FINDING CONSERVED PATTERNS IN BIOLOGICAL SEQUENCES, NETWORKS AND GENOMES

A Dissertation

by

QINGWU YANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2007

Major Subject: Computer Science

FINDING CONSERVED PATTERNS IN BIOLOGICAL SEQUENCES, NETWORKS AND GENOMES

A Dissertation

by

QINGWU YANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2007

Major Subject: Computer Science

ABSTRACT

Finding Conserved Patterns in Biological Sequences, Networks and Genomes.
(December 2007)

Qingwu Yang, B.S., Peking University
Chair of Advisory Committee: Dr. Sing-Hoi Sze

Biological patterns are widely used for identifying biologically interesting regions within macromolecules, classifying biological objects, predicting functions and studying evolution. Good pattern finding algorithms will help biologists to formulate and validate hypotheses in an attempt to obtain important insights into the complex mechanisms of living things.

In this dissertation, we aim to improve and develop algorithms for five biological pattern finding problems. For the multiple sequence alignment problem, we propose an alternative formulation in which a final alignment is obtained by preserving pairwise alignments specified by edges of a given tree. In contrast with traditional NP-hard formulations, our preserving alignment formulation can be solved in polynomial time without using a heuristic, while having very good accuracy.

For the path matching problem, we take advantage of the linearity of the query path to reduce the problem to finding a longest weighted path in a directed acyclic graph. We can find k paths with top scores in a network from the query path in polynomial time. As many biological pathways are not linear, our graph matching approach allows a non-linear graph query to be given. Our graph matching formulation overcomes the common weakness of previous approaches that there is no guarantee on the quality of the results.

For the gene cluster finding problem, we investigate a formulation based on constraining the overall size of a cluster and develop statistical significance estimates that allow direct comparisons of clusters of different sizes. We explore both a restricted version which requires that orthologous genes are strictly ordered within each cluster, and the unrestricted problem that allows paralogous genes within a genome and clusters that may not appear in every genome. We solve the first problem in polynomial time and develop practical exact algorithms for the second one.

In the gene cluster querying problem, based on a querying strategy, we propose an efficient approach for investigating clustering of related genes across multiple genomes for a given gene cluster. By analyzing gene clustering in 400 bacterial genomes, we show that our algorithm is efficient enough to study gene clusters across hundreds of genomes.

To my wife Fengling Wang, and my daughters Connie and Judy

ACKNOWLEDGMENTS

My deepest gratitude is to my advisor, Dr. Sing-Hoi Sze, for his tremendous support and guidance in my study and research. His continuous support helped me overcome many difficult situations and complete this dissertation.

I am grateful to the members of my committee, Dr. Nancy Amato, Dr. Jianer Chen and Dr. Daniel J. Ebbole, for their input and interest in my research.

I would like to thank our collaborators for valuable discussion and input. The multiple sequence alignment project is joint work with Yue Lu. The gene cluster finding project is joint work with Dr. Michael Thon, Fenghui Zhang, and Gangman Yi.

I met a lot of new friends after I came to Texas A&M University. I greatly value their friendship and what I have learned from them.

Finally, this dissertation would not have been possible without the support, understanding, encouragement and sacrifice of my wife, Fengling Wang, who has been taking care of our family and who has shared so many difficulties, uncertainties and challenges for completing this dissertation. I would like to thank my parents and parents-in-law for their constant unselfish support and encouragement throughout all these years. My thanks also go to my daughters, Connie and Judy, who were both born while I was working on this dissertation and who have brought unparalleled and numerous joy to me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

A. Motivation

Biological patterns are widely used for identifying biologically interesting regions within macromolecules, classifying biological objects, predicting functions and studying evolution. Biological patterns can be, among others, a short chain of similar amino acid or nucleotide residues appearing in several protein or nucleic acid sequences, a metabolic pathway existing in different organisms, a subgraph appearing in several biological networks, or a cluster of homologous genes occurring in multiple genomes. Occurrences of a pattern in the biological pattern finding problem are often not identical, but have high similarity that can be defined by various scoring schemes. Pattern finding is one of the most fundamental problems in computational biology and algorithms for finding different patterns are among the most frequently used computational tools. Good pattern finding algorithms will help biologists to formulate and validate hypotheses in an attempt to obtain important insights into the complex mechanisms of living things.

The pattern finding problem is usually addressed in two directions: (1) given several related biological samples, identify possible common patterns; (2) given a pattern instance, search for all possible occurrences within biological objects. In this study, the first direction is referred to as the pattern identification problem, while the latter as the pattern matching problem. Both directions have been extensively studied. The pattern identification problems include multiple sequence alignment (Sze, *et al.*, 2006), motif finding (Sze, *et al.*, 2004; Wang, *et al.*, 2006), gene cluster

This dissertation follows the style of *Journal of Computational Biology*.

finding (Heber and Stoye, 2001a), and so on. Some of the pattern matching problems are string pattern matching (Jonassen, *et al.*, 1995), path matching (Kelly, *et al.*, 2003), and graph matching (Yang and Sze, 2007).

We aim to improve existing algorithms and develop new algorithms for biological pattern finding problems. Specifically, we will address the following biological pattern finding problems: (i) The multiple sequence alignment problem; (ii) The path matching problem; (iii) The graph matching problem; (iv) The gene cluster finding problem; (v) The gene cluster querying problem. Problems (i) and (iv) are pattern identification problems, while (ii), (iii) and (v) are pattern matching problems. Our basic strategy is to transform biological pattern finding problems into computer science problems. The problems will then be solved by computational approaches.

B. Outline and Our Contribution

This chapter provides an overview of the dissertation, including the motivation of this research and the overall idea of our approach and contributions. The rest of the dissertation covers the above-mentioned five biological pattern finding problems. Since these problems are different in nature, they are stated specifically in four chapters and each chapter is self-contained. Path matching and graph matching are studied in Chapter III. Multiple sequence alignment, gene cluster finding and gene cluster querying are studied in Chapters II, IV and V, respectively. In each chapter, one or two pattern finding problems are introduced, pertinent previous research in the field is surveyed, our motivation and research design is discussed, then results and findings from applying our new algorithms to real-world biological problems are reported. Specifically, the rest of the dissertation is organized as follows.

In Chapter II, we propose an alternative formulation of multiple alignment based on the idea of finding a multiple alignment of k sequences which preserves $k-1$ pairwise alignments as specified by edges of a given tree. By using pairwise alignments that incorporate consistency information from other sequences, we show that it is possible to obtain very good accuracy with the preserving alignment formulation. We show that a reasonable objective function to use is to find the shortest preserving alignment, and, by a reduction to a graph-theoretic problem, that the problem of finding the shortest preserving multiple alignment can be solved in polynomial time. We demonstrate the success of this approach on three sets of benchmark multiple alignments by using consistency-based pairwise alignments from the first stage of two of the best performing progressive alignment algorithms TCoffee and ProbCons, and replace the second heuristic progressive step of these algorithms by the exact preserving alignment step. We apply this strategy to TCoffee and show that our approach outperforms TCoffee on two of the three test sets. The most important advantage of the preserving alignment formulation is that we are certain that we can solve the problem in polynomial time without using a heuristic.

