Background

2.1 Biological background

The study of proteins and their interactions is very important in biology and medicine. Proteins play a key role in every cellular process, from signal transduction to gene expression regulation and metabolism. Protein–protein interactions (PPIs) are stable or temporary connections between proteins that take part in almost every level of cell function. For example, signaling proteins transfer signals from the exterior of a cell to its interior by interacting with one another.

Many approaches exist to detect and identify PPIs [27]. One standard method is co-immunoprecipitation [49, 39]. In standard immunoprecipitation, an antibody is used to isolate a specific protein from a solution. The process of co-immunoprecipitation (CoIP) begins in a similar manner. When a protein is being isolated, other proteins "stick" to it. These are its putative interaction partners, that can be determined using other methods (e. g., western blotting [14]). Another method is *two-hybrid screening*, or the *yeast two hybrid* (Y2H) system [15, 41]. Here, each of the two tested proteins is fused to a different domain. When the two domains bind together, they can form a transcriptional activator that goes on to transcribe a reporter gene, whose activation can be detected visually. Those two domains can bind only as a result of a physical interaction between the tested proteins. Thus, the activation of the reporter gene implies the existence of a physical interaction between the proteins. Y2H can be used to study the interaction between two predetermined proteins that are suspected to interact. Alternatively, given a protein (the bait), Y2H can find proteins that interact with it (its preys). More information about these and other methods can be found, e. g., in the Protein Interactions Guide [51].

The knowledge gained from these experiments can be used to reconstruct the *PPI*

network of a studied species. A PPI network is the complete set of the species' proteins and all the known binary interactions among them. Mathematically, it is a graph whose nodes are the proteins and whose edges are the PPIs. As more interactions are observed and experimentally validated, larger parts of the full underlying PPI network are uncovered. For some well-studied species many interactions are known (the yeast network, for example, contains about 40,000 interactions and 5,000 proteins), making it possible to visualize and analyze the PPI network [35, 25]. Naturally, the PPI networks of different species are related through evolution. Thus, when enough information is available about the network of some individual species, it might be possible to learn new information by comparing it to the networks of other species.

A protein complex is a group of two or more proteins, linked by PPIs, that together perform a particular role. Complexes form various types of molecular machinery, and perform a vast array of biological functions. Complexes are therefore essential to many biological processes, and their investigation is a central research area [35, 25]. Different complexes usually perform different functions, and the same complex can sometimes perform very different functions that depend on a variety of factors [25]. In addition, the same protein can participate in the formation of several complexes. The experimental methods for identifying complex members most commonly identify the proteins that take part in the complex, but not the individual interactions between pairs of proteins in the complex. For example, the CoIP method described above is commonly used to identify protein complexes by isolating a protein, identifying its interaction partners, and then repeating the process for some of the partner proteins. This process will yield a possible set of interaction partners for each protein, without specifying the interactions among the partners themselves. One consequence of this limitation is that, for many protein complexes, their internal structure — or *topology* — is not known. It is useful, therefore, to have methods that can study complexes without relying on their topology.

Proteins in related species that are similar due to shared ancestry are *orthologs* [1, Chapter 1]. One way of inferring orthology of proteins across species is via sequence similarity between their respective DNA or amino-acid sequences. This similarity can be quantified using methods of sequence alignment, where the sequences are arranged in a way that allows identifying similar, or conserved, regions and then scoring the similarity. A standard method for computing sequence alignment is BLAST (Basic Local Alignment Search Tool [3]), which aligns the sequences, computes a score called *S-score* of the alignment, and outputs the significance of the result as a number, the *e-value* (Expectation value). The e-value is defined as the number of different alignments with scores equivalent to or better than the S-score that are expected to occur in a database

search by chance. The lower the e-value, the more significant the score [44]. This score for similarity between proteins can be used as a building block in determining similarities between larger components of the networks, such as pathways or complexes [33].

2.2 Computational background

Several important concepts in theoretical computer science underlie algorithms detailed in this work.

The problems we attempt to solve are NP-complete problems, therefore it is widely assumed that there is no algorithm that solves them and runs in time polynomial in the size of the problem. Vast literature exists on NP-complete problems, see, e. g., [23].

Fortunately, some NP-complete problems can be solved efficiently in a parametric sense. Parameterized complexity studies the complexity of problems with multiple input parameters. The theory introduced by Downey and Fellows [18] originated from the observation that while NP-complete problems are believed to require exponential running time in terms of the input size, some of them can be solved in time polynomial in the input size and only exponential in some (small) parameter k. Thus, if k is small, such problems can still be considered "tractable". A problem is called *fixed-parameter tractable* (FPT) with respect to a parameter k if an instance of size n can be solved in $f(k) \cdot n^{O(1)}$ time, where f is an arbitrary function. Thus, fixed-parameter algorithms allow solving relatively large instances of NP-hard problems exactly, as long as the parameter value is modest. More information about fixed-parameter algorithms can be found, e. g., in [45].

One useful fixed-parameter algorithm is color-coding. Color-coding is a randomized method for searching graphs for specific connected substructures of bounded size k having certain properties, such as paths or trees. Specifically, Alon et al. [2] showed that finding the minimum weight simple path of length k in a graph can be done with high probability in $O(-\ln(\varepsilon)5.44^k m)$ time, where m is the number of edges and the probability of not finding an optimal solution is bounded by ε. They also give similar results for finding a k-vertex cycle or a subgraph of bounded treewidth and show that the method can be derandomized.

In color-coding, the solution space that needs to be searched is reduced from all possible subgraphs of size k (of which there are n^k) to $c^{O(k)}m$ for, e. g., paths, where m is the number of edges in the graph. Each vertex is assigned one of several colors at random. Then the new solution space contains only those possible subgraphs of size k that are *colorful*: each of the vertices has a different color. If the subgraph is colorful, then the task of finding it can be greatly simplified. The problem becomes fixed-parameter

tractable, and can be solved, for example, by dynamic programming (see below). Of course, there is no guarantee that the real solution will be colorful: Most of the time it will not be. Hence, the process of randomly coloring the vertices and applying the search algorithm must be repeated many times, until with sufficiently high probability (determined by ε) the solution will be colorful at least once. This number of times is exponential, but only in the size of the solution k. Therefore, the problem is FPT.

Dynamic Programming (DP) is a general strategy for solving complex problems by breaking them down into simpler subproblems. It has many uses, including the search of colored structures in color-coding algorithms as described above. A DP algorithm is applicable to problems where an optimal solution can be constructed efficiently from optimal solutions to its subproblems, and those solution can be reused [16]. A simple example is computing the Fibonacci sequence: Since $F(n) = F(n-1) + F(n-2)$, the problem of computing the n-th Fibonacci number $F(n)$, can be broken down into two subproblems, computing $F(n-1)$ and computing $F(n-2)$. This is a recursive process, since the subproblem of computing $F(n-1)$ can itself be broken down into the subproblem of computing $F(n-2)$ and $F(n-3)$. Thus, the computation of $F(n-2)$ is reused. The general DP paradigm is: Solve subproblems of the problem first, record these solutions for later, then put together a solution to the problem. Further information and examples can be found, e. g., in [16] or [6].

Another approach we apply to our problem is *integer linear programming*. Linear Programming (LP) is a technique for optimizing a value (such as a profit) under constraints (such as availability of raw material). More formally, it optimizes a linear objective function on the real variables $x_1, \ldots, x_n$, subject to linear equality and/or inequality constraints. Geometrically, this corresponds to finding a point in an n-dimensional space that maximizes the objective. Each inequality corresponds to a half-space, and their intersection forms a convex polyhedron; the solution has to be inside the polyhedron. Due to the linearity, the optimum will always be attained at a corner or a facet of the polyhedron. This is a key reason why it can be solved efficiently. The seminal method for solving LPs is the Simplex algorithm, due to George Dantzig [43]. The algorithm consists of "walking" on the edges of the polyhedron to find a solution. This algorithm has no polynomial worst case guarantees on running time, but is usually very efficient in practice. Newer polynomial algorithms for LP were developed using the ellipsoid [63] and interior point [32] methods. In Integer Linear Programming (ILP) we add the additional constraint that the values assigned to some of the variables must be integers. This makes the problem much harder, and the general ILP is NP-hard. Numerous methods for solving ILPs in practice were developed over the years, and implemented efficiently.

Some of them rely on branch-and-bound methods and good preprocessing. In this work we utilize the ILOG CPLEX solver(http://www.ilog.com/products/cplex/), a commercial optimization software package. More information on LP and ILP can be found, e. g., in [16] and [56, chapter 29].

2.3 The network querying problem

In this thesis we study the problem of *network querying* in the context of PPI networks. In a *network querying* problem, one is given a small subnetwork, corresponding to a pathway or a complex of interest. The goal is to identify similar instances in a large target network, where similarity is measured in terms of sequence or interaction patterns. Such a matched instance is then assumed to be a protein complex or pathway in the target network. If this match overlaps with a previously known complex or pathway, then new characteristics of this complex or pathway could be inferred from the query, such as its functionality. Otherwise, the matched complex or pathway could be novel. Hence, network querying can also help identify previously unknown complexes. We now sketch problem formulations that have been used in the literature so far. We will give a formal definition of our models in Section 2.5.

In its simplest form, when ignoring sequence similarity, and requiring exact match of edges in the query and target subnetworks, this problem corresponds to the NP-hard Subgraph Isomorphism problem. This basic model is usually augmented in several ways to make it more realistic.

First, restrictions are often imposed on the target nodes a query node can map to, e. g., based on sequence similarity of the corresponding proteins in the network, or other measures of similarity. In a restrictive formulation, every target vertex is similar to at most one query vertex. In a more flexible model, a single target vertex can be similar to many vertices in the query. Even more generally, there could be a score assigned to every possible match between a query vertex and a target vertex.

Further, it is often desirable to relax the required match by allowing to omit query vertices (*deletions*) or add additional vertices to the match to fulfill some constraints (*insertions*). This may correspond to evolutionary changes in the complex or pathway, or compensate for noisy or missing data. A variant of this is graph homeomorphism [28], which allows an arbitrary number of insertions of degree-2 vertices. Biologically, this model is a good description of, e. g., metabolic networks, where a single enzyme in one pathway may replace a few consecutively acting enzymes in another pathway [50].

Different scoring models are available that can take into account the similarity of

vertices, the confidence score of the interactions represented by the edges of the target network, and other factors.

Since our knowledge of the PPIs for many species is noisy and incomplete, and hence the data available on the topology of complexes are sparse, researchers have also began to consider methods for network querying that do not require a correspondence between the edges in the query and the match in the target network. To ensure the consistency of the matches they instead demand that the match is connected. This *topology-free* formulation is a main feature of our methods.

2.4 Previous work

Here we detail the previous literature regarding network querying and related problems. We begin by describing some network alignment algorithms and their relation to network querying. We then discuss network querying algorithms that rely on the topology of the query, first the exact algorithms that are applicable to queries of a limited topology and then the heuristic and enumerative algorithms that can be used to query general graphs. Finally, we describe methods for finding colorful motifs in networks that provide a base for our approach.

2.4.1 Network alignment

Network querying has firm connections to the problem of *network alignment* [33]. Network alignment is the process of comparing two networks, identifying regions of similarity and dissimilarity. The algorithms for network alignment can be divided into methods for *local* and *global* alignment. In global alignment the goal is to find a mapping between the nodes of the two networks, that maximizes some score. The score can take into account sequence similarity, topology, interaction probabilities, and other factors. This problem continues to attract a lot attention: In the recent ISMB 2009 conference, two papers present new approaches to global alignment. The paper by Zaslavskiy et al. [73] formulates PPI network alignment as a graph matching problem, and uses well-known matching algorithms to resolve it. Liao et al. [38] propose an algorithm based on spectral clustering that improves on the previous IsoRank [64] algorithm that matches proteins from different networks if they are similar to each other and their neighborhoods are similar to each other. Local alignment is concerned with finding subnetworks that are conserved across species and, hence, likely to represent true functional modules [60]. *Local alignment* was introduced in 2000 by Ogata et al. [46] with the aim of detecting functionally related

enzyme clusters (FRECs) by aligning metabolic networks. This heuristic alignment algorithm allows for insertions and deletions (termed *gaps* and *mismatches*) and a many-to-many similarity between the vertices of the two graphs being compared, hence the resulting matches were not necessarily isomorphic subgraphs. Another variant of the local network querying problem was studied by Narayanan and Karp [42]. They identify conserved modules by finding subgraphs between the two network graphs composed of *locally similar* proteins: sequence-similar to each and having similar neighborhoods. This definition of similarity is less rigid than those of other methods that require isomorphism, although there is no obvious way to include indels. An advantage of their approach is that its running time is provably polynomial.

Finally, a recent ISMB 2009 paper [29] presents an algorithm for local alignment that operates on similar interactions rather than similar proteins. The algorithm constructs a set of pairs of edges that can be aligned based on the domains that participate in the interactions, and then heuristically searches the graph induced on those edges for conserved complexes.

The first line of research that was concerned with network querying is an extension of the problem of local network alignment: instead of comparing two large networks, align a query and a network, and try to find the region in the network where the query is best aligned.

PathBLAST [34] is a web-tool and algorithm for network querying and local alignment. The core algorithm [33] is concerned with the local alignment of two networks from different species in order to identify their conserved pathways, while the web-server deals with network querying, which is our focus. The algorithms are topology-based: The input is a linear path of proteins from one species, and the PPI network of a different species. The goal is to find a set of matching paths in the network most similar to the query path, where a matching path must contain similar proteins to the query in the same order, and the similarity between proteins is measured by sequence similarity. Insertions, deletions and mismatches are allowed, and the scoring scheme takes into account interaction probabilities. The query algorithm is based on the PathBLAST network alignment algorithm: the two networks (or the query and the target network) are combined into a single alignment graph, where every vertex is a pair of sequence-similar proteins (one from the query, one from the network), and edges represent a conserved interaction or a mismatch. The goal is then to find a path in the graph. The alignment algorithm was applied to the yeast and H. pylori networks, resulting in the identification of many evolutionarily conserved pathways, of which a large number appear to have duplicated and specialized within yeast. The PathBLAST web-tool allows for querying

paths in many other networks as well, such as fly, human, and worm.

2.4.2 Exact Topology-based approaches to network querying

Following the success of network alignment methods, new approaches appeared that are specific to the network querying problem. A large body of work exists on querying methods that are topology-based. These methods rely strongly on the structure of the query and the network. The algorithms presented below are exact methods for network querying when the queries or target networks are limited to structures like paths and trees. Those algorithms have provable bounds on running time, and obtain results that are accurate and significant. However, they are limited due to their reliance on topology and may miss many potential matches.

Pinter et al. [50] propose MetaPathWayHunter, a topology based network querying tool that matches query trees in a larger collection of trees. The authors also provide a web implementation. The method compares the metabolic networks of two species, and can be applied to cross-species PPI querying. It begins by defining a local similarity between the network vertices (functional homology between the enzymes they represent) and a scoring scheme. Unlike our model, the similarity is part of the scoring scheme, thus allowing flexibility in finding solution that match proteins that may not be similar if the overall score is high. Insertions are not supported, and deletions are possible only for vertices of degree 2, and only in the large target tree and not in the query. The metabolic pathway alignment engine is based on the subtree homeomorphism model, and thus can be solved in polynomial time. The paper then introduces a novel algorithm for finding the best-matching subtree. The algorithm is a dynamic programming, bottom up procedure that finds the best score, first for pairs, and then for subtrees of increasing size, until a solution is reached. The method can be extended to directed, multi-source trees which may be better models in some scenarios. The running time is $O(n_2^2 n_1 / \log n_2 + n_2 n_1 \log n_1)$, where n_1 is the size of the large tree and n_2 is the size of the query. The advantage of the method is in its flexibility in scoring, while its main limitation is its restriction to querying trees in forests, rather than general graphs. The authors applied the approach to the genome-scale metabolic networks of bacterium E. coli and yeast, resulting in many statistically significant, confirmed matches.

QPath [62] is a fixed-parameter algorithm for querying linear paths in networks. It matches query paths to paths in the target network where the proteins are sequence-similar to those of the query proteins, in the same order, while allowing insertions and deletions. The QPath algorithm is based on color-coding and dynamic programming,

utilizing similar principles to those upon we later build our own algorithms. The dynamic programming formula incorporates a scoring scheme that takes into account the sequence similarity, the number of insertions and the number of deletions. The running time is $\ln(\frac{n}{\varepsilon})2^{O(k+N_{ins})}mN_{del}$, where ε is the probability of error, N_{ins} and N_{del} are the upper bounds on the number of allowed insertions and deletions, n, m are, respectively, the number of vertices and edges in the network, and k is the size of the query complex. QPath was tested with cross-species queries on networks from yeast, human, and fly to obtain tens of significant, functionally coherent matches. A substantially faster implementation is given by [31].

QPath was succeeded by QNet [17]. QNet is an exact fixed-parameter algorithm for PPI network querying when the query is a tree with a known topology, and is thus applicable to more general queries than QPath. QNet uses dynamic programming and color coding, and finds homeomorphic matches as in MetaPathwayHunter [50]. The algorithm is supplemented by a heuristic for reducing the number of iterations in the cases of queries whose protein members tend to have non-overlapping sets of homologs. Since the algorithm does not handle general graph queries, but only trees, QNet adds a preprocessing step that produces several possible tree structures (spanning trees) on the query proteins from the known interactions in the network of the query species. Each of those trees is then queried in the target PPI network and the best solution is returned. The running time of this algorithm is $2^{O(k)}m$. QNet presents an additional algorithm for queries of bounded treewidth, also with known topology. It uses a tree decomposition to transform the query graph to a tree with "supernodes", each containing a small subset of the original query nodes, and uses a similar dynamic programming algorithm to match this transformed graph. The running time is $2^{O(k)}n^{t+1}$, where t is a bound on the treewidth of the query graph (finding the exact treewidth of a general graph is an NP-hard problem). This variant of the algorithm has not been implemented at this time. QNet was used to query yeast complexes from MIPS in the fly network, resulting in many identified matches that point to strong conservation between the two species.

PADA1 [10] is an alternative network querying algorithm, which attempts to extend the QNet algorithm to more general query graphs. Similarly to QNet, PADA1 transforms the query into a tree and queries that tree, allowing insertions and deletions. While both QNet and PADA1 exploit the fact that the queries are tree-like, while QNet assumes they have small treewidth, PADA1 assumes they have a small feedback set (vertices to delete to make the graph into a tree). The algorithm transforms the graph into a tree by iteratively finding a cycle, duplicating a node of the cycle, and then breaking the cycle by deleting an edge from it. The result is a tree that contains duplicated vertices, from which

the original graph can be easily reconstructed. The time complexity of the algorithm depends on the number of those duplicated vertices, which translates to the size of the feedback vertex set. Once the query is transformed, a dynamic programming similar to QNet is applied; it seeks a best match for the query, mapping all the duplicates of a node to the same network vertex. The running time is $2^{O(k)}n^{|F|}$ where F is the feedback set. The theoretical running times of QNet and PADA1 are difficult to compare since they depend on the graph treewidth and the size of the feedback set respectively. The authors claim, however, that their method is fast in practice. PADA1 was tested on the data sets proposed by QNet, and obtained comparable results in the number of queries matched.

2.4.3 Other Topology-based approaches to querying general graphs

The following methods apply enumerative or heuristics-based algorithms for matching general subgraphs in target graph. Since such algorithms can have high running times in practice, they are usually applied to searching small subgraphs, of size 2–5.

GraphFind [21] is an algorithm for identifying exact and approximate matches of small graphs in large databases of graphs, or in a single large graph. It is also implemented as a Cytoscape [59] plugin called NetMatch [20]. The input is a labeled query graph and the output is the set of its occurrences as a subgraph in a large, labeled target graph, or in the graph database. The occurrence must match both in topology and in the labels, allowing some mismatches. Since this requires solving the Subgraph Isomorphism problem, heuristic methods are used to reduce the search space by filtering the database and then using well-known methods for approximate matching to find the best candidates from the filtered ones. The filtering is done via graph indexing, where structural features of the target and query graphs such as short paths (length 2 or 3) are extracted and compared. The experimental framework for querying a single graph in a much larger graph, as relevant to our application, was to-date performed only on synthetic data.

Yang and Sze [70] propose a method for path queries and an enumerative method for general ones. They relax the matching requirements by allowing query vertices to repeat in the solution, which seems to make the problem easier. This allows them to simplify the problem by making the graph acyclic. They thus reduce the path matching problem to the polynomial problem of finding a longest weighted path in a directed acyclic graph. The general graph query is solved by first enumerating all possible deletions and then finding matches by brute force. Since the method is based on exhaustive enumeration, it can handle only queries with few possible similarities between query and network vertices.

Several methods propose a more general approach to pattern finding in networks which is not exclusively motivated by network querying, but can be easily adapted to the problem. Sohler and Zimmer [65] developed a general framework for subnetwork querying. Their network query is differently defined and motivated: a query is some small labeled graph corresponding to a hypothesis one has about the existence of some structure in the labeled network. Their model then allows for, e. g., edges in the query matching paths in the target, and single query proteins matching multiple target proteins. The problem is then similar to the labeled Subgraph Isomorphism problem as above, under some relaxations. The method employed by the authors is based on translating the problem to that of finding a clique in an appropriately defined graph. Due to its complexity, their method is applicable only to very small queries.

NetGrep [5] is a web-tool for identifying subgraphs of interest in the network, where those subgraphs match "network schemas": small graphs where the nodes have certain properties, such as functionality, or putative domains, and the edges between them correspond to interactions. The matched subgraph in the network must have the same topology as the network schema (allowing some mismatches), making this approach structure-dependent as well. The algorithm used is based on heuristic pruning of the possible solution space. A major advantage of this method is its running time, which is fast due to the emphasis on implementation and the highly specialized nature of their small queries.

2.4.4 Topology-free methods

Another line of research that can be applied to network querying is searching a vertex-colored network for *colored motifs*. The query to these methods is a set of k colors and the goal is to search the network for a subgraph that is both connected and colorful, with no other restriction on the topology of the match. A first algorithm for searching networks for such connected motifs was given by Lacroix et al. [37]. Similarly to the methods we present in this work, the motif is modeled as a set of colors and the vertices of the graph are colored by the same colors. The goal is to find a colorful, connected subgraph. The authors propose a brute-force algorithm that traverses the solution space using breadth-first-search in an attempt to find a colorful subgraph. The model allows for insertions, adding an additional parameter that restricts the number of sequential insertion vertices allowed between a pair of matched vertices. The authors do not consider the variant of multiple colors per vertex and do not allow deletions. This topology-free algorithm has been applied to metabolic networks but is applicable to cross-species PPI

network querying. It has so far been attempted on small (sizes 2–5) queries. Due to the algorithm's heuristic nature, no theoretical bound on the running time can be given. MOTUS [36], the software implementing this algorithm, is available for download and web-use.

Another theoretical viewpoint is presented by Betzler et al. [8]. The authors seek a connected colorful subgraph as defined above. They give a theoretical framework similar to the one we propose in this thesis, providing fixed-parameter algorithms whose running time is improved by a subset convolution trick (see [9] for details). The problem is then solved using a dynamic programming algorithm very similar to our own, with the same running time. The extended model proposed does not explicitly include insertions and deletions, but can be solved by a fixed-parameter algorithm with running time $O(|\ln(\epsilon)|4.32^k k^2 m)$ that finds a subgraph with up to d connected components for some d where the subgraph is colored by the colors of the motif. The authors also present a variant that parallels our multiple colors per vertex variant — list coloring. They suggest a color-coding based algorithm that disregards edge weights, with a running time $O(|\ln(\epsilon)|10.88^k m)$, which is prohibitive in practice. An extension is also provided for the case when the color motif is a multiset and can contain more than one instance of the same color. The paper does not include any implementation or experiments.

2.5 Definition of problem variants

In this section we present the problem variants formally. All these variants will be analyzed in subsequent chapters.

Preliminaries Let $G = (V, E)$ be a PPI network where vertices represent proteins and edges correspond to PPIs. Denote $|V| = n$ and $|E| = m$. For a vertex v, let $N(v)$ denote the set of its neighbors, i. e., $N(v) = \{u : (u, v) \in E\}$. For two disjoint sets S_1 and S_2, we write $S_1 \uplus S_2$ for their union $S_1 \cup S_2$. We denote by $G[K]$ the subgraph of G induced by the vertex set K.

Given a set of colors $C = \{1, 2, \ldots, k\}$, a *coloring constraint* function $\Gamma : V \to 2^C$ associates with each $v \in V$ a subset of colors $\Gamma(v) \subseteq C$. For $S \subseteq C$, we define a subset $H \subseteq V$ as *S-colorful* if $|H| = |S|$ and there is a function c that assigns each $v \in H$ a color from $\Gamma(v)$, such that there is exactly one vertex in H of each color in S.

Problem variants The central problem that we study is the *Colorful Connected Subgraph*:

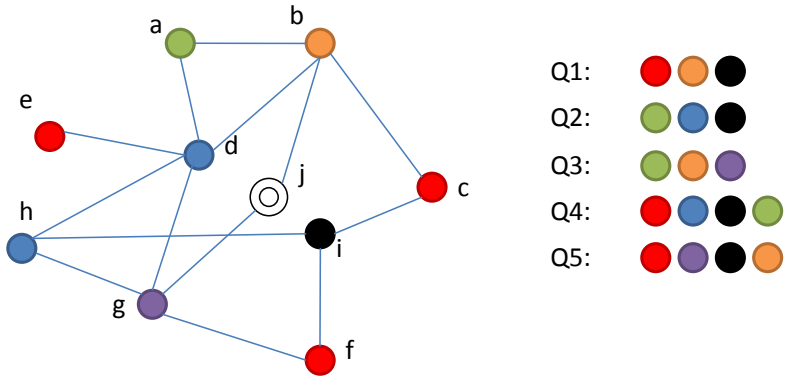


Figure 2.1: Network query problems. Left: the network, where vertex j is non-colored. Right: queries. For the basic problem disallowing indels, Q1 is solved by $\{c, b, i\}$, while Q2 and Q4 have no solution. When allowing a single arbitrary insertion, Q2 has solution $\{a, d, h, i\}$ and Q4 has the solution $\{a, b, c, d, i\}$. When allowing a single special insertion, Q3 has the solution $\{a, b, g, j\}$. When allowing one deletion, Q2 has the solutions $\{a, d\}$, $\{i, f\}$. When allowing repeated nodes and no indels, Q5 has the solution $\{b, c, i, f, g\}$.