In Chapter III, we develop algorithms for the following path matching and graph matching problems: (i) given a query path p and a graph G, find a path p' that is most similar to p in G; (ii) given a query graph G_0 and a graph G, find a graph ${G_0}'$ that is most similar to G_0 in G. In these problems, p and G_0 represent a given substructure of interest to a biologist, and G represents a large network in which the biologist desires to find a related substructure. These algorithms allow the study of common substructures in biological networks in order to understand how these networks evolve both within and between organisms. We reduce the path matching problem to finding a longest weighted path in a directed acyclic graph and show that the problem of finding top k suboptimal paths can be solved in polynomial time. This is in contrast with most previous approaches that used exponential time algorithms to find simple paths which are practical only when the paths are short. We reduce the graph matching problem to finding highest scoring subgraphs in a graph and give an exact algorithm to solve the problem when the query graph G_0 is of moderate size. This eliminates the need for less accurate heuristic or randomized algorithms. We show that our algorithms are able to extract biologically meaningful pathways from protein interaction networks in the DIP database and metabolic networks in the KEGG database.

In Chapter IV, we develop algorithms for solving the gene cluster finding problem. The most popular approaches require only that the distance between adjacent genes in a cluster to be small, and thus can return extremely large clusters even when the distance allowed is small. We investigate a different formulation based on constraining the overall size of a cluster and develop statistical significance estimates that allow direct comparisons of clusters of different sizes. Since the problem is *NP*-hard, we first consider a restricted version which requires that orthologous genes are strictly ordered within each cluster and show that it can be solved in polynomial time. We then develop practical exact algorithms for the unrestricted problem that allow paralogous genes within a genome and clusters that may not appear in every genome. In order to represent multiple functions of a gene due to different domains, we consider a general model in which a gene is allowed to appear in more than one orthologous group. We show that our algorithms can discover biologically relevant gene clusters in a set of four bacterial genomes *Bacillus subtilis*, *Streptococcus pyogenes*, *Streptococcus pneumoniae* and *Clostridium acetobutylicum*, and a set of four yeast genomes *Saccharomyces cerevisiae*, *Saccharomyces paradoxus*, *Saccharomyces mikatae* and *Saccharomyces bayanus*.

The gene cluster querying problem is addressed in Chapter V. We propose an efficient approach for investigating clustering of related genes across multiple genomes given a known gene cluster. Although existing algorithms are available that can identify gene clusters across two or more genomes, very few algorithms are efficient enough to study gene clusters across hundreds of genomes. We observe that a querying strategy can be used to analyze gene clusters across a large number of genomes and develop an efficient algorithm to identify all related clusters on a genome from a given query cluster. We use this algorithm to study gene clustering in 400 bacterial genomes by starting from a well-characterized list of operons in *Escherichia coli* K12 and perform comparative analysis of operon occurrences, gene orientations and rearrangements both within and across clusters.

Finally, in Chapter VI, we give our conclusion and discuss some future directions.

CHAPTER II

A POLYNOMIAL TIME SOLVABLE FORMULATION OF MULTIPLE SEQUENCE ALIGNMENT[1]

A. Introduction

The goal of the multiple alignment problem is to bring similar regions from different sequences as closely together as possible, with applications in diverse types of biosequence analysis (Taylor 1987; Carillo and Lipman 1988; Thompson et al. 1994; Gotoh, 1996; Morgenstern et al. 1996; Stoye 1998; Notredame et al. 2000; Lee et al. 2002; Edgar 2004; Van Walle et al. 2004; Do et al. 2005). Since traditional multiple alignment formulations are NP-hard (Just 2001), it is unreasonable to expect that one will ever be able to find an efficient approach that always returns an optimal alignment. The best known exact algorithm employs dynamic programming techniques with time complexity $O(n^k)$ (Carillo and Lipman 1988), where n is the maximum sequence length and k is the number of sequences, and thus is useful only when k is small. Stoye (1998) proposed a divide-and-conquer heuristic to limit the search space by subdividing the input sequences into shorter segments, but it is not efficient enough for large-scale applications.

The inherent difficulty of the multiple alignment problem leads naturally to the development of heuristic approaches. Among the most successful of these are progressive approaches, which combine the given sequences in some order to obtain a multiple alignment (Feng and Doolittle 1987; Thompson et al. 1994; Notredame et al.

[1]Part of the data reported in this chapter is reprinted with permission from "A Polynomial Time Solvable Formulation of Multiple Sequence Alignment" by Sze, S.-H., Lu, Y. and Yang, Q., 2006, *Journal of Computational Biology*, 13, 309-319. Copyright 2006 by Mary Ann Liebert Inc.

2000; Edgar, 2004; Do et al. 2005). These heuristics are often coupled with iterative refinement of the initial multiple alignment to obtain improved performance (Gotoh 1996; Edgar 2004; Do et al. 2005). Alternatively, other non-progressive approaches assemble a final multiple alignment from short alignments of local similarities (Morgenstern et al. 1996; Van Walle et al. 2004). Although in most cases, a scoring scheme and an accompanying objective function can be defined for these heuristics, it is often unclear how close the final alignment is to the optimal. Some efforts have been spent to develop approximation algorithms for multiple alignment with guaranteed performance bound, but the theoretical bound is usually too weak to reflect the actual performance (Gusfield 1993).

We propose an alternative formulation of multiple alignment that is solvable in polynomial time. Such a formulation is very important as it makes it possible to know what the alignment means and also ensures that the optimal solution can be found. Instead of employing objective functions that are very difficult to optimize, the formulation is based on the idea of finding a multiple alignment of k sequences which preserves $k-1$ pairwise alignments as specified by edges of a given tree. In particular, one can use the optimum spanning tree that includes the best $k-1$ pairwise alignments covering all the sequences. Although it is well known that such a preserving alignment always exists (Feng and Doolittle 1987; Gusfield 1993; Pevzner 2000), it did not become a mainstream method for multiple alignment since it seems that a lot of information is lost from ignoring pairwise similarities outside the tree.

The preserving alignment approach can be seen as a restricted version of a broader class of consistency-based approaches, which aim to maximize the consistency between the resulting multiple alignment and a given set of pairwise alignments on aligned residue pairs (Kececioglu 1993; Notredame et al. 1998). A distinct advantage of these consistency-based approaches is that once the pairwise alignments are fixed, the multiple alignment follows logically without the need to define a multiple alignment score (Notredame et al. 1998). Since the pairwise alignments are not restricted to be on a tree in this more general formulation, they can be conflicting and thus the objective function is likely to be more accurate, but it is also likely to be intractable to optimize. In another direction, by incorporating consistency information from other sequences when computing individual pairwise alignments, these consistency-based pairwise alignments have been successfully used in the pairwise stage of progressive approaches to give some of the best performing multiple alignment approaches to date (Notredame et al. 2000; Edgar 2004; Do et al. 2005). By employing these consistency-based pairwise alignments, we will show that it is possible to obtain very good accuracy even with the more restrictive preserving alignment formulation. Other studies that made use of the notion of consistency include Gotoh (1990) and Vingron and Argo (1991).

A complication with the preserving alignment formulation is that there may be many multiple alignments which preserve the given $k-1$ pairwise alignments and previous studies did not suggest how to choose among them. Ideally, one would like to maximize the similarity level over all columns. However, many of the objective functions that attempt to exploit these similarities are likely to be intractable to optimize. One natural way that allows us to develop a tractable approach is to find the shortest preserving multiple alignment with the smallest number of columns, corresponding to adding as few gaps as possible while preserving the pairwise alignments along the tree edges. Without being able to control the similarity level in each individual column, this formulation discourages gaps (similar to other traditional formulations) while making sure that the resulting multiple alignment resembles the given pairwise alignments. One important advantage of the preserving alignment formulation is that once the tree and the pairwise alignments are fixed, no additional parameters or a scoring scheme for multiple alignment are needed. This makes it possible to use any tree or pairwise alignments directly from other approaches, including ones that have made use of structural information. Similar ideas of utilizing structural pairwise alignments have been proposed by a few studies (O'Sullivan et al. 2004; Van Walle et al. 2004).