Problem 1 (Colorful Connected Subgraph). *Given a graph* $G = (V, E)$*, a color set* C*, and a coloring constraint function* $\Gamma : V \to 2^C$*, is there a connected subgraph of* G *that is* C*-colorful?*

This problem was shown to be NP-complete by Fellows et al. [19], even for the case of trees of maximum degree 3. Here we provide fixed-parameter algorithms for several variants of this problem, where the parameter is the number of colors $|C| = k$.

We first consider the *single color per vertex* variant, which allows only coloring constraint functions that associate each $v \in V$ with a single color. In this case, the input is a graph where each vertex is assigned a color from C, and we aim to find a connected subgraph having exactly one vertex of each color. We later extend this to *multiple colors per vertex* variant, where we give a reduction from the general case to the single color case. These variants and the others we present here can all be generalized to the case where *the edges are weighted*, and we seek the maximum-weight solution.

Further variants of the problem add flexibility to the model by allowing insertion and deletions (indels). We would like to allow deletions of query proteins that cannot be matched and insertions of network proteins that assist in connecting matched vertices. When allowing insertions, there are several problem variants to consider (see Figure 2.1).

In the first variant, some network vertices are not assigned a color, and only non-colored vertices can be inserted. For convenience, assign non-colored vertices the color 0. Let us call such insertions *special*.

Definition 1. *An* S-colorful solution allowing special insertions *is a connected subgraph*

$H \subseteq G$, *where* $\exists H' \subseteq H$ *such that* $V(H')$ *is* S*-colorful and all other vertices of* H *are non-colored.*

In a second variant of insertion handling, any vertex can be inserted (rather than only non-colored ones). We call such insertions *arbitrary*. The exact methods we propose in the next chapter support solutions with deletions and the two types of insertions.

Chapter 3

Exact algorithms

3.1 Dynamic Programming

3.1.1 Colorful Connected Subgraph

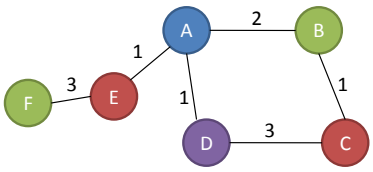
In this section we show how to solve Problem 1 using a randomized DP approach. We first consider the variant where the coloring constraint functions associate each $v \in V$ with a single color. In this case, the input is a graph where each vertex is assigned a color from C, and we aim to find a connected subgraph having exactly one vertex of each color.

Since every connected subgraph has a spanning tree, it suffices to look for colorful trees. As we noted in the previous chapter, this problem has been studied by Scott et al. [57] in another context, as well as by Betzler et al. [8]. We provide a DP algorithm, which is the unweighted version of the algorithm given by Scott et al. [57]. We construct a table B with rows corresponding to vertices and columns corresponding to subsets $C' \subseteq C$. We define $B(v, S)$= *True* if there exists in G a subtree rooted at v that is S-colorful, and *False* otherwise. For $S = \{\gamma\}$ and $v \in V$ we initialize $B(v, \gamma)$ =*True* if and only if $\Gamma(v) = \{\gamma\}$. Other entries of B can be computed using the following recurrence:

$$B(v, S) = \bigvee_{\substack{u \in N(v) \\ S_1 \uplus S_2 = S \\ \Gamma(v) \in S_1, \Gamma(u) \in S_2}} B(v, S_1) \wedge B(u, S_2), \tag{3.1}$$

See Figure 3.1 for an example execution of the algorithm and DP table. The algorithm runs in $O(3^k m)$ time[1]. One can easily generalize (3.1) to the weighted case, where each edge is assigned a weight, and the heaviest tree is sought. In this case, the algorithm

[1]It can be further reduced to $O(2^k m)$ using the techniques of Björklund et al. [9]; however, this version cannot be generalized to the weighted case, so we do not use it.



A	0	---	---	---	1	2	1	---	---	---	4	3	3	---	6
B	---	---	0	---	---	2	---	---	---	1	---	3	3	4	6
C	---	---	---	0	---	---	---	---	3	1	4	3	---	4	6
D	---	0	---	---	1	---	---	---	3	---	4	---	3	4	6
E	---	---	---	0	---	---	1	---	---	3	2	4	---	---	5
F	---	---	0	---	---	---	---	---	---	3	---	4	---	---	5

Figure 3.1: A single run of the dynamic programming. Top: a vertex-colored graph with weighted edges. Bottom: The dynamic programming the table. The columns correspond to the possible color subsets, and the rows correspond to the vertices. Each cell (v, S) contains the score of the maximum-weight tree rooted in v and colored exactly by S. For easier tracking of the algorithm, the values in the cells in the stages where there were several trees to choose from are underlined.

finds, for each vertex, the heaviest (maximum total edge weight) colorful subtree rooted at it. We note that this does not guarantee finding the heaviest subgraph containing a given vertex. The latter variant seems not to be amenable to this approach, unless some structure is assumed on the heaviest subgraph (such as being of bounded treewidth).

Initialization: For $S = \{\gamma\}$ define $B(v, \gamma) = 0$ if and only if $\Gamma(v) = \gamma$, $B(v, \gamma) = -\infty$ otherwise.

Recursion:

$$B(v, S) = \max_{\substack{u \in N(v) \\ S_1 \uplus S_2 = S \\ \Gamma(v) \in S_1, \Gamma(u) \in S_2}} B(v, S_1) + B(u, S_2) + w(u, v), \tag{3.2}$$

where $B(v, S)$ is now a real number instead of a Boolean value. The weight of an optimum match is given by $\max_v B(v, C)$.

3.1.2 Indels

We now consider extensions to the algorithm that include deletions, special insertions and arbitrary insertions.

Deletions. Deletions can be directly handled by the DP algorithm: If no C-colorful solution was found, then $B(v, C) =$*False* for all v. Allowing up to N_{del} deletions can be done by scanning the entries of B. If there exists $\hat{C} \subseteq C$ such that $|\hat{C}| \geqslant |C| - N_{del}$, and $B(v, \hat{C}) =$*True* then a valid solution exists.

Special insertions. An obvious extension of the DP algorithm to handle up to N_{ins} special insertions is based on the color-coding paradigm of Alon et al. [2]: Randomly color the non-colored vertices with N_{ins} new colors and use DP to look for colorful trees. This procedure is repeated a sufficient number of times to ensure that every tree is colorful with high probability. However, the running time increases by a factor of $(3e)^{N_{ins}}$. We provide a more efficient solution below.

Theorem 1. *Finding a C-colorful connected subgraph with up to N_{ins} special insertions can be solved in $O(3^k m N_{ins})$ time.*

Proof. We extend the DP table to represent also the number of special insertions used in an intermediate solution. Formally, $B(v, S, j)$ if and only if there is an S-colorful subtree rooted at v that allows j special insertions, and j is the minimal number of insertions possible. Here j ranges between 0 and N_{ins}. We initialize the table by setting all entries to *False*, except: (i) For $\gamma \neq 0$, $B(v, \{\gamma\}, 0)$ if and only if $\Gamma(v) = \gamma$; and (ii) if $\Gamma(v) = 0$, $B(v, \emptyset, 1)$. Entries for which $|S| \geqslant 1$ and $j > 0$ are then computed using the following recurrence:

$$B(v, S, j) = \Big[\bigvee_{\substack{u \in N(v) \\ S_1 \uplus S_2 = S \\ j_1 + j_2 = j}} B(v, S_1, j_1) \wedge B(u, S_2, j_2) \Big] \wedge \forall j' < j : \neg B(v, S, j') \tag{3.3}$$

We prove correctness by induction on $|S|$ and j. The cases $j = 0$ and $S = \emptyset$ are immediate. Therefore, consider $j > 0$ and as the first case $S = \{\gamma\}$ (i. e., $|S| = 1$). By definition:

$$\begin{aligned} B(v, \{\gamma\}, j) \iff \exists u \in N(v), S_1, S_2, j_1, j_2 : \\ B(v, S_1, j_1),\ B(u, S_2, j_2),\ S_1 \uplus S_2 = \{\gamma\},\ j_1 + j_2 = j,\ \forall j' < j : \neg B(v, \{\gamma\}, j') \end{aligned} \tag{3.4}$$

We prove the case $|S| = 1, j > 0$ by induction over j. Assuming there are u, S_1, S_2, j_1, j_2 as above, then S_1 cannot be $\{\gamma\}$ since $\forall j' < j : \neg B(v, \{\gamma\}, j')$. It follows that $S_1 = \emptyset$ and $S_2 = \{\gamma\}$, implying that $\Gamma(v) = 0$, $j_1 = 1$, and $j_2 = j - 1$ (see initialization of B). By the induction hypothesis on j, $B(u, \{\gamma\}, j - 1)$ implies that there exists a tree T rooted at u having one vertex colored γ and a minimal number of $j - 1$ non-colored vertices. Clearly, $v \notin T$. Otherwise, there will be a tree T' rooted in v having one vertex colored γ and

$j' < j$ special vertices, in contradiction to the minimality of j. Since $u \in N(v)$, then $T \uplus \{v\}$ is a tree having one vertex colored γ and j non-colored vertices, as desired.

It remains to handle the case where $|S| > 1$ by induction over $|S|$ and j. By definition:

$$B(v, S, j) \iff \exists u \in N(v), S_1, S_2, j_1, j_2 : \\ B(v, S_1, j_1),\ B(u, S_2, j_2),\ S_1 \uplus S_2 = S,\ j_1 + j_2 = j,\ \forall j' < j : \neg B(v, S, j') \quad (3.5)$$

Suppose such u, S_1, S_2, j_1, j_2 exist. Then by the induction hypothesis, there is a tree T_v rooted at v that is S_1-colorful and contains a minimal number j_1 of special vertices. Similarly, there is a tree T_u rooted at u that is S_2-colorful and contains a minimal number j_2 of special vertices. T_u and T_v are clearly disjoint: Otherwise, there would be another tree T' rooted at v which is S-colorful and contains $j' < j_1 + j_2$ special vertices, in contradiction to $\neg B(v, S, j')$. Since $u \in N(v)$, the union of these trees is $(S_1 \uplus S_2)$-colorful and has $j_1 + j_2$ special vertices, as desired.

To achieve the stated running time, we maintain an auxiliary function $t(v, S)$ which is evaluated to j when for the first time $B(v, S, j)$ is true for some j. In the recursion (3.3), we replace the condition $j_1 + j_2 = j$ by $t(v, S_1) + t(u, S_2) = j$. Since the table is $(N_{ins} + 1)$ times the size of the table in the basic case, the running time increases by a factor of N_{ins} compared to the basic case. □

Arbitrary insertions. In this second variant of insertion handling, any vertex can be inserted. We solve this variant by using the algorithm for the problem with special insertions as a black box. Instead of running the algorithm on the input graph G, we run it on an auxiliary graph $G' = (V', E')$, which is constructed as follows: Add a non-colored copy v^0 for each $v \in V$, and set $E' = E \cup \{(v^0, u) \mid (v, u) \in E\} \cup \{(v^0, u^0) \mid (v, u) \in E\}$. Asymptotically, this does not change the running time.

Theorem 2. *The above algorithm solves the arbitrary insertions variant in time* $O(N_{ins} 3^k m)$.

Proof. A match in G can be translated to a match in G' by replacing vertices of repeating colors with their copies. Conversely, let $\tilde{T} \subseteq G'$ be a subgraph satisfying the coloring constraints, having j special insertions. Let $U = \tilde{T} \cap V$ and $U' = \tilde{T} \setminus U$. The crucial observation is that $\tilde{T}$ does not contain a pair $\{v, v^0\}$, as otherwise we obtain a contradiction to the minimality of j (we can replace each edge (u, v^0) with an edge (u, v) and obtain a solution having $j - 1$ insertions). It follows that $U \cup \{v \mid v^0 \in U'\}$ is a valid match in G.

The bound on the running time follows from the observation that the size of G' is linear in the size of G. □

3.1.3 Multiple colors per vertex

We now turn to the more general case, where a color constraint function can associate each vertex with a set of colors and not just a single color. This problem arises when a target protein is homologous to several query proteins. Betzler et al. [8] gave a fixed-parameter algorithm for the problem, where the running time was increased by a factor of $(2e)^k$ compared to the case of single color constraints. Here we give an alternative fixed-parameter algorithm (coupled with some speedup heuristics). The basic idea is to reduce the problem to the single color case by randomly choosing a single valid color for every vertex. Our main effort is in computing an upper bound on the number of coloring iterations needed.

Define a *color graph* to be a bipartite graph $B = (V, C, E)$ where V is the set of network vertices, C is the set of colors and $(v, c) \in E \iff c \in \Gamma(v)$. Consider a possible match to the query; for clarity, we assume that this match does not contain insertions or deletions. Then we can prove the following bound:

Theorem 3. *The probability* P_c *for a subset of vertices of size* k *to become colorful in a random coloring is at least* $\frac{1}{k!}$.

Proof. Take a solution set $S = \{v_1, \ldots, v_k\}$ and the color subgraph B' induced by $S \cup C$. Let d_i denote the degree of v_i in B'. Suppose, w.l.o.g., that $d_1 \geqslant \ldots \geqslant d_k$. Finally, denote by $\mathrm{Perm}(B')$ the number of perfect matchings in B'. Note that there is a 1-1 mapping between colorful colorings of S in G and perfect matchings in B'.

The probability in question is equal to the ratio of number of perfect matchings to the number of ways to color the vertices of the solution, i. e., $P_c = \frac{\mathrm{Perm}(B')}{\prod_{i=1}^{k} d_i}$. We claim that if $\mathrm{Perm}(B') > 0$ then $P_c \geqslant \frac{1}{k!}$. Indeed, under the conditions set above, Ostrand [47] proved the following bound:

$$\mathrm{Perm}(B') \geqslant \prod_{i=1}^{k} \max\{1, d_i - i + 1\} \tag{3.6}$$

Let $D_1 = \{d_i \mid d_i \geqslant i\}$ and $D_2 = \{d_i \mid d_i < i\}$. Observe that if $d_i \in D_1$, then $\frac{d_i - i + 1}{d_i} \geqslant \frac{1}{i}$. Otherwise, $d_i < i$ and $\frac{1}{d_i} > \frac{1}{i}$. Thus,

$$P_c \geqslant \frac{\prod_{i=1}^{k} \max\{1, d_i - i + 1\}}{\prod_{i=1}^{k} d_i} = \prod_{i \in D_1} \frac{d_i - i + 1}{d_i} \prod_{i \in D_2} \frac{1}{d_i} \geqslant \prod_{i \in D_1} \frac{1}{i} \prod_{i \in D_2} \frac{1}{i} = \prod_{i=1}^{k} \frac{1}{i} = \frac{1}{k!}. \tag{3.7}$$

□

Theorem 3 implies an overall running time of $O(k!3^k m N_{ins}^2)$ in the case of multiple color constraints. However, this bound is excessive in many instances, for the following reason. Let V' be a set of colored vertices. Following Dost et al. [17], define the *constraint graph* $G(V')$ as follows: the vertices are the colors, and an edge exists between two colors γ_1, γ_2 if there is a vertex v in V' such that $\gamma_1, \gamma_2 \in \Gamma(v)$. The resulting graph is then partitioned into connected components $P_1, P_2, \ldots, P_s$. This partition induces a partition of the colored network vertices into sets $Q_1, Q_2, \ldots, Q_s$, where all the vertices of Q_i can be colored only by colors from P_i. The expected number of iterations required for a P_i-sized subset of Q_i to become colorful is bounded by $|P_i|!$, and thus the number of iterations required for a solution of size k to become colorful is bounded by $\prod_{i=1}^{s} |P_i|!$. Therefore, the expected number of iterations of the algorithm is also bounded by $\prod_{i=1}^{s} |P_i|!$.

We can reduce this upper bound using the following two rules: (i) If for some i, the product of all color degrees in Q_i is smaller than $|P_i|!$, then it is beneficial to exhaustively enumerate all possible colorings of Q_i. (ii) By Hall's Theorem [40], if a graph has a perfect matching and its minimum degree is d, then it has at least $d!$ perfect matchings. Therefore, if the minimal color degree in Q_i is d, $\frac{|P_i|!}{d!}$ random iterations suffice.

In practice, we find that the above reductions bring the number of required iterations to under 100 in the majority of cases. This is considerably less than the theoretical bound proposed by Betzler et al. [8]. For example, when querying for human complexes in yeast, we obtain an improvement of at least 50% in the number of iterations in 45% of the complexes.

Preprocessing. We apply a preprocessing step in each iteration of the DP algorithm to reduce the graph size. Each such iteration assigns each vertex a single color from its set of allowed colors. We then look at the subgraph H induced by the set of vertices that are assigned a *private color*, i. e., a color that was not assigned to any other vertex. We split H into its connected components, and merge the vertices of each connected component to a new vertex. Each new vertex is assigned a new color, and this reduced set of new colors replaces the private colors. The DP algorithm then receives as input a reduced graph and set of colors, resulting in a faster running time and no effect on the quality of the results.

3.2 Integer Programming formulation

In this section we provide an ILP formulation of the network querying problem, allowing us to employ ILP solvers that on certain instances are faster than DP. Formally, the problem that we aim to solve using the ILP is Problem 1 (C-colorful Connected

SUBGRAPH) with exactly N_{ins} arbitrary insertions and exactly N_{del} arbitrary deletions. Further, we are given edge weights $\omega : E \to \mathbb{Q}$ and wish to find a vertex subset $K \subseteq V$ of size $t := k + N_{ins} - N_{del}$ that maximizes the total edge weight $\sum_{(v,w)\in E, v,w\in K} \omega_{vw}$.

We declare binary variables $\{c_v : v \in V\}$ that express whether a vertex v is selected into the complex K. It is easy to give constraints that ensure correct coloring; the difficulty is in expressing the connectivity. The idea is to find a flow[2] with $t-1$ selected vertices as sources of flow 1, and a selected sink r that drains a flow of $t-1$, while disallowing flow between non-selected vertices. We use the following variables:

$$\{c_v : v \in V\}, c_v \in \{0,1\} \quad \text{vertex } v \text{ is selected } (v \in K) \tag{3.8}$$

$$\{e_{vw} : (v,w) \in E, v < w\}, e_{vw} \in \{0,1\} \quad \text{edge } (v,w) \text{ is in } G[K] \tag{3.9}$$

$$\{r_v : v \in V\}, r_v \in \{0,1\} \quad \text{vertex } v \text{ is the sink} \tag{3.10}$$

$$\{f_{vw}, f_{wv} : (v,w) \in E\}, f_{vw}, f_{wv} \in \mathbb{Q} \quad \text{flow from } v \text{ to } w/w \text{ to } v \tag{3.11}$$

$$\{g_{v\gamma} : v \in V, \gamma \in \Gamma(v)\}, g_{v\gamma} \in \{0,1\} \quad \text{vertex } v \text{ has color } \gamma \tag{3.12}$$

and the following constraints

$$\sum_{v\in V} c_v = t \tag{3.13}$$

$$\sum_{v\in V} r_v = 1 \tag{3.14}$$

$$e_{vw} \leqslant c_v \wedge e_{vw} \leqslant c_w \quad \forall (v,w) \in E \tag{3.15}$$

$$2e_{vw} \geqslant c_v + c_w - 1 \quad \forall (v,w) \in E \tag{3.16}$$

$$f_{vw} = -f_{wv} \quad \forall (v,w) \in E \tag{3.17}$$

$$\sum_{w\in N(v)} f_{vw} = c_v - t r_v \quad \forall v \in V \tag{3.18}$$

$$f_{vw}, f_{wv} \leqslant (t-1)e_{vw} \quad \forall (v,w) \in E \tag{3.19}$$

[2] That is, a function $f : V \times V \to \mathbb{Q}$ that satisfies skew symmetry ($\forall v, w \in V : f(v,w) = -f(w,v)$) and flow conservation ($\sum_{w\in V} f(v,w) = 0$) for all vertices v except sources and sinks; see e. g. Cormen et al. [16] for an introduction on flows.

$$\sum_{\gamma \in \Gamma(v)} g_{v\gamma} \leqslant 1 \qquad \forall v \in V \tag{3.20}$$

$$\sum_{v \in V} g_{v\gamma} \leqslant 1 \qquad \forall \gamma \in C \tag{3.21}$$

$$\sum_{v \in V} \sum_{\gamma \in \Gamma(v)} g_{v\gamma} = t - N_{\text{ins}} \tag{3.22}$$

$$g_{v\gamma} \leqslant c_v \qquad \forall v \in V, \gamma \in \Gamma(v) \tag{3.23}$$

with the objective

$$\text{maximize} \sum_{(v,w) \in E} \omega_{vw} e_{vw}. \tag{3.24}$$

We now turn to proving the correctness of the formulation above.

Theorem 4. *The ILP defined by* (3.8)–(3.24) *correctly solves Problem 1.*

Proof. The integrality constraints (3.8) and (3.9) and the inequalities (3.15) and (3.16) ensure that $e_{vw} = 1$ if and only if $c_v = 1 \wedge c_w = 1$. Therefore, the two objectives match. It remains to show that the constraints of Problem 1 are met if and only if the constraints (3.13)–(3.23) are met.

"$\Rightarrow$": Given a solution of Problem 1, set c_v, e_{vw}, and $g_{v\gamma}$ as described in (3.8), (3.9), and (3.12), respectively. Select an arbitrary $r \in K$ and set $r_r = 1$ and $r_v = 0$ for each $v \in V, v \neq r$. Let T be a spanning tree of $G[K]$ (which exists because $G[K]$ is connected) with edges directed towards r. Set f_{vw} for $(v, w) \in E(T)$ to the number of vertices in the subtree of T rooted in v, and set $f_{wv} = -f_{vw}$ and $f_{vw} = 0$ for $v \notin K$ or $w \notin K$. It is then easy to verify that (3.13)–(3.23) hold.

"$\Leftarrow$": Given a solution of the ILP, let K, r, and c be defined by (3.8), (3.10) and (3.12), respectively. Because of (3.13), we have $|K| = t$, and because of (3.14), r is well-defined. Constraints (3.20) make g be well-defined, and (3.21) make sure it is colorful. Because of (3.22), there are exactly $t - N_{\text{ins}}$ colored vertices, and with (3.23) they must all be in K, implying that there are exactly N_{ins} uncolored vertices in K.

It remains to show that $G[K]$ is connected. For this, we show that every $v \neq r \in K$ has a path to r consisting only of vertices from K. Constraint (3.17) ensures flow skew symmetry. Constraint (3.18) ensures that the outgoing flow $\sum_{w \in N(v)} f_{vw}$ is 1 for $v \in K, v \neq r$ and 0 for $v \notin K$. For the root r, we always have $c_r = 1$, since (3.17) and (3.18) requires at least one edge with positive flow incident on r, and so (3.19) and (3.15) force $c_r = 1$. Thus, for the root r, the outgoing flow is $-(t-1)$, and f forms a valid flow with $(t-1)$ sources $K \setminus \{r\}$ and one sink r.

Consider now $v \neq r \in K$. Because of (3.18), v has at least one neighbor w with $f_{vw} \geqslant 1$. Because of (3.19), we have $w \in K$. We continue this way until reaching r or reaching a vertex w' for the second time. If we reached w' again, we can decrease the flow on all edges traversed after w' by 1, yielding another valid flow without violating any of (3.17), (3.18), and (3.19). This will eventually provide a path to r. □

Chapter 4

Heuristics

The similarity measure we employ between query and network proteins is sequence-based. Therefore, the distribution of colors among the vertices of the network is often far from uniform: some vertices can be assigned many colors, some are assigned a single color or two, and many others are not similar to any query node and thus receive no color at all. Also, different colors can match a different number of vertices, hence, some colors may appear once or twice in the network while others may appear many times. Here we propose several heuristic speedups that try to take advantage of the unique properties of the color distribution.

4.1 Shortest path based heuristic

We often observed that the majority of network proteins were not sufficiently sequence-similar to any query protein, and can be used only as "special" insertion vertices. Therefore, in practice, only a small fraction of the network proteins are colored. We thus developed a fast heuristic that solves problem 1 when the number of colored vertices is small, without allowing indels. We then use this method as a preliminary step, accepting the solutions it returns and running the DP or ILP algorithm in the cases where the heuristic returns no solution (either because it failed to find one, or because indels are required).

Our heuristic is based on a shortest-path algorithm to obtain a fast solution. Several fast iterations are run. During each iteration, one new vertex is added to the solution while a set of vertices is removed from the network, making the problem smaller. The model includes weighted edges and supports multiple colors per vertex. A "vertex of color c" in this context is a vertex that has c in its coloring constraints.

The algorithm maintains a partition of V into three sets, V_{in}, V_{out} and V_{open}. Starting

with $V_{open} = V$, vertices are greedily moved from V_{open} either to V_{in}, meaning that they are to be part of H, or to V_{out}, meaning that they are not to be part of H.

We define several dynamically changing variables, functions and properties determined by the partition $(V_{in}, V_{out}, V_{open})$.

- A vertex in V_{open} is *unique* if there is no other vertex of the same color in V_{open}.
- For v in V_{open}, let $d(v)$ denote the number of edges in a shortest path in $G[V_{in} \cup V_{open}]$ from v to the closest vertex of V_{in} (or ∞ if no such path exists.)
- For any color c such that V_{open} contains at least one vertex of color c, let $dist(c) = \min_{v|c\in\Gamma(v)} d(v)$. Let $c^* = \text{argmax}_c\, dist(c)$. Let v^* be a vertex with color c^* and $d(v^*) = dist(c^*)$. If there are several options for v^*, break ties according to the sum of the edge weights along the path from the candidate to V_{in}.

The following actions occur automatically at any point in the algorithm and take priority over the other steps of the algorithm.

- If a vertex becomes unique it is moved from V_{open} to V_{in};
- If a vertex is placed in V_{in}, all other vertices of the same color are placed in V_{out}, if they cannot be assigned any other colors that are not already represented in V_{in}.
- If $v \in V_{open}$ and $d(v) = \infty$ then v is placed in V_{out}.