We will show, by a reduction to a graph-theoretic problem, that the problem of finding the shortest preserving multiple alignment can be solved in polynomial time, and, by using consistency-based pairwise alignments, that the accuracy of this exact approach is comparable to the best heuristic multiple alignment approaches on three sets of benchmark multiple alignments from Thompson et al. (1999), Edgar (2004) and Van Walle et al. (2004), thus justifying the use of the proposed polynomial time formulation over other NP-hard formulations. In particular, we reduce the multiple alignment problem to finding a topological partial ordering in a directed acyclic graph where each vertex represents a partially aligned column and unordered vertices are allowed to share the same label. The label assigned to each vertex from the ordering specifies its position in the multiple alignment and the ordering itself represents a preserving multiple alignment.

B. Problem Formulation

Let $S = \{s_1, \ldots, s_k\}$ be a given set of sequences. Assume that we are given a tree T with k vertices where each vertex of T is labeled by a distinct sequence and each edge (i, j) of T represents a pair of sequences s_i and s_j, and we are also given a pairwise alignment P_{ij} between sequences s_i and s_j for each edge (i, j) of T. A multiple alignment M of S is said to preserve all the $k - 1$ pairwise alignments on T if for each edge (i, j) of T, the induced pairwise alignment on sequences s_i and

s_j is the same as P_{ij} when the columns containing only gap characters are removed. Since the tree T specifies pairwise alignments that can be simultaneously preserved, it is obvious that such a preserving multiple alignment M always exists (Feng and Doolittle 1987; Gusfield 1993; Pevzner 2000). Likewise, a multiple alignment M of S is said to preserve all matches and mismatches (or just the matches) in the $k-1$ pairwise alignments on T if for each edge (i,j) of T, each column containing a match or a mismatch (or just a match) has to stay in the same column in M. We formulate the multiple alignment problem as follows: given a tree T and a pairwise alignment P_{ij} for each edge (i,j) of T, find a preserving multiple alignment M with the smallest number of columns.

The simplest way to obtain pairwise alignments is by applying standard techniques, including global (Needleman and Wunsch 1970) and local (Smith and Waterman 1981) approaches, or a combination of these approaches. However, we found that it is often better to use other kinds of pairwise alignments such as those that have incorporated consistency information from other sequences. These consistency-based pairwise alignments can be obtained from the pairwise stage of a few progressive approaches (Notredame et al. 2000; Do et al. 2005). From these pairwise alignments, one reasonable tree T to use is an optimum spanning tree on the complete graph C_k where each vertex is labeled by a distinct sequence and each edge (i,j) is labeled by the score of P_{ij}, which can either be the pairwise alignment score of P_{ij} or other scores given by the approach generating the pairwise alignments. To compute the optimum spanning tree from C_k, Prim's algorithm can be used which has time complexity $O(k^2 \log k)$ (Cormen et al. 2001). Alternatively, one can use a star tree with a central vertex and $k-1$ leaves (Gusfield 1993; Pevzner 2000). Although computational results show that using a star tree works well when all the given sequences are closely related, it does not give very good performance when none of the given sequences can act as the center, such as when there is more than one cluster of closely related sequences. In contrast, by using a general tree, it is possible to utilize the best non-conflicting pairwise alignments.

Note that the tree used here represents each sequence as a vertex, which is different from the phylogenetic tree (Morrison 1996) or the guide tree (Thompson et al. 1999) used in traditional progressive approaches in which sequences are represented only at the leaves. As a result, only alignments between sequences are needed in the preserving alignment formulation, which is similar to the comparisons made between sequences during the greedy extension step of the multiple alignment algorithm of Taylor (1987) and Taylor (1988), while progressive approaches need to consider alignments between alignments (Altschul and Lipman 1989). However, it is possible to make use of a phylogenetic tree when one is given. Instead of using pairwise alignment scores, we can use the distances between sequences s_i and s_j on the phylogenetic tree to compute an optimum spanning tree. In this case, we only need to compute $k-1$ pairwise alignments along the spanning tree.

C. Exact Algorithm

For simplicity of analysis, assume that each of the input sequences is of the same length n. Given a tree T and a pairwise alignment P_{ij} for each edge (i, j) of T, we give an algorithm to solve the shortest preserving multiple alignment problem in linear time by two successive graph reductions. Gusfield (1993) gave an algorithm to solve the problem in the important special case when the given tree is a star. We first consider preserving only the matches and mismatches instead of entire pairwise alignments. We will show that this strategy also preserves the indel columns under normal situations. Let s_{ij} be the letter at the jth position of sequence s_i. The first reduction constructs an undirected graph $G = (V, E)$ as follows (see Figure 1(a)–(d)):

> Let $V = \{v_{ij}\}$, where v_{ij} represents the jth position of sequence s_i, and $E = \{\{v_{ip}, v_{jq}\} \mid (i, j) \in T$ and (s_{ip}, s_{jq}) is a match or a mismatch column in $P_{ij}\}$.

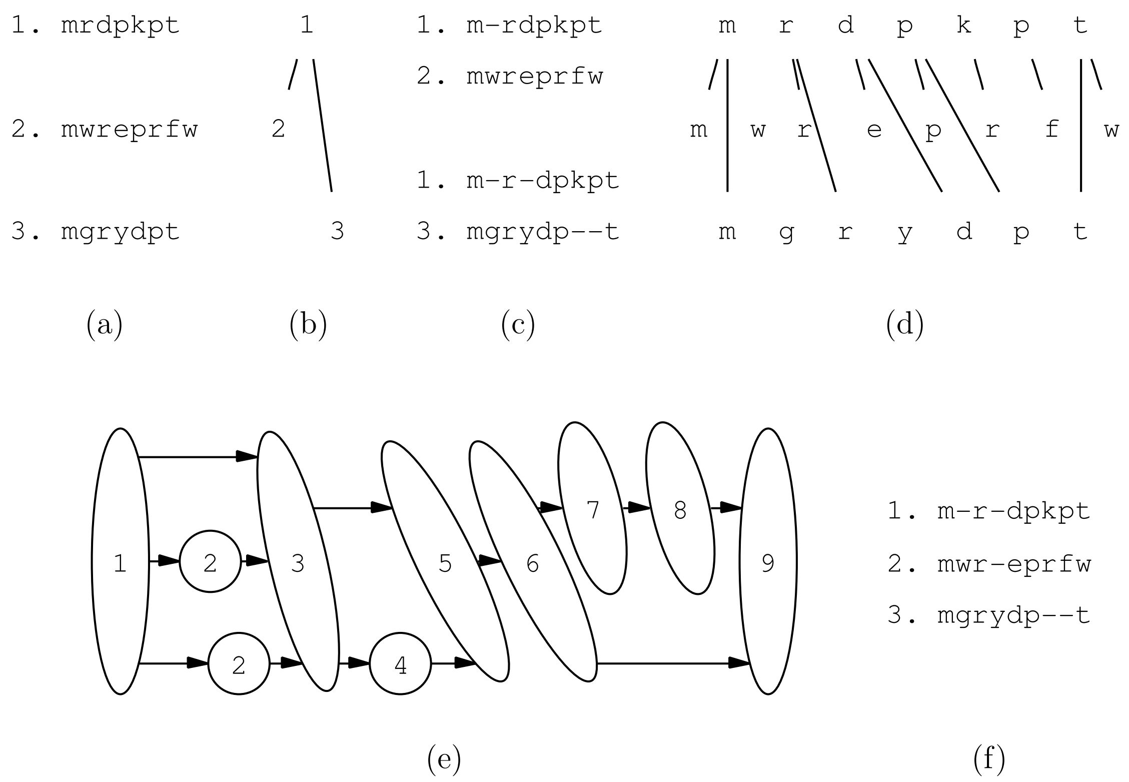
Intuitively, E contains all the match and mismatch columns within the pairwise alignments along the edges of T, and thus specifies exactly all the preservation constraints. The observation below follows directly from T being a tree and P_{ij} being pairwise alignments.