We assume that there is at least one unique vertex at the beginning of the algorithm. If there are no unique vertices, we choose the least frequent color and run the heuristic several times, keeping only a single vertex of this color and removing the rest each time. A connected component of $G[V_{in} \cup V_{open}]$ is called *essential* if there is some color c such that the component contains a vertex of color c, and no other component contains a vertex of color c. The partition $(V_{in}, V_{out}, V_{open})$ is called *bad* if $G[V_{in} \cup V_{open}]$ contains two or more essential components. The algorithm tries to keep adding vertices to V_{in} without creating a bad partition. A path in G is called *feasible* if it can be colored in such a way that every vertex on it has a different color.

Now we are ready to describe a general step of the algorithm, in which V_{in}, V_{out}, and V_{open} are given. We may assume that there are no unique vertices in V_{open}, since they would have been automatically added to V_{in}.

General step: Execute the first case for which the precondition holds.

Case 1 $G[V_{in}]$ is connected and V_{open} is empty: Return SUCCESS

Case 2 $G[V_{in}]$ is connected or the degree $d(v^*)$ in the graph induced by $V_{in} \cup V_{open}$ is $\geqslant 3$: $V_{in} \leftarrow V_{in} \cup \{v^*\}$.

Case 3 There is no feasible path joining two connected components of $G[V_{in}]$: Return FAILURE

Choose a shortest feasible path joining two connected components of $G[V_{in}]$. This path can be found, e. g., by a simple depth search first (DFS) procedure, where we also keep a list of all possible color combinations that are feasible along the path. When a path is extended and it has several options of colors, combinations repeating a color are eliminated. If there are several options for this path, break ties according to the sum of the edge weights along it. Let the first vertex of V_{open} in that path be v.

Case 4 Adding v to V_{in} and the other vertices of the same color as v to V_{out} does not create a bad partition: $V_{in} \leftarrow V_{in} \cup \{v\}$

Case 5 There is a vertex w of the same color as v such that adding w to V_{in} does not create a bad partition: $V_{in} \leftarrow V_{in} \cup \{w\}$

Case 6 Return FAILURE

4.2 Color-frequency based heuristics

In many cases we observed that some of the colors can be assigned to only a few of the vertices ("infrequent colors"). We already take advantage of this fact in the preprocessing step for the DP (Section 3.1.3) in the single color per vertex variant. We thus attempted to extend this approach to obtain larger reductions even in the multiple colors per vertex model, while forfeiting optimality. This heuristic centers around the notion of merging vertices with infrequent colors, assigning those new vertices a new color, and running the ILP algorithm on the reduced graph with the new set of colors.

The heuristic consists of four steps:

1. Construct the set of *infrequent colors*: For each color, in increasing order of frequency, mark all vertices that can be assigned this color (regardless of the other colors they could be assigned). Vertices with no color (which can serve as insertion vertices) are considered to be frequent. Stop when 50% of the colored vertices are marked.

2. Construct H, the induced graph on the marked vertices with infrequent colors. Split H into connected components. Then greedily select a subset of the components such that the vertices in the selected components cover all the infrequent colors. More precisely, repeatedly add the component with the best ratio of new colors covered to number of vertices in the component. Each such step increases the size of a matching between the infrequent colors and the vertices in the selected components. Remove the unselected components.

3. Merge the vertices of each selected component. Each merged component gets a single new color, and the new set of colors is the frequent colors plus the new colors; the infrequent colors disappear. The new merged vertices have only one color, while the remaining vertices retain their multiple color assignment with the frequent colors.

4. The reduced instance is then given as input to the ILP solver. For this, we made several small modifications to the ILP formulation to accommodate for the fact that deleting a "component vertex" actually corresponds to several deletions.

Since the ILP solution can only select components as a whole, it can contain many redundant vertices. We get rid of them by taking as a new input graph only the graph induced by the solution vertices, and repeating the whole process until no more improvement is possible.

Unfortunately, the heuristic often failed to find the optimal solution; further, the speedup was not as pronounced as we hoped. Therefore, we do not include this heuristic in our pipeline. More details about the results as well as several other variants that we tested are described in the Appendix.

Chapter 5

Implementation

We implemented a pipeline called Torque for querying a complex given as a set of proteins from a source species in the PPI network of a target species. Torque runs with increasing number of allowed indels until a match is found or a pre-specified bound on the number of indels is reached. Matches are assigned a score based on edge weights, and the highest scoring match is finally output. The problem version addressed is the multiple colors per vertex model with arbitrary insertions. Before applying the computationally intensive DP or ILP methods, we try the fast heuristic based on shortest paths that does not allow indels but works well in practice on small instances (our tests show it returns a good match about 60% of the time, see below). We now describe the stages of the algorithm, the scoring scheme, and the parameters we used for our testing.

5.1 Preprocessing

A protein complex is specified as a set of proteins. We associate a distinct color with each query protein and define a corresponding coloring constraint function. Each vertex in the target network is associated with a subset of colors corresponding to the query proteins it is sequence-similar to (see Section 6.1). In practice, only 5% of the vertices on average are associated with one or more colors. The rest are treated as non-colored.

While some of the non-colored vertices can be used as insertion vertices, many are too far from any colored vertex to be feasible insertions under the given upper bound N_{ins}. Let v be a non-colored vertex, and let $d_0(u, v)$ be the the length (number of edges) of the shortest path between u and v where every vertex on the path is required to be non-colored. We keep v if there are colored vertices u_1, u_2 such that $d_0(u_1, v) + d_0(u_2, v) \leqslant N_{ins} + 1$, and the corresponding paths are vertex-disjoint. Otherwise, we remove v from the network. On the networks and complexes that we tested (see

below), subnetworks containing only colored vertices are usually of size less than 50, those allowing 1–2 insertions have 200–1000 vertices, and those allowing more insertions typically cover up to 99% of the network.

After computing the current subnetwork to search in, we partition it into its connected components and search in each one independently. A component is *feasible* if the color constraints of its vertices contain at least $k - N_{del}$ colors of the query. Next, we process feasible connected components of increasing size, searching for the highest scoring matched complex using any of our methods. We increase the number of indels, generating larger connected components, until a solution is found that contains the minimal number of insertions and deletions, where insertions are preferred over deletions, as they can be better attributed to incomplete data. Thus, a solution having two insertions and no deletions is preferred to one having a single insertion and a single deletion. A solution having no insertions and two deletions is preferred over a solution having three insertions. We have found that for the majority of queries, between one and two connected components had to be generated and tested.

5.2 Algorithm pipeline

When querying a connected component, its unique properties dictate which of our methods will find a match more efficiently. In the case when the connected component does not contain any special insertion vertices, we begin with the shortest-path heuristic described in Chapter 4, which terminates quickly. If it returns a match, we accept it as the solution for this connected component. Otherwise, we attempt either the DP or the ILP methods. While the ILP solution is faster on many cases, its running time is unpredictable, unlike the DP algorithm, which tends to be more stable in running time and to also guarantee optimality or determine that no solution exists. We therefore chose to use it only in certain cases: As a rule of thumb, when the number of vertices is very close to the number of colors k, and k is large, the ILP algorithm is preferable, since we observed that its running time is less sensitive to k, while the color-coding algorithm is exponential in k. Based on empirical tests, we apply the ILP algorithm whenever $2^{n-k} < 3^k$, where n is the size of the connected component. This condition was satisfied in about 2/3 of the connected components we tested. For the DP algorithm, we used the multiple colors per vertex model, generating color assignments for the vertices using the bounds described in Section 3.1.3, thus reducing the number of iterations required.

5.3 Scoring

We score a set of proteins matching a query using the approach of Sharan et al. [61]. Briefly, a match is assigned a likelihood ratio score, which measures its fit to a protein complex model (assuming that every two proteins in a complex should interact with high probability, independently of all other pairs) versus the chance that its connections in the target network arise at random. The protein complex model assumes that every two proteins in a complex should interact, independently of all other pairs, with high probability β. The random model assumes that the PPI graph was chosen uniformly at random from the collection of all graphs with the same vertex degrees as the observed one. This random model induces a probability of occurrence p_{uv} for each edge (u, v) of the graph. To accommodate for information on the reliability of interactions, the interaction status of every vertex pair is treated as a noisy observation, and its reliability is combined into the likelihood score. Overall, for a match U, the likelihood ratio score is expressed as a sum over the vertex pairs in the match:

$$\mathcal{L}(V) = \sum_{(u,v)\in U\times U} \log \frac{\beta Pr(O_{uv} \mid T_{uv}) + (1-\beta)Pr(O_{uv} \mid F_{uv})}{p_{uv}Pr(O_{uv} \mid T_{uv}) + (1-p_{uv})Pr(O_{uv} \mid F_{uv})}, \tag{5.1}$$

where O_{uv} denotes the set of experimental observations on the interaction status of u and v, T_{uv} denotes the event that u and v truly interact, and F_{uv} denotes the event the u and v do not interact. The computation of $Pr(O_{uv} \mid T_{uv})$ and $Pr(O_{uv} \mid F_{uv})$ is based on the reliability assigned to the interaction between u and v.

When applying the DP algorithm, we output the highest scoring tree rooted at each vertex. Then, for each such solution tree, we compute the score of the subgraph that is induced by its vertices, taking into account edges and non-edges, to produce a final score for this vertex set.

5.4 Web server

The TORQUE web-server (http://www.cs.tau.ac.il/~bnet/torque.html) implements the algorithms presented in this work for querying protein sets across species. This section describes the technical details on using the service.

5.4.1 Input

The input for TORQUE consists of:

1. A query set of proteins in species A.
2. Their protein sequences.
3. A PPI network for species B.
4. The sequences of the network proteins.

All inputs are in simple text format:

- The query set can be entered directly as a comma-delimited or whitespace-delimited list.
- Protein sequences are given in the standard FASTA format.
- The PPI network is given as a text file, where each row represents an interaction and contains the IDs of the interacting pair and a confidence value for it in the range $[0, 1]$.

It is possible to use a single FASTA file (input 2) for many queries, if it contains the sequences for all proteins in all queries. When the query field is left blank, Torque will use all the proteins in input 2 as the query. If input 1 contains Uniprot protein IDs (www.uniprot.org), their sequences need not be entered in input 2; instead, Torque automatically retrieves them from the Uniprot database. For several target species, the user needs not provide inputs 3 and 4. Currently, the server supports this option for the three target species *S. cerevisiae*, *D. melanogaster* and *H. sapiens*. The user can indicate one of these target species instead of providing inputs 3 and 4. Details on how these networks were constructed can be found in Chapter 6 and in the appendix.

The user can control two parameters of the algorithm, setting a trade-off between speed and sensitivity. First, the user can control the threshold for sequence similarities. Torque applies BLAST to find putative matches between query and target proteins. The user can set the threshold for BLAST similarity (e-value). By setting a lower threshold, less homologs will be identified for the query proteins, making the algorithm faster but less sensitive. The second parameter is a threshold for the confidence values of PPI network edges provided as part of input 3. Edges whose confidence value is lower than the threshold are discarded; hence, this parameter determines the sparsity of the target network and affects the number of possible matches and the running time.

5.4.2 Processing

The running time of Torque is typically a few minutes, but may be up to an hour, depending on the size and other properties of the query (See Chapter 6 for details).

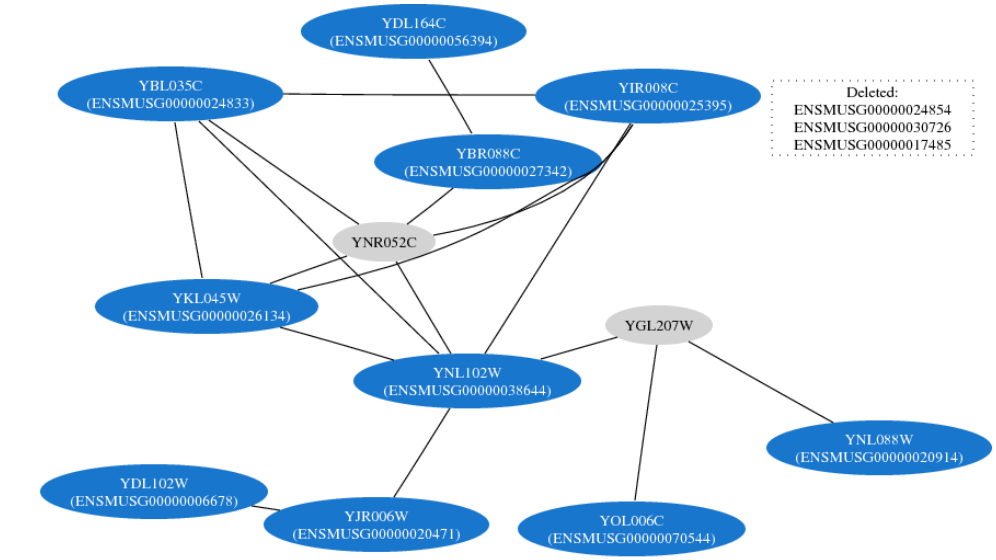


Figure 5.1: An example of the output of a TORQUE run. The query consists of 13 proteins. The match has 12 proteins with two insertions and three deletions.

If several queries are submitted to the server at the same time, they are queued and executed sequentially. Rather than waiting for the results online, they can be accessed later, in two ways:

- When a TORQUE job is started, it is assigned a 9-digit *job ID*. This ID can be used to access the results later from the main TORQUE page.
- Before submitting a query, the user may enter an email address. Once TORQUE has finished processing the query, the results will be sent to the email address provided. This process is done automatically and the email address is then discarded.