Proposition 1. *Each connected component C in G is a tree and contains at most one vertex from each sequence s_i.*

To obtain a preserving multiple alignment, letters within each connected component in G must be put into the same column. On the other hand, we are free to put two different connected components in the same column as long as they do not contain vertices from the same sequence. Also, when assigning components to different columns to obtain a multiple alignment, the order of the letters within each sequence must be maintained. To represent these constraints precisely, the second reduction constructs a directed graph $G' = (V', E')$ from G as follows (see Figure 1(e) for a transitively reduced version of G'):

> Let V' be the set of all connected components C in G and let $s(C)$ be the set of sequences that the vertices in C reside. Connect a component C_1 to another component C_2 by a directed edge in E' if $s(C_1) \cap s(C_2) \neq \phi$ and for every sequence $s_i \in s(C_1) \cap s(C_2)$ shared by C_1 and C_2, the vertex v_{ip} in C_1 appears before the vertex v_{iq} in C_2 (i.e., $p < q$). (1)

Note that two connected components that contain vertices from the same sequence are strictly ordered and thus will be connected by an edge (in one of the



```
1. mrdpkpt

2. mwreprfw

3. mgrydpt
```

```
1. m-rdpkpt
2. mwreprfw

1. m-r-dpkpt
3. mgrydp--t
```

```
1. m-r-dpkpt
2. mwr-eprfw
3. mgrydp--t
```

Fig. 1. Illustration of the exact algorithm. (a) Set of sequences S. (b) Tree T. (c) Pairwise alignments P_{12} and P_{13}. (d) Undirected graph G constructed from S, T, P_{12} and P_{13}. (e) Directed graph G' (transitively reduced) constructed from G by taking connected components in G as vertices. Labels of vertices in G' are assigned by the topological partial ordering algorithm. (f) Shortest preserving alignment by interpreting labels as columns.

directions), since if there are two vertices v_{ip} and v_{jq} in C_1 and two vertices v_{ir} and v_{js} in C_2 with $p < r$, then we must have $q < s$. The reasoning is as follows. Let $v_{ip} = u_1, \ldots, u_t = v_{jq}$ be the unique path between v_{ip} and v_{jq} in C_1 and $v_{ir} = w_1, \ldots, w_t = v_{js}$ be the unique path between v_{ir} and v_{js} in C_2. For $1 \leq l < t$, since u_l and w_l are on the same sequence, the adjacent pairs (u_l, u_{l+1}) and (w_l, w_{l+1}) represent two match or mismatch columns within one single pairwise alignment. The fact that $p < r$ and $q \geq s$ contradicts with these being columns in the pairwise alignments. A more elaborate argument gives the following.

Proposition 2. *G' is a directed acyclic graph.*

Proof Let $C_1, \ldots, C_t$ be a cycle in G'. Then there exist sequences $s_{i_1}, \ldots, s_{i_t}$ such that $v_{i_1p_1}$ is in C_1 and $v_{i_1q_1}$ is in C_2 with $p_1 < q_1$, $v_{i_2p_2}$ is in C_2 and $v_{i_2q_2}$ is in C_3 with $p_2 < q_2$, and so on, until finally, $v_{i_tp_t}$ is in C_t and $v_{i_tq_t}$ is in C_1 with $p_t < q_t$. Between these vertices, there is a unique path $v_{i_1q_1}, \ldots, v_{i_2p_2}$ on C_2, $v_{i_2q_2}, \ldots, v_{i_3p_3}$ on C_3, and so on, until finally, $v_{i_tq_t}, \ldots, v_{i_1p_1}$ on C_1. Thus the cycle can be represented by the path $v_{i_1p_1} \xrightarrow{j} v_{i_1q_1} \xrightarrow{w} v_{i_2p_2} \xrightarrow{j} v_{i_2q_2} \xrightarrow{w} \cdots \xrightarrow{w} v_{i_tp_t} \xrightarrow{j} v_{i_tq_t} \xrightarrow{w} v_{i_1p_1}$, where $\xrightarrow{j}$ denotes a jump to a later connected component on the same sequence, and $\xrightarrow{w}$ denotes walking along the tree edges in T within a connected component which visits each sequence at most once. On this path, whenever a sequence s is visited again, the position of visit on s must increase, since a jump increases the position of visit on the same sequence from p_l to q_l, at least one jump has to occur before s is visited again, and whenever a walk moves from s to another sequence t along a tree edge in T, the only way to return to s is through t along the same edge in T. Walking along T this way with no increase in the position of visit on s when s is revisited contradicts the given pairwise alignments. In particular, this is true for sequence s_{i_1}, a contradiction to the above cycle which keeps the position of visit on s_{i_1} at p_1. ∎

Since the primary purpose of G' is to specify ordering constraints, a transitively reduced version of G' suffices (see Figure 1(e)). Instead of performing the transitive reduction step, such a graph G' can be obtained directly from G by requiring further that there exists a sequence s_i such that $p + 1 = q$ in (1). This reduces the number of edges in G' substantially and we will be using this definition in what follows. The above results suggest that a multiple alignment can be obtained by finding a topological partial ordering in G'.

Definition 3. *A topological partial ordering of a directed acyclic graph $G' = (V', E')$ is an assignment of an integer label $l(v)$ to each vertex $v \in V'$ such that for each edge $(u, v) \in E'$, $l(u) < l(v)$.*

Since it is possible that two vertices are assigned the same label, the result is not necessarily a total order (a total order corresponds to a topological sorting (Knuth 1997)). Without loss of generality, assume that the labels are consecutive integers from 1 to m. From a given graph G', there are many ways to realize such an ordering. For each fixed ordering, a multiple alignment can be obtained by putting each letter within a connected component C in G (C is a vertex in G') in column $l(C)$ and filling other unassigned spaces by gap characters (see Figure 1(f)). The set of all possible ways to do this represents the solution space of all preserving alignments.

Proposition 4. *Each topological partial ordering of G' specifies a multiple alignment preserving all matches and mismatches in the $k - 1$ pairwise alignments on T.*

By making additional assumptions, it is possible to ensure that the resulting multiple alignment preserves the given pairwise alignments entirely, which includes the indel columns in addition to the match and the mismatch columns, without requiring an algorithm change. The observation below follows directly from the constraints imposed on the placement of gap characters between two match or mismatch columns that must be preserved and are separated only by indel columns.

Proposition 5. *If for each pairwise alignment P_{ij} on T, there do not exist two adjacent indel columns (without match or mismatch columns in between) such that the gap character is on sequence s_i in column l and on sequence s_j in column $l-1$ or $l+1$, then each topological partial ordering G' specifies a multiple alignment preserving the $k-1$ pairwise alignments on T.*

It is very rare to be given pairwise alignments that violate the condition given in Proposition 5, and thus in most cases the resulting multiple alignment also preserves the given pairwise alignments entirely. In the other direction, one can relax the constraints by requiring only the match columns (or the columns with a positive score with respect to a given substitution matrix) to be preserved, which can be achieved by allowing only edges representing these columns to be added to G. Computational results show that simply finding the shortest preserving multiple alignment in this case does not give very good performance since the flexibility in the placements of the resulting smaller connected components in G becomes excessive. The following observation completes the reduction.