5.4.3 Output

The web-server generates a web page (see Figure 5.1) with the image of the top-scoring match for the query in the target network, as well as an auxiliary file in .sif format that can be viewed using the Cytoscape software [59]. The content of the .sif file includes, for each edge, a numerical value representing the confidence in the interaction it represents, as provided in the input. This value determines the thickness of the edge in the Cytoscape visualization. The image shows the subgraph induced by the top-scoring match in the PPI network. Each vertex is labelled with its protein name in the PPI network and its matching query protein, if such exists. Insertion vertices are shown in grey, and

proteins from the query for which there was no match in the solution (deletions) are listed separately.

Chapter 6

Experiments

We applied TORQUE to query protein complexes within the three largest eukaryotic PPI networks available to date: yeast (5430 proteins, 39936 interactions), fly (6650 proteins, 21275 interactions) and human (7915 proteins, 28972 interactions). As queries, we used six collections of protein complexes from different species: yeast, fly, human, bovine, mouse, and rat. The first three served us to validate our algorithm and compare it to the state-of-the-art QNet algorithm [17][1]. The last three, for which no large-scale PPI information exists, allowed us to explore the power of the algorithm in querying protein complexes for which no topology information is available. We compared our results for all data collections in yeast with those of MOTUS (http://genome.imim.es/~vlacroix/motus/), a program for querying for colorful motifs in networks that is based on the work of Lacroix et al. [37]. In the following we describe the data, the evaluation measures, and the results obtained.

6.1 Setup

Data acquisition. For yeast, fly and human, we used the networks recently published by Yosef et al. [71]. These networks were obtained using up-to-date PPI data gathered from several papers [66, 54, 67, 24, 35, 53] and from public databases [69, 22, 48]. High-throughput mass spectrometry data [24, 35] was translated into binary PPIs using the spoke model [4]. The networks can now be downloaded from http://www.cs.tau.ac.il/~bnet/TORQUE_Input_format.htm. Yeast complexes were downloaded from SGD [58] (Macromolecular Complex GO-Slim category). Fly complexes were obtained using the AmiGo [26] browser to collect all proteins annotated with GO:0043234 (protein complex).

[1] A comparison to GraphFind [21] was not feasible, since its interface does not allow automated execution of the more than 600 queries.

The complexes for all mammals (human, mouse, rat, bovine) were downloaded from the CORUM website [55]. The source of the FASTA data for the BLAST computations of all the species was the ENSEMBL database [30, Version 55], which we accessed through the interface of the BioMart web-site (http://www.biomart.org).

Parameter settings. Our tests were performed using the following set of parameters. We queried complexes of size 4–25. Query and network protein sequence similarities were evaluated using BLAST. We ran the *blastall* program with the standard settings on every pair of query and target species. The output of this process is a table of protein pairs, one from each network, and an e-value quantifying their similarity. For a vertex v and a color γ, we let $\gamma \in \Gamma(v)$ if the BLAST e-value obtained by comparing the sequences of v and the query protein corresponding to γ was less than 10^{-7}, the same threshold used by QNet [17]. Such protein pairs are *sequence similar*. For each complex, we allowed Torque to run at most one hour, and took the best solution up to that point. We set the maximum number of allowed insertions and deletions to 2 of each for small complexes (size < 7), 3 of each for medium sized complexes (size 8–14), and 4 of each for larger complexes.

6.2 Evaluation

To evaluate the quality of the matches, we used two measures: *functional coherence* and *specificity*. The first measure reports the percent of matches that are significantly functionally coherent with respect to the Gene Ontology (GO) annotation [68]. Note that while the query is functionally coherent, the reported matches may not be so due to permissive homology matching and the noise in the PPI data. To compute the functional coherence of a match, represented as a set of proteins, we used the GO TermFinder tool [11]. The p-values returned by the tool were further corrected for multiple match testing using the false discovery rate (FDR) procedure [7].

The second measure reports the specificity of the suggested solution, i. e., the fraction of matches that significantly overlap with a known protein complex. The significance of the overlap was evaluated using the hypergeometric distribution. The resulting p-value was compared to those obtained on 100 random sets of proteins of the same size to produce an empirical p-value. Those p-values were FDR-corrected for multiple testing. This specificity computation was applied to the matches that had a non-zero overlap with the collection of complexes to which they were compared. We also report separately those *novel matches* that had no overlap with known complexes. Although it is possible

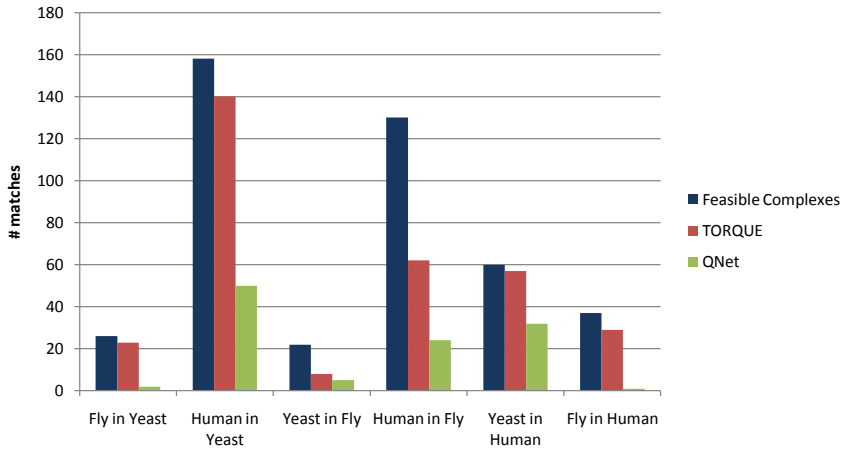


Figure 6.1: Comparison of number of matches for TORQUE and QNet.

that some of these non-overlapping matches are false positives, we believe that the high percentage of specific matches indicate that some—or most—of these are indeed novel complexes.

Comparison to QNet. Our first set of experiments focused on the yeast, fly and human networks and protein complex collections. For each of the three species, we queried its complexes in the networks of the other two species. As large-scale networks are available in this setting, we could compare ourselves to the QNet algorithm [17], which was designed to tackle topology-based queries. While exact topology for the query complexes is mostly unknown, QNet infers it by projecting the complexes onto the corresponding network. This results in a set of possible spanning trees for the complex that are hence provided to QNet as inputs. This makes QNet very dependent on the quality of the source network, in addition to the usual dependence on the quality and completeness of the target network. We used the original QNet code with the same machine setup and parameters as our algorithm: sequence similarity, insertions and deletions, and time limits.

A striking difference between TORQUE and QNet can be seen from the results in Figure 6.1—out of 433 feasible (as defined in Section 5.1) queries overall, TORQUE detected matches for 311 of them, while QNet found matches for 114 only. As we show below, this 170% gain in sensitivity did not harm the specificity of the results.

Next, we turned to evaluate the results using the functional coherence and specificity measures described above. The results for the three data sets are summarized in Table 6.2. As evident from the table, even though TORQUE matched many more queries, its results

		Functional coherence		Specificity		Novel matches	
Network	Complex	Torque	QNet	Torque	QNet	Torque	QNet
Yeast	Fly	23 (100%)	2 (100%)	19 (82%)	2 (100%)	7	0
	Human	134 (95%)	49 (98%)	119 (85%)	47 (94%)	8	2
Fly	Yeast	8 (100%)	3 (60%)	8 (100%)	4 (80%)	1	0
	Human	56 (90%)	21 (87%)	62 (100%)	23 (95%)	22	5
Human	Yeast	48 (84%)	25 (78%)	43 (75%)	23 (71%)	8	6
	Fly	21 (72%)	0 (—)	21 (72%)	0 (—)	7	0
Total		290	100	272	99	46	13

Table 6.1: Quality evaluation. The table lists the number and percentage of matches, out of all found matches that pass a significance threshold of 0.05, and the number of novel complexes detected.

exhibit higher functional coherence and similar specificity levels.

Comparison to MOTUS. MOTUS [36] is a software for colored motif search and inference in vertex colored graphs. Its inputs are a network (list of unweighted edges), where each vertex has a label, and a query motif, modeled as a set of labels. MOTUS then lists all connected occurrences of this motif in the network. The version we tested, provided by the authors in March 2009, does not include support for multiple labels per vertex (equivalent to our multiple colors per vertex), deletions and gaps (insertions), or weighted edges. Therefore, we tested MOTUS on our data in the cases where every network protein was similar to at most one query protein, implying that there is only a single color (or label) per vertex. We removed from the query all nodes that had no similar vertex in the network at all (a priori deletions). For each occurrence found, we looked at the subgraph induced by this occurrence on our original weighted-edges network, and scored the subgraph as described in section 5.3. We tested MOTUS on complexes from all species queried in the yeast PPI network. On this reduced set of instances, MOTUS found a match for 44 out of 220 feasible complexes ($\sim$ 20%), while Torque found a match in 199 ($\sim$ 90%) of the cases. MOTUS returned a match for most small motifs (sizes 3–5) that did not require insertions or deletions, while larger motifs or those that Torque matched using indels were usually not found within the time constraint of one hour and the memory constraint of 8 GB. Since MOTUS finds all possible occurrences of the motif, the optimal, highest scoring solution is output as well.

Topology-free queries. A unique characteristic of Torque is its ability to query protein complexes for which a topology is not known. Here we apply our algorithm to query, for the first time, sets of protein complexes of mouse (59 complexes), rat (55) and bovine

Network	Complex	#Feasible	#Matches	Functional coherence	Specificity	Novel matches
Yeast	Bovine	4	4	4	4	0
	Mouse	17	17	16	13	1
	Rat	23	20	19	9	6
Fly	Bovine	3	0	-	-	-
	Mouse	14	7	0	1	6
	Rat	34	21	17	7	14
Human	Bovine	4	4	2	1	0
	Mouse	48	46	32	24	6
	Rat	44	43	32	24	4
Total		191	162	122	83	37

Table 6.2: Statistics of querying protein complexes for which no topology information is available.

(10)—species for which no large scale PPI data are currently available. In Table 6.2, we present the results of querying these complexes within the networks of yeast, fly and human. As evident from the table, more than 95% of the feasible queries had a match, and the majority of the matches were functionally enriched or matched a known complex.

Indels and Running time. A major advantage of our approach is its flexibility in allowing insertions and deletions. Indeed, only 80 of our 482 identified matches required no insertions or deletions at all. We attempted to further explore the insertions and deletion in our results, see appendix A.4 for details. The number of allowed insertions affects the size of the components tested, and therefore affects the running time. In general, the running time of Torque depends on many factors: complex size, number of homologs for each query protein, and the size of the connected component tested. Figure 6.2 gives the running time distribution of all queries, i. e., the fraction that were completed within each time frame, for each of the three target networks. As the graph shows, more than 30% of queries from all networks were processed in 10 seconds or less, and 100 seconds are enough for more than 70% of queries. For a few of the queries the running time required is higher, however approximately 95% of the queries ended under the 1 hour time bound. Of the 624 feasible complexes, only 32 did not end in time. The three networks show similar behavior, and we attribute the minor differences among them to the difference in the quality and completeness of the networks.

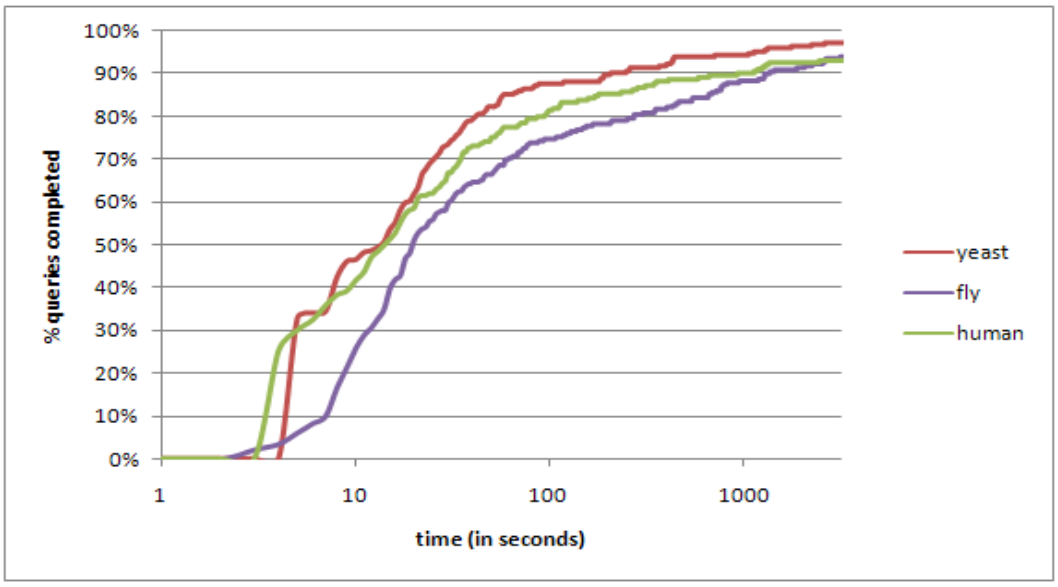


Figure 6.2: Runtime distribution of queries. The figure shows the cumulative percentage of queries completed within each running time. Queries were stopped after 3600 seconds, so 100% is not reached.