Proposition 6. *A topological partial ordering of G' that uses the smallest number of labels specifies a shortest preserving multiple alignment.*

Since edges in G correspond to match or mismatch columns in the given $k-1$ pairwise alignments along the given tree T and each pairwise alignment is of length $O(n)$, there are $O(kn)$ vertices and edges in G. Thus there are $O(kn)$ connected components in G which make the vertices in G'. These connected components can be obtained in $O(kn)$ time by a depth-first search on G. In the simplest case, each connected component in G is of size one (which represents one position on a single sequence), and edges in G' are constructed between components that represent adjacent positions within the same sequence, resulting in a total of $O(kn)$ edges in G'. The graph G' in any other case with larger connected components can be obtained from this simplest case by merging the corresponding vertices and collapsing each resulting multi-edge into a single edge, and thus the number of edges in G' is $O(kn)$ in all cases. To find a topological partial ordering that uses the smallest number of labels in G', an algorithm very similar to the standard topological sorting algorithm (Knuth 1997) can be used: initially, all vertices are unmarked. Repeatedly find an unmarked vertex v with all its incoming vertices marked. Set the label of v to be one plus the maximum label among all its incoming vertices and mark v (see Figure 1(f)). If a count is kept in each vertex representing the number of remaining unmarked incoming vertices, the algorithm can be implemented in time linear in the number of edges in G'. Thus, with appropriate data structures, the overall time complexity of the entire procedure is $O(kn)$, which is linear in the input size. If it is not important to obtain a totally ordered multiple alignment, it is possible to return the graph G' directly as a partially ordered multiple alignment, which is similar in concept but different in structure to the notion of partial order multiple alignment proposed in Lee et al. (2002). In this case, there is no need to define any objective function.

D. Performance

We evaluate the accuracy of the preserving alignment algorithm (PSAlign) on three sets of benchmark multiple alignments: BAliBASE from Thompson et al. (1999), PREFAB from Edgar (2004), and SABmark from Van Walle et al. (2004). We compare our performance to TCoffee (Notredame et al. 2000) and ProbCons (Do et al. 2005), which are currently considered to be among the best multiple alignment algorithms. To make fair comparisons, we compare our performance (PSAlign[TCoffee]) to TCoffee when pairwise alignments from TCoffee are used and we compare our performance (PSAlign[ProbCons]) to ProbCons when pairwise alignments from ProbCons are used. For TCoffee, we use pairwise alignments computed from the extended library that have incorporated consistency information from other sequences (Notredame et al. 2000). For ProbCons, we use pairwise alignments computed after consistency transformation (Do et al. 2005). All these pairwise alignments incorporate consistency information from other sequences and it has been shown that this significantly improves the quality of the pairwise alignments with respect to the overall consistency. We then compute an optimum spanning tree from these consistency-based pairwise alignments using pairwise scores given by the two algorithms (normalized by the length of each pairwise alignment) and apply the preserving alignment step. Since our main goal is to show that the heuristic progressive step of these approaches can be replaced by the exact preserving alignment step, we compare to a variant of ProbCons with no iterative refinements. We also compare our performance (without performing further refinements) to ProbCons with iterative refinements.

Following Thompson et al. (1999), two score measures are used to evaluate the accuracy of each algorithm in finding the core blocks in BAliBASE (which are annotated regions that can be reliably aligned): the sum-of-pairs score (SPS) measures how well an algorithm can align pairs of residues within the same column correctly, while the column score (CS) measures how well an algorithm can align entire columns correctly. For PREFAB, we follow Edgar (2004) and use the Q score, which has the same meaning as SPS used in BAliBASE. For SABmark, we define the Q score for each test case as the average Q score over all pairs of reference sequences. For both PREFAB and SABmark, the reference alignments are based on pairwise comparisons and thus the CS score is not applicable. For each test set, we compare average accuracy over meaningful subsets and use the Wilcoxon matched-pairs signed-ranks test (Wilcoxon 1947) to check if there are significant performance differences with $p = 0.05$ as cutoff. Note that in the preserving alignment computation, the shortest solution is not necessarily unique and we simply report an arbitrary one.

Tables I, II and III show performance comparisons of the various algorithms on BAliBASE, PREFAB and SABmark respectively. A general trend was that ProbCons tends to perform better than TCoffee, whether PSAlign is used or not. When we consider the ability of PSAlign[TCoffee] to replace TCoffee and the ability of PSAlign[ProbCons] to replace ProbCons, PSAlign was a much better replacement when used in conjunction with TCoffee than with ProbCons and the only case where PSAlign[TCoffee] is worse than TCoffee was in reference 4 of BAliBASE. Also, PSAlign was a better replacement when used on SABmark than on BAliBASE, but PSAlign[ProbCons] was not an adequate replacement when used on PREFAB.

On BAliBASE, when compared to TCoffee and ProbCons($ir = 0$) that do not perform iterative refinements, PSAlign had better accuracy on references 1V1 and 5, but this was offset by worse accuracy on references 3 and 4. Although there were no noticeable differences in the overall accuracy, the Wilcoxon matched-pairs test revealed that ProbCons($ir = 0$) performed better than PSAlign[ProbCons] in the SPS score with $p = 0.02$. The differences in all the other cases on the overall accuracy were insignificant with TCoffee or in the CS score. We did not apply the Wilcoxon test to any of the subsets of BAliBASE due to their small sizes. When compared to ProbCons($ir = 100$) that performs iterative refinements, PSAlign[ProbCons] still maintained better accuracy on reference 5, but it no longer had better accuracy on

Table I. Average SPS and CS scores (in %) on BAliBASE 2.01. Reference 1 is further subdivided into three subsets: V1 ($<$ 25% identity), V2 (20%–40% identity) and V3 ($>$ 35% identity). Comparisons are made between TCoffee 1.37 and PSAlign utilizing TCoffee pairwise alignments (PSAlign[TCoffee]) and between ProbCons 1.10 and PSAlign utilizing ProbCons pairwise alignments (PSAlign[ProbCons]). No iterative refinements are performed for ProbCons($ir = 0$). Default parameters are used otherwise, with ProbCons($ir = 100$) performing 100 rounds of iterative refinements.

SPS	1V1	1V2	1V3	1 (Overall)	2	3	4	5	Overall
TCoffee	64.1	95.2	98.5	87.6	93.5	78.6	93.9	96.0	89.1
PSAlign[TCoffee]	67.3	95.1	98.4	88.4	93.6	78.5	89.1	97.3	89.2
ProbCons($ir = 0$)	69.1	96.6	98.5	89.5	94.2	84.0	90.8	97.4	90.6
ProbCons($ir = 100$)	74.5	96.8	98.5	91.1	94.2	84.0	93.7	97.4	91.8
PSAlign[ProbCons]	71.5	96.6	98.4	90.1	94.0	80.9	90.1	98.0	90.7
CS	1V1	1V2	1V3	1 (Overall)	2	3	4	5	Overall
TCoffee	41.5	90.9	96.8	79.1	58.4	50.9	80.4	90.3	74.4
PSAlign[TCoffee]	47.3	90.6	96.5	80.5	58.3	54.8	68.4	90.0	74.5
ProbCons($ir = 0$)	49.6	93.4	96.9	82.3	61.6	63.5	72.1	89.3	77.1
ProbCons($ir = 100$)	59.6	94.0	96.9	85.3	61.6	63.5	81.1	89.3	79.6
PSAlign[ProbCons]	55.8	93.5	96.6	84.0	61.7	52.2	69.7	93.6	77.2

Table II. Average Q scores (in %) on PREFAB 4.0. Each subset includes all structure pairs with identity within the specified range.

	0%–20%	20%–40%	40%–70%	70%–100%	Overall
TCoffee	49.2	82.9	93.8	95.1	66.5
PSAlign[TCoffee]	51.0	84.6	95.2	97.9	68.3
ProbCons($ir = 0$)	55.6	87.2	95.4	97.3	71.7
ProbCons($ir = 100$)	55.6	87.2	95.4	97.3	71.7
PSAlign[ProbCons]	52.1	85.2	93.0	94.2	68.8

Table III. Average Q scores (in %) on SABmark 1.65. The FP variant of the two subsets includes false positive sequences.

	Superfamily	Superfamily-FP	Twilight	Twilight-FP	Overall
TCoffee	52.9	45.6	23.7	17.0	39.8
PSAlign[TCoffee]	54.8	54.1	25.8	25.4	45.0
ProbCons($ir = 0$)	56.7	52.5	28.6	23.0	45.2
ProbCons($ir = 100$)	57.1	53.2	29.3	24.0	45.8
PSAlign[ProbCons]	56.1	53.6	28.1	25.1	45.6

reference 1V1. The Wilcoxon test revealed that ProbCons($ir = 100$) had significantly better overall accuracy than PSAlign[ProbCons] in both the SPS and the CS scores with $p < 0.001$, which was mainly due to large accuracy improvements on references 1V1 and 4 from iterative refinements (no noticeable improvements were observed on the other references).

On PREFAB, PSAlign[TCoffee] had better accuracy than TCoffee in all five categories and these improvements were significant with $p < 0.001$ for the subsets with 0% to 20% identity, with 20% to 40% identity, with 70% to 100% identity, and for the entire set. On the other hand, ProbCons($ir = 0$) performed significantly better than PSAlign[ProbCons] in all five categories with $p < 0.001$, while no noticeable improvements were observed with ProbCons($ir = 100$) over ProbCons($ir = 0$).

On SABmark, PSAlign[TCoffee] showed highly significant improvements over TCoffee with $p < 0.001$ in all five categories. However, on the Superfamily and Twilight subsets, ProbCons($ir = 0$) performed significantly better than PSAlign[ProbCons] (with $p < 0.001$ for the Superfamily subset and $p = 0.03$ for the Twilight subset). The situation was reversed on the Superfamily-FP and Twilight-FP subsets when PSAlign[ProbCons] performed significantly better than ProbCons($ir = 0$) (with $p = 0.01$ for the Superfamily-FP subset and $p < 0.001$ for the Twilight-FP subset), while the difference between ProbCons($ir = 0$) and PSAlign[ProbCons] on the overall accuracy was insignificant. Although ProbCons($ir = 100$) further increased the performance differences from PSAlign[ProbCons] on the Superfamily and Twilight subsets to $p < 0.001$ in both cases, PSAlign[ProbCons] was able to maintain significantly better accuracy than ProbCons($ir = 100$) on the Twilight-FP subset with $p < 0.001$ while having insignificant difference in accuracy on the Superfamily-FP subset. Although ProbCons($ir = 100$) performed significantly better than PSAlign[ProbCons] on the overall accuracy with $p = 0.003$, one important advantage of PSAlign is that it had a much smaller accuracy decrease when either the Superfamily or Twilight subset is replaced by its FP variant with false positive sequences.

Overall, PSAlign[TCoffee] performed at least as well as TCoffee on BAliBASE and was much better than TCoffee on PREFAB and SABmark. When compared to ProbCons($ir = 0$), PSAlign[ProbCons] achieved similar or better accuracy on BAliBASE and SABmark, but did not perform as well on PREFAB. When compared to ProbCons($ir = 100$), PSAlign[ProbCons] achieved similar or better accuracy on many sub-categories even without further refinements, but had worse overall accuracy. These results did not lead to a conclusive statement that shows that using PSAlign has a definite advantage or disadvantage, and thus it is hard to predict whether there will be significant accuracy differences if we replace the heuristic progressive step of some given multiple alignment algorithm by the exact preserving alignment step. Nevertheless, the most important advantage of the preserving alignment formulation is that we are certain that we can solve the problem in polynomial time without using a heuristic. Since the time complexity is dominated by the computations of the pairwise alignments, the preserving alignment step does not add much to the running time. What we actually observe was at most a two times slowdown when PSAlign was used to replace TCoffee or ProbCons, due to the need to compute all consistency-based pairwise alignments to obtain the optimum spanning tree.

E. Discussion

The proposed multiple alignment formulation divides the multiple alignment problem into two subproblems. The first subproblem requires the computation of pairwise alignments and a tree, which can be defined systematically so that optimal solutions can be computed in polynomial time. For example, one can use a technique similar to that used in Notredame et al. (2000) to compute consistency-based pairwise alignments based on comparisons of three sequences so that each of them can be computed in $O(kn^2)$ time. It is especially important to obtain high quality pairwise alignments in this stage, since we found that good accuracy cannot be obtained when simple non-consistency-based pairwise alignments are used. This was confirmed by a much bigger decrease in accuracy and a much worse performance of PSAlign[ProbCons] in most cases as compared to ProbCons($ir = 0$) when the consistency transformation in ProbCons was disabled (Table IV). The second stage computes a shortest preserving multiple alignment from this information, which can be used to replace the progressive step of any approach in which unified pairwise alignments are available before the progressive step, or as a second step to construct multiple alignments for algorithms that only produce a set of pairwise alignments from the given sequences (Heger et al. 2003; Van Walle et al. 2004), without requiring additional parameters.

The graph-theoretic technique employed allows further extensions to consider more general models of pairwise similarity. In its full generality, all we need from each pairwise comparison is an ordered list of non-intersecting connections (representing matches or mismatches) that reflect significant pairwise similarities. With these inputs, the preserving alignment approach naturally returns either local or incomplete multiple alignments. To further improve accuracy, it is possible to consider formulations other than finding the shortest solution, although many of these objective functions may be intractable to optimize. One possible strategy is to employ various heuristics to find a preserving alignment from G' that tries to assign related connected components in G to the same column as much as possible. Other directions include improving the quality of the pairwise alignments and devising strategies to perform iterative refinements (Gotoh 1996; Edgar 2004; Do et al. 2005) after the

Table IV. Performance of ProbCons($ir = 0$) and PSAlign[ProbCons] when consistency transformation in ProbCons is disabled. Both algorithms use ordinary pairwise alignments instead of consistency-based pairwise alignments. (a) Average SPS and CS scores (in %) on BAliBASE 2.01. (b) Average Q scores (in %) on PREFAB 4.0. (c) Average Q scores (in %) on SABmark 1.65.

(a) BAliBASE

SPS	1V1	1V2	1V3	1 (Overall)	2	3	4	5	Overall
ProbCons($ir = 0$)	63.2	95.5	98.3	87.4	93.3	82.4	88.6	94.7	88.7
PSAlign[ProbCons]	65.2	93.3	97.1	86.7	90.8	76.6	85.2	93.0	86.9

CS	1V1	1V2	1V3	1 (Overall)	2	3	4	5	Overall
ProbCons($ir = 0$)	42.7	91.1	96.5	79.4	56.4	50.9	64.7	81.2	72.1
PSAlign[ProbCons]	44.6	86.7	94.1	77.4	47.7	44.5	57.1	80.6	68.3

(b) PREFAB

	0%–20%	20%–40%	40%–70%	70%–100%	Overall
ProbCons($ir = 0$)	52.6	85.1	95.5	97.6	69.4
PSAlign[ProbCons]	41.5	80.0	92.2	96.0	61.4

(c) SABmark

	Superfamily	Superfamily-FP	Twilight	Twilight-FP	Overall
ProbCons($ir = 0$)	54.8	50.5	26.3	20.3	43.1
PSAlign[ProbCons]	51.5	50.8	24.3	23.0	42.2

preserving multiple alignment is obtained.