Chapter 7

Conclusions

7.1 Summary

In this study we presented a method and a tool for querying protein complexes within a PPI network, where no topology information about the complex is available. Our method combines three algorithms: a dynamic programming exact algorithm, an integer linear programming formulation, and a fast heuristic. Compared to a topology-based approach and to heuristics for colorful motif finding, our approach produces substantially more matches, while preserving high functional coherence. Thus, our tool is appropriate for a wide range of network query tasks, and in particular when interaction data on the query species are sparse or unavailable. Our method also suggests new complexes for which no prior experimental evidence is available, as they do not overlap any known complex. Checking these hypotheses experimentally is an interesting next step.

7.2 Limitations and possible extensions

Our approach has several limitations that we hope to address in future work.

- The running time for some queries is still prohibitively high, depending on the problem size and the number of colors (query size) and their distribution. We are examining additional heuristics in an attempt to bring the running time down further.

- The integer linear programming formulation proved very powerful, and usually performs well in practice, especially when commercial software was used. However, the performance depends on the availability of such software, and the specific

algorithm chosen by such software is often not transparent and hence its behavior is less predictable.

- Since our methods do not rely on the topology of the query, they do not depend on PPI data from the query species. However, our approach is sensitive to the quality of the PPI data in the target species, and specifically to the false negative rate in that network (fraction of true interactions that are still unknown). A true complex in the target network, which biologically is a match to the query complex, may not be found if the corresponding subnetwork is disconnected due to false negative edges. This can be seen, for example, in the fly network, which is noisier than the human and mouse networks: there are many feasible queries for which Torque finds no solution. False positive edges in the target network are less problematic, as they do not disrupt the connectivity (but can create spurious solutions).

- Extending the notion of similarity between query and target proteins might also improve the solutions. Currently we rely on sequence similarity, and a fixed threshold for sequence similarity is used. Combining other similarity measures such as structural similarity could be considered. Furthermore, replacing the threshold by a weighted model could improve the results.

On the theoretical side, several follow-up directions are of interest. The main problem that we addressed can be generalized also to allowing special insertions where colored vertices may repeat freely without penalty for reuse. The DP approach may be generalizable to cover that version as well, and we intend to explore it

We are also interested in continuing the development of the Torque web-server. We are considering adding more predefined data: networks and sequences for additional target species, when such high-quality data become available. Another interesting direction is allowing the user more control over the query by exposing more parameters: manually setting the number of insertions and deletions, specifying network proteins that must take part in the solution, etc. If the tool gains a faithful user-base, as we hope it will, integration with other platforms such as Cytoscape [59] should also be considered.

Bibliography

[1] B. Alberts and E. Al. *Molecular Biology of the Cell*. Garland Science, 2002. Cited on p. 4.

[2] N. Alon, R. Yuster, and U. Zwick. Color coding. *Journal of the ACM*, 42:844–856, 1995. Cited on pp. 1, 5, and 19.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990. Cited on p. 4.

[4] G. D. Bader and C. W. Hogue. Analyzing yeast protein-protein interaction data obtained from different sources. *Nature Biotechnology*, 20(10):991–997, 2002. Cited on p. 37.

[5] E. Banks, E. Nabieva, R. Peterson, and M. Singh. NetGrep: fast network schema searches in interactomes. *Genome Biology*, 9(9), 2008. Cited on p. 13.

[6] R. E. Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 1957. Cited on p. 6.

[7] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B (Methodological)*, 57(1):289–300, 1995. Cited on p. 38.

[8] N. Betzler, M. R. Fellows, C. Komusiewicz, and R. Niedermeier. Parameterized algorithms and hardness results for some graph motif problems. In *Proc. 19th CPM*, volume 5029 of *LNCS*, pages 31–43. Springer, 2008. Cited on pp. 14, 17, 21, and 22.

[9] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proc. 39th STOC*, pages 67–74, New York, 2007. Cited on pp. 14 and 17.

[10] G. Blin, F. Sikora, and S. Vialette. Querying protein-protein interaction networks. In *Proc. 5th ISBRA*, volume 5542 of *Lecture Notes in Computer Science*, pages 52–62. Springer, 2009. Cited on p. 11.

[11] E. I. Boyle, S. Weng, J. Gollub, H. Jin, D. Botstein, J. M. Cherry, and G. Sherlock. GO::TermFinder—open source software for accessing gene ontology information and finding significantly enriched gene ontology terms associated with a list of genes. *Bioinformatics*, 20(18):3710–3715, 2004. Cited on p. 38.

[12] S. Bruckner, F. Hüffner, R. M. Karp, R. Shamir, and R. Sharan. Topology-free querying of protein interaction networks. In *Proc. 13th RECOMB*, volume 5541 of *Lecture Notes in Bioinformatics*, pages 74–89. Springer, 2009. Cited on p. 2.

[13] S. Bruckner, F. Hüffner, R. M. Karp, R. Shamir, and R. Sharan. Torque: Topology-free querying of protein interaction networks. *Nucleic Acids Research*, 37(Web Server issue):W106–W108, July 2009. Cited on p. 2.

[14] N. W. Burnette. "Western Blotting": Electrophoretic transfer of proteins from sodium dodecyl sulfate-polyacrylamide gels to unmodified nitrocellulose and radiographic detection with antibody and radioiodinated protein A. *Analytical Biochemistry*, 112 (2):195–203, April 1981. Cited on p. 3.

[15] C. Chien, P. L. Bartel, R. Sternglanz, and S. Fields. The two-hybrid system: A method to identify and clone genes for proteins that interact with a protein of interest. *Proceedings of the National Academy of Sciences (USA)*, 88:9578–9582, 1991. Cited on p. 3.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001. Cited on pp. 6, 7, 23, and 54.

[17] B. Dost, T. Shlomi, N. Gupta, E. Ruppin, V. Bafna, and R. Sharan. QNet: A tool for querying protein interaction networks. *Journal of Computational Biology*, 15(7): 913–925, 2008. Cited on pp. 2, 11, 22, 37, 38, and 39.

[18] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141(1–2):109–131, 1995. Cited on p. 5.

[19] M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Borderlines for finding connected motifs in vertex-colored graphs. In *Proc. 34th ICALP*, volume 4596 of *LNCS*, pages 340–351. Springer, 2007. Cited on p. 15.

[20] A. Ferro, R. Giugno, G. Pigola, A. Pulvirenti, D. Skripin, G. D. Bader, and D. Shasha. NetMatch: a Cytoscape plugin for searching biological networks. *Bioinformatics*, 23 (7):910–912, 2007. Cited on p. 12.

[21] A. Ferro, R. Giugno, M. Mongiovì, A. Pulvirenti, D. Skripin, and D. Shasha. GraphFind: enhancing graph searching by low support data mining techniques. *BMC Bioinformatics*, 9 Suppl 4:1471–2105, 2008. Cited on pp. 12 and 37.

[22] FlyBase-Consortium. The FlyBase database of the Drosophila genome projects and community literature. *Nucleic Acids Research*, 31(1):172–175, 2003. Cited on p. 37.

[23] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. Cited on p. 5.

[24] A. C. Gavin, P. Aloy, P. Grandi, R. Krause, M. Boesche, M. Marzioch, C. Rau, et al. Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 440(7084): 631–636, 2006. Cited on p. 37.

[25] A.-C. Gavin, P. Aloy, P. Grandi, R. Krause, M. Boesche, M. Marzioch, C. Rau, et al. Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 2006. Cited on p. 4.

[26] GO Consortium. Amigo. http://amigo.geneontology.org/, Sept. 2008. Cited on p. 37.

[27] E. Golemis and P. D. Adams. *Protein-Protein Interactions – A Molecular Cloning Manual*. Cold Spring Harbor Laboratory Press, 2002. Cited on p. 3.

[28] J. Gross and J. Yellen. *Graph theory and its applications*. CRC Press, Inc., Boca Raton, FL, USA, 1999. Cited on p. 7.

[29] X. Guo and A. J. Hartemink. Domain-oriented edge-based alignment of protein interaction networks. *Bioinformatics*, 25(12):i240–1246, 2009. Cited on p. 9.

[30] T. Hubbard, D. Andrews, M. Caccamo, G. Cameron, Y. Chen, M. Clamp, L. Clarke, et al. Ensembl 2005. *Nucleic Acids Research*, 33 (Database Issue):D447–D453, 2005. URL http://nar.oxfordjournals.org/cgi/content/full/33/suppl_1/D447. Cited on p. 38.

[31] F. Hüffner, S. Wernicke, and T. Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008. Cited on p. 11.

[32] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proc. 16th STOC*, pages 302–311. ACM, 1984. Cited on p. 6.

[33] B. P. Kelley, R. Sharan, R. M. Karp, T. Sittler, D. E. Root, B. R. Stockwell, and T. Ideker. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Sciences USA*, 100(20):11394–11399, 2003. Cited on pp. 5, 8, and 9.

[34] B. P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B. R. Stockwell, and T. Ideker. PathBLAST: a tool for alignment of protein interaction networks. *Nucleic Acids Research*, 32(Web Server issue), July 2004. Cited on p. 9.

[35] N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, et al. Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. *Nature*, 440 (7084):637–643, 2006. Cited on pp. 4 and 37.

[36] V. Lacroix. Motus. http://genome.imim.es/~vlacroix/motus/, 2009. Cited on pp. 2, 14, and 40.

[37] V. Lacroix, C. Fernandes, and M. Sagot. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):360–368, 2006. Cited on pp. 2, 13, and 37.

[38] C.-S. Liao, K. Lu, M. Baym, R. Singh, and B. Berger. IsoRankN: spectral methods for global alignment of multiple protein networks. *Bioinformatics*, 25(12), 2009. Cited on p. 8.

[39] D. Liebler. *Introduction to Proteomics, Tools for the New Biology*. Humana Press, 2002. Cited on p. 3.

[40] L. Lovász and M. D. Plummer. *Matching Theory*, volume 29 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, 1986. Cited on p. 22.

[41] P. MacDonald. *Two-Hybrid systems: Methods and Protocols*. Methods in Molecular Biology. Humana Press, 2002. Cited on p. 3.

[42] M. Narayanan and R. M. Karp. Comparing protein interaction networks via a graph match-and-split algorithm. *Journal of Computational Biology*, 14(7):892–907, 2007. Cited on p. 9.

[43] J. C. Nash. The (Dantzig) simplex method for linear programming. *Computing in Science and Engineering*, 2(1):29–31, 2000. ISSN 1521-9615. Cited on p. 6.

[44] NCBI. NCBI glossary. http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/glossary2.html, 2004. Cited on p. 5.

[45] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. Cited on p. 5.

[46] H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Research*, 28(20):4021–4028, October 2000. Cited on p. 8.

[47] P. Ostrand. Systems of distinct representatives II. *Journal of Mathematical Analysis and Applications*, 32:1–4, 1970. Cited on p. 21.

[48] S. Peri, J. D. Navarro, R. Amanchy, T. Z. Kristiansen, C. K. Jonnalagadda, V. Surendranath, V. Niranjan, et al. Development of human protein reference database as an initial platform for approaching systems biology in humans. *Genome Research*, 13 (10):2363–2371, 2003. Cited on p. 37.

[49] E. Phizicky and S. Fields. Protein-protein interactions: Methods for detection and analysis. *Microbiological Reviews*, 1(59):94–123, 1995. Cited on p. 3.

[50] R. Y. Pinter, O. Rokhlenko, E. Yeger-Lotem, and M. Ziv-Ukelson. Alignment of metabolic pathways. *Bioinformatics*, 21(16):3401–3408, 2005. Cited on pp. 7, 10, and 11.

[51] Promega. Protein interactions guide. http://www.promega.com/guides/protein.interactions_guide/, 2007. Cited on p. 3.

[52] H. J. Prömel and A. Steger. *The Steiner Tree Problem: A Tour Through Graphs, Algorithms, and Complexity*. Vieweg Advanced Lectures in Mathematics, 2002. Cited on p. 54.

[53] T. Reguly, A. Breitkreutz, L. Boucher, B. J. Breitkreutz, G. C. Hon, C. L. Myers, A. Parsons, et al. Comprehensive curation and analysis of global interaction networks in Saccharomyces cerevisiae. *Journal of Biology*, 5(4):11, 2006. Cited on p. 37.

[54] J. F. Rual, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, N. Li, G. F. Berriz, et al. Towards a proteome-scale map of the human protein-protein interaction network. *Nature*, 437(7062):1173–1178, 2005. Cited on p. 37.

[55] A. Ruepp, B. Brauner, I. Dunger-Kaltenbach, G. Frishman, C. Montrone, M. Stransky, B. Waegele, et al. CORUM: the comprehensive resource of mammalian protein complexes. *Nucleic Acids Research*, 36(Database issue):D646–D650, 2008. Cited on pp. 1 and 38.

[56] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986. Cited on p. 7.

[57] J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of Computational Biology*, 13(2):133–144, 2006. Cited on p. 17.

[58] SGD project. Saccharomyces genome database. http://www.yeastgenome.org/, Sept. 2008. Cited on pp. 1 and 37.

[59] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, November 2003. Cited on pp. 12, 35, and 44.

[60] R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24(4):427–433, 2006. Cited on p. 8.

[61] R. Sharan, T. Ideker, B. P. Kelley, R. Shamir, and R. M. Karp. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. *Journal of Computational Biology*, 12(6):835–846, 2005. Cited on p. 33.

[62] T. Shlomi, D. Segal, E. Ruppin, and R. Sharan. QPath: a method for querying pathways in a protein-protein interaction network. *BMC Bioinformatics*, 7:199, 2006. Cited on p. 10.