CHAPTER III

PATH MATCHING AND GRAPH MATCHING IN BIOLOGICAL NETWORKS[1]

A. Introduction

As recent advances in experimental design produce a large amount of data to describe biological interactions at a genome scale, these data are increasingly being deposited into biological databases. These interaction networks include, among others, protein interaction networks (Kelley et al. 2003; Koyutürk et al. 2006; Scott et al. 2006), metabolic networks (Dandekar et al. 1999; Ogata et al. 2000; Tohsato et al. 2000; Koyutürk et al. 2004), gene regulatory networks (Akutsu et al. 1998), and signal transduction networks (Steffen et al. 2002). Since there is strong evidence that conserved interaction pathways exist across organisms (Kelley et al. 2003), efficient algorithms to analyze common pathways in these networks will contribute a lot to the understanding of these networks. The most common representation of these networks is a graph in which vertices represent biological entities and edges represent interactions between them. With this representation, one can look for the presence of special substructures in these graphs to answer various questions. By studying paths in these graphs, one can investigate the properties of pathways and their relationships, and by finding similar subgraphs in these graphs, one can search for network motifs and study common substructures embedded in these networks.

We study the following path matching and graph matching problems: (i) given a query path p and a graph G, find a path p' that is most similar to p in G; (ii)

[1]Part of the data reported in this chapter is reprinted with permission from "Path matching and graph matching in biological networks" by Yang, Q. and Sze, S.-H., 2007, *Journal of Computational Biology*, 14, 56-67. Copyright 2007 by Mary Ann Liebert Inc.

given a query graph G_0 and a graph G, find a graph $G_0{}'$ that is most similar to G_0 in G. In these problems, p and G_0 specify a given biological substructure, and G represents a large network in which each vertex is labeled by a gene, a protein, an enzyme or a chemical compound. While path matching allows the study of individual linear paths or chains, graph matching allows the study of entire non-linear pathways or functional modules. Efficient algorithms for these problems will allow biologists to study evolutionary mechanisms such as pathway conservation, duplication and specialization. To achieve this, one important strategy is to start from a given pathway of interest and find related pathways within the same organism (Kelley et al. 2003). This is especially useful for studying organisms such as yeast that are believed to have undergone whole-genome duplications (Wolfe and Shields 1997). Another strategy is to start from a given pathway of interest in a well characterized organism such as yeast, and find related pathways in less studied organisms such as *C. elegans* or *D. melanogaster* (Kelley et al. 2003). By comparing the similarities and differences in the returned pathways, it is possible to evaluate various hypotheses concerning evolution. It is also possible to predict unknown functions or interactions from the results (Sharan et al. 2005).

Instead of solving the path matching problem directly, most previous approaches addressed the more general problem of finding common paths from two or more biological networks without a given query. To solve the problem of finding similar paths in two graphs, Kelley et al. (2003) constructed a combined graph from the two given graphs so that each vertex in the combined graph represents a pair of related vertices, one from each of the two given graphs, and each pathway alignment is represented as a simple path in the combined graph. They proposed a randomized algorithm to find high scoring simple paths by imposing acyclic edge orientations. Their formulation includes the path matching problem as a special case when one of the two graphs is a linear path. However, due to the exponential time complexity, their technique is practical only when the paths are short. Scott et al. (2006) proposed an improved randomized algorithm based on the color coding technique (Alon et al. 1995) and was able to find simple paths of length around 10, but its exponential time complexity makes it impractical to identify even slightly longer paths. While such general approaches allow the simultaneous study of many conserved pathways on a genome scale, it is both more sensitive and more efficient to make use of the given query to guide the search when it is available.

We observe that the difficulty stems from the presence of cycles in the given graphs. To avoid substantial repetitions of vertices in a path, the problem of finding similar paths in two graphs was reduced to the NP-hard problem of finding high scoring simple paths of a given length l in a combined graph (Kelley et al. 2003). It was necessary to require that paths be simple in the combined graph since highly repeating non-simple paths that do not have biological meaning often concentrate around cycles of large weight and have much better scores than other simple paths. Note that the simple path requirement was imposed only on the combined graph in Kelley et al. (2003) and it is still possible to have repeated vertices in some of the returned paths within each of the two given graphs, but this does not create problems since such repeated occurrences should be infrequent.

Since one of the given graphs is a path in the path matching problem, we take advantage of its linearity to reduce the problem to finding a longest weighted path in a directed acyclic graph, which is a much easier problem since it also corresponds to a shortest path problem. This is possible since for each vertex v in the query path p, we can group together all vertices in G that are related to v and choose at most one vertex from each group to form a path. Since it is well known that the problem of finding k shortest paths in a directed graph can be solved in polynomial time (Fox

1975; Eppstein 1998), the problem of finding top k suboptimal paths in G given a query path p can be solved in polynomial time.

Although it is possible that a vertex in G may appear multiple times in different groups, these repeated occurrences should not be extensive unless the vertices in p are all very similar. Such limited repeated occurrences are biologically useful in identifying multiple roles of a vertex. There are many known examples of such multi-functional genes or proteins that have multiple roles within a pathway. Figure 2(a) shows part of the citric acid cycle in which two different enzymes each participate in two consecutive steps of the pathway. Figure 2(b) shows a more complicated example in which two distinct regions of the same protein participate in two non-consecutive steps of the pathway. In general, such multi-domain enzymes that have more than one functionality within a pathway have been shown to be quite common (Teichmann et al. 2001). By allowing repeating occurrences in the returned paths, our algorithm allows automatic discovery of such complex mechanisms. This is in contrast with previous algorithms in Kelley et al. (2003) and Scott et al. (2006) in which these repeating occurrences were not considered explicitly due to the restriction of finding simple paths in the combined graph. We will show that this flexibility does not lead to excessive repeats in the returned paths and such repeats occur naturally in the returned paths when multi-functionality is possible.

Although the path matching problem adequately models linear interaction chains, many biological pathways are not linear and may consist of multiple interacting components. We model this complexity in the graph matching problem by allowing a non-linear graph query to be given. However, the problem becomes much more difficult to solve and no efficient exact algorithms have been developed before. To solve the problem of finding similar subgraphs in two graphs, Koyutürk et al. (2006) generalized the notion of alignment to include non-linear structures and proposed

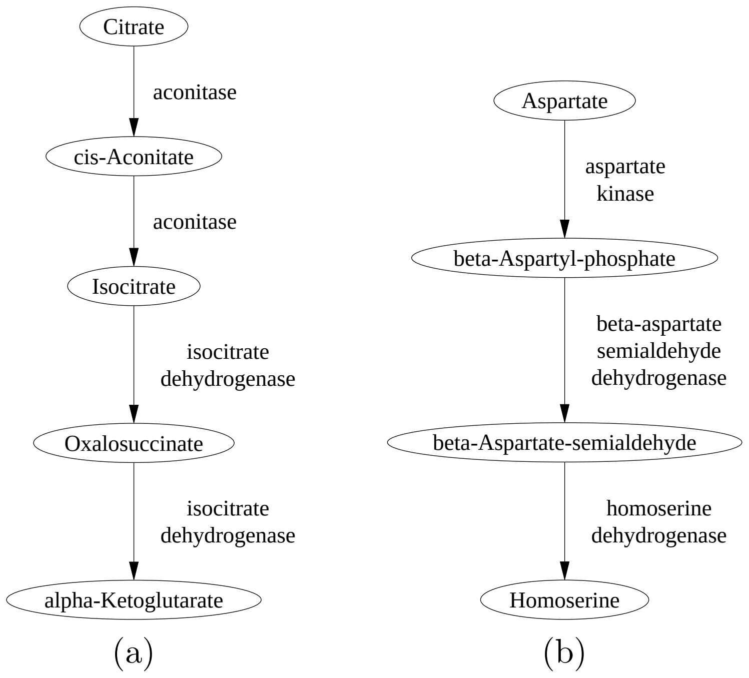


Fig. 2. Examples of pathways in which the same enzyme has multiple roles. (a) Part of the citric acid cycle in which each of the two enzymes aconitase and isocitrate dehydrogenase has two different functions. (b) Pathway converting aspartate to homoserine in plants and bacteria. Aspartate kinase and homoserine dehydrogenase are the same protein within the pathway but are named differently to reflect that the functions are carried out by two distinct regions of the same polypeptide chain (Paris et al. 2003).