[63] N. Shor. Cut-off method with space extension in convex programming problems. *Kibernetika*, 3:94–96, 1977. Cited on p. 6.

[64] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, 105(35):12763–12768, 2008. Cited on p. 8.

[65] F. Sohler and R. Zimmer. Identifying active transcription factors and kinases from expression data using pathway queries. *Bioinformatics*, 21(Suppl. 2):ii115–ii122, 2005. Cited on p. 13.

[66] C. A. Stanyon, G. Liu, B. A. Mangiola, N. Patel, L. Giot, B. Kuang, H. Zhang, et al. A drosophila protein-interaction map centered on cell-cycle regulators. *Genome Biology*, 5(12):R96, 2004. Cited on p. 37.

[67] U. Stelzl, U. Worm, M. Lalowski, C. Haenig, F. H. Brembeck, H. Goehler, M. Stroedicke, et al. A human protein-protein interaction network: a resource for annotating the proteome. *Cell*, 122(6):957–968, 2005. Cited on p. 37.

[68] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000. Cited on p. 38.

[69] I. Xenarios, L. Salwnski, J. X, P. Higney, K. S, and E. D. DIP, the database of interacting proteins: a research tool for studying cellular networks of protein interactions. *Nucleic Acids Research*, 30:303–5, 2002. Cited on p. 37.

[70] Q. Yang and S.-H. Sze. Path matching and graph matching in biological networks. *Journal of Computational Biology*, 14(1):56–67, 2007. Cited on p. 12.

[71] N. Yosef, M. Kupiec, E. Ruppin, and R. Sharan. A complex-centric view of protein network evolution. *Nucleic Acids Research*, 2009. To appear. Cited on p. 37.

[72] Yu et al. High-quality binary protein interaction map of the yeast interactome network. *Science*, 322(5898):104–110, 2008. Cited on p. 1.

[73] M. Zaslavskiy, F. Bach, and J.-P. Vert. Global alignment of protein–protein interaction networks by graph matching methods. *Bioinformatics*, 25(12):i259–1267, 2009. Cited on p. 8.

Appendix A

Supplemental Material

A.1 Query statistics

In the following we give additional statistics on our experiments from Chapter 6.

Figure A.1 shows the size distribution of all queries and of the feasible queries. A small number of queries of size > 18 are omitted. We see that the distribution is exponentially decreasing, reflecting the size distribution of complexes. We also see that the fraction of feasible queries is roughly the same (40-50%) for queries of size > 10.

Table A.1 shows the average size of connected components in which a solution was found, as a function of the number of insertions in the solution. As expected, the size of the component increases dramatically with the number of insertions, which explains the difficulty of handling a high number of insertions.

Figure A.2 displays the distribution of the number of indels in the solution as a function of the query size. We see a growing number of indels as a function of the query size (the numbers for query size > 10 are based on a small sample, and thus are less reliable, compare Figure A.1). Note that part of that effect is due to the constraints set by our algorithms for the number of allowed indels (compare Section 6.1).

Finally, Figure A.3 plots the success rate as a function of query size. Like in Figure A.1, we distinguish between the statistics for feasible queries and for all queries. The plot shows a general decreasing trend of the success rate with query size, for all queries. Interestingly, while only few of the queries of size greater than 13 are feasible, more of those are successfully solved. Again, this is based on a rather small sample.

A.2 Heuristics results

The heuristic we presented in Section 4.2 for reducing the problem size by merging vertices with infrequent colors did not improve the running time as we hoped. Indeed, the running time of this heuristic when tested on human complexes in yeast was slower than than the running time of ILP alone in more than 90% of the queries we tested, and in many cases, solutions were not found or were did not contain the optimal number of insertions and deletions. The problem is that even when selecting components optimally with respect to color coverage, we do not take distances between them into account, and therefore might need many extra vertices to connect them. This means that we often fail to find solutions with the allowed number of insertions. Possibly, an improved component selection process that takes distances into account might mitigate this. As far as the speedup is concerned, the speedup obtained by the reduction in vertices and colors was mostly offset by the amount of time the heuristic selection process required and the fact that we ran the ILP repeatedly. Thus, the average running time of the heuristic on the human queries in yeast was 37.2 seconds, while ILP ran on the same queries in 19.9 seconds on average.

A.3 Variants of the color-frequency based heuristic

In addition to the basic color-frequency based heuristic described in Section 4.2, we also considered several variants, by changing steps 3-4 of that heuristic.

Our goal was to reduce the dependency on the ILP solver to obtain an algorithm that can run without commercial software. Instead, we attempt to find a Steiner tree connecting the components using frequent color vertices, e. g., using some Steiner tree approximation algorithm [52]. Another alternative is to find a minimum spanning tree (MST) [16] on the complete graph whose vertices are the merged components and whose edges are weighted according to the length of the shortest path through the frequent color vertices between the components they connect. As stated above, the results of the ILP were unsatisfactory both in quality and running time. Since the ILP gives optimal solutions and is very fast according to our tests, this implies that the first two steps of the heuristic are the source of the problem. Thus we did not pursue this line of research further.

We considered adding a postprocessing heuristic to avoid the running time costs of repeating the heuristic several times on the progressively reduced solution. In this heuristic, we repeatedly prune the leaves of solution received after the first round. We

remove a leaf if there is still a matching between the set of query colors and the vertices of the solution when it is discarded. Naturally, removing a leaf could create a new leaf in the graph. Thus we run this process until all the leaves that remain are essential to the match. Regrettably, this heuristic did not perform as we had hoped, as the poor quality of the generated solutions outweighed the improvements to the running time.

A.4 Exploring indels

We provide a closer look at insertions and deletions and analyze their contribution to the results. As we stated previously, only 80 of our 482 identified matches required no insertions or deletions at all. The majority of solutions required 1–3 indels, while 39 matches required 4 or 5 indels. We distinguish between two types of deletions: *apriori* deletions are query proteins that are not sufficiently sequence-similar to any network protein. *arbitrary* deletions, on the other hand, are similar to at least one network protein, but do not participate in the solution. Similarly, we distinguish between *special* and *arbitrary* insertions (as defined in 2.3), where the special insertions are network vertices not similar to any query protein, and arbitrary insertions match at least one query protein.

We tested the following hypothesis: If a query solution requires an equal number of special insertions and apriori deletions, then the insertions in the target networks correspond to the deleted query proteins. Hence, they are actually similar, but their sequence similarity is higher (poorer) than the cutoff e-value. However, out of 99 matches with an identical number of such insertions and deletions, we found no insertion-deletion pairs that had meaningful similarity values, even when raising the cutoff to 0.1. Thus, we could not deduce a correspondence between the special insertions and apriori deletions in a solution based on sequence similarity. Different measures of similarity between proteins, such as structural similarity, could still potentially prove a connection between the indel pairs.

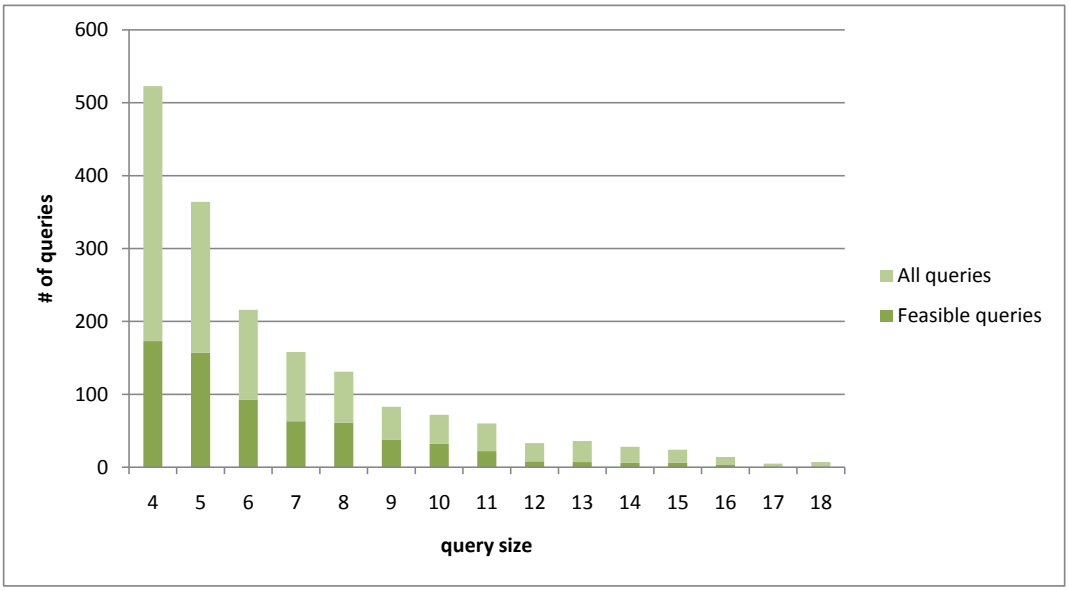


Figure A.1: Size distribution of queries

Number of insertions	Connected component size
0	32
1	209
2	564
3	1304

Table A.1: Average size of the connected component a solution was found in

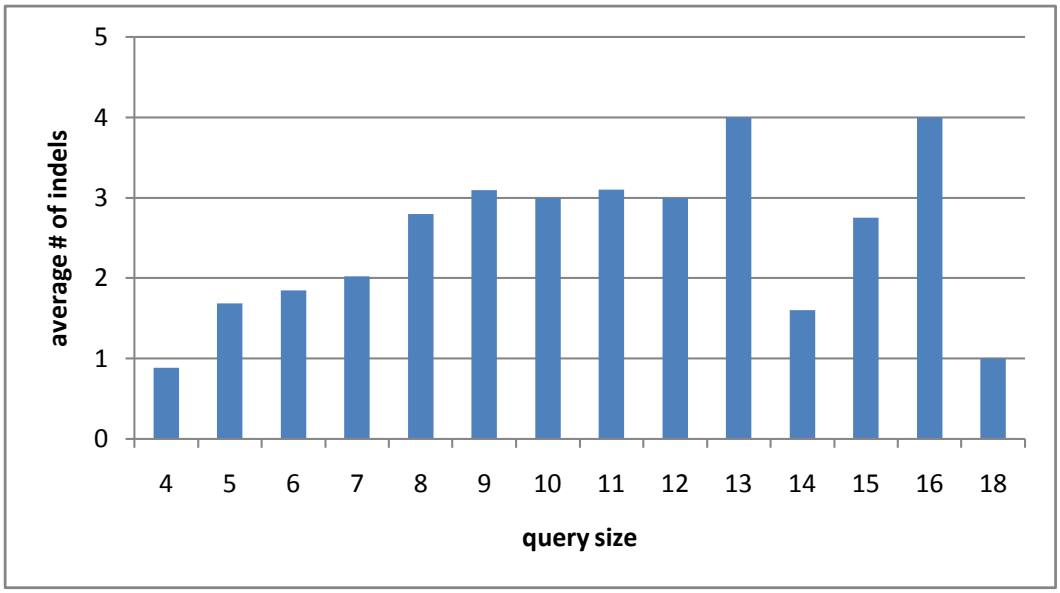


Figure A.2: Indel distribution. The figure shows the average number of indels in the solutions as a function of the query size.

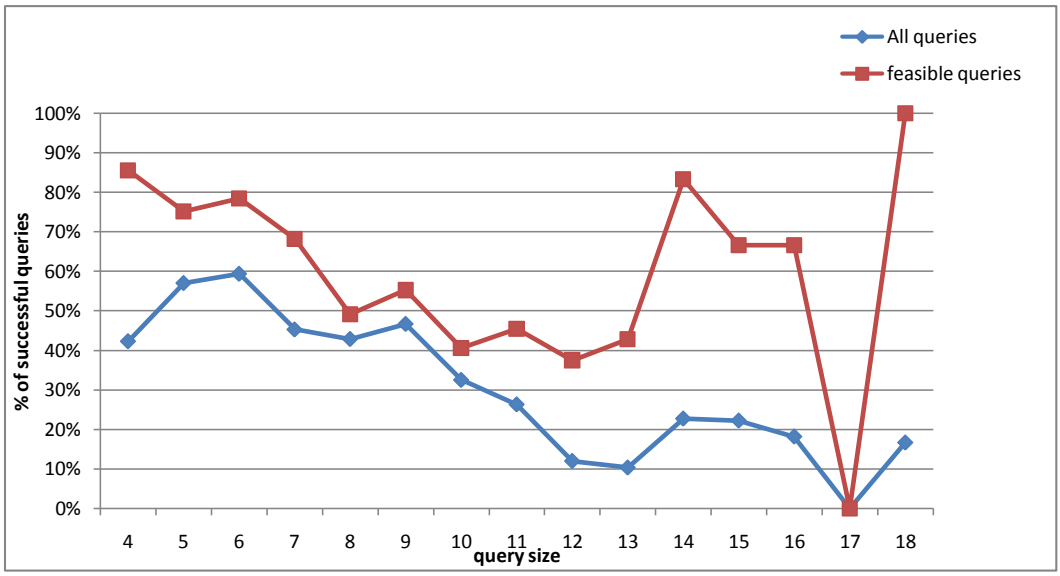


Figure A.3: The figure shows, for each query size, the ratio of queries for which a solution was found to the total number of queries and to the total number of feasible queries.