a greedy heuristic to find high scoring network alignments. Sharan et al. (2005) developed heuristic algorithms for finding network alignments in multiple species. Scott et al. (2006) used the color coding technique (Alon et al. 1995) to develop randomized algorithms for finding specialized common substructures such as trees and series-parallel graphs. One common weakness of these approaches is that there is no absolute guarantee on the quality of the results since some of the highest scoring structures may be missed. In our graph matching formulation, since the query graph G_0 usually corresponds to a small functional module, we can similarly take advantage of groupings of related vertices in G for each vertex in G_0 to develop exact algorithms for the graph matching problem.

Software programs implementing these techniques, i.e. PathMatch and GraphMatch, are freely available online at http://faculty.cs.tamu.edu/shsze/pathmatch and http://faculty.cs.tamu.edu/shsze/graphmatch, respectively.

B. Polynomial Time Algorithm for Path Matching

We formulate the path matching problem as follows: given a query path $p = p_1p_2\cdots p_n$, a graph $G = (V, E)$, and for each p_i, a set of correspondences $V_i = \{v_{i1}, v_{i2}, \ldots, v_{i,t_i}\}$ defining vertices in V that can be associated with p_i, find a path p' in G that aligns best to p (see Figure 3). We define a path alignment between p and p' in the usual way by treating each aligned column that contains an associated vertex pair as a match, each aligned column that contains a non-associated vertex pair as a mismatch, and other columns as indels. To score a path alignment, we assume that a similarity-based score s_{ij} is given for each associating vertex pair (p_i, v_{ij}) that will serve as the match score. We also assume that mismatches and indels are penalized by the same amount Δ. To avoid an exceedingly large number of mismatches or indels between matches, we follow the approach in Kelley et al. (2003) to impose an upper limit m on the number of mismatches or indels allowed between two matches in a path alignment.

To solve the path matching problem, we construct a directed graph $G' = (V', E')$ as follows (see Figure 3): let $V' = \bigcup_{i=1}^{n} V_i \cup \{s, t\}$, where V_i denotes the correspondence list for p_i with each $v_{ij} \in V_i$ treated as a distinct vertex with weight s_{ij}, and s and t are two additional vertices with zero weight denoting the source and sink of a path in G' respectively. Intuitively, each v_{ij} represents a match with p_i in a path alignment and vertices in V_i form level i of G' with s at level 0 and t at level $n+1$. Let m be the maximum number of mismatches or indels allowed between two matches in a path alignment. We add directed edges to G' as follows:

> For each pair of vertices v_{ij} and $v_{i+d,l}$ satisfying $0 < d \leq m+1$, compute the length of the shortest path d' from v_{ij} to $v_{i+d,l}$ in G, and construct a directed edge from v_{ij} to $v_{i+d,l}$ if $d' \leq m+1$. Add a directed edge from s to each v_{ij} and from each v_{ij} to t.

Each edge $(v_{ij}, v_{i+d,l})$ represents that it is always possible to find a path from v_{ij} to $v_{i+d,l}$ in G so that there are a total of at most m mismatches or indels between the matches (p_i, v_{ij}) and $(p_{i+d}, v_{i+d,l})$. To impose mismatch and indel penalties, if $\Delta \leq 0$ represents both the mismatch and indel penalty, set the weight of each edge $(v_{ij}, v_{i+d,l})$ to $(max(d, d') - 1)\Delta$, the weight of each edge (s, v_{ij}) to $(i-1)\Delta$, and the weight of each edge (v_{ij}, t) to $(n-i)\Delta$. The above construction reduces the path matching problem to finding a path p' from s to t in G' with the maximum sum of vertex and edge weights.

In the above construction, while the given graph G can either be undirected or directed, G' is always a directed graph. Although some vertices in G' can represent

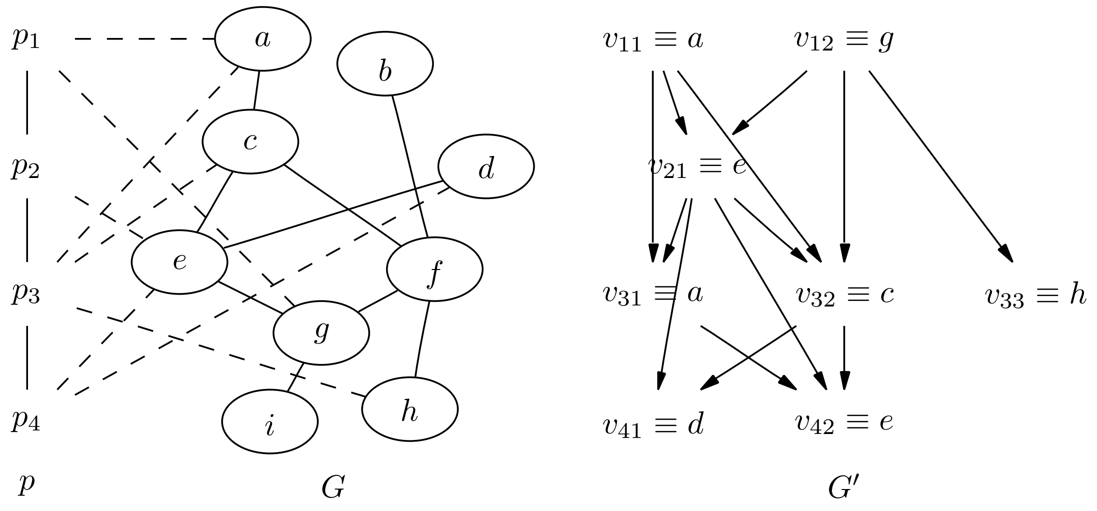


Fig. 3. Illustration of the construction of G' from a given path p and an undirected graph G in the PathMatch algorithm. Dashed lines indicate vertex correspondences and the equivalence sign within each vertex of G' indicates the vertex it represents in G. In this example, at most $m = 1$ mismatches or indels are allowed between two matches in a path alignment. For clarity, the source vertex s and the sink vertex t are omitted and vertex and edge weights are not shown in G'. There is an edge from s to each vertex v_{ij} in G' and from each vertex v_{ij} in G' to t (not shown).

the same vertex in G, they are considered to be distinct in G'. It is important to retain these repeated occurrences since it is not necessarily true that the most similar genes or proteins participate in related pathways (Kelley et al. 2003; Sharan et al. 2005). In the edge construction procedure of G', there is no need to consider connecting v_{ij} with $v_{i+d,l}$ when $d > m+1$, since in order to satisfy the upper limit m, the corresponding path must contain at least one match and this path has already been included in G' through another vertex that represents the match. Likewise, we can assume that each edge in G' (with $d \leq m+1$) represents only mismatches and indels since a path with matches has already been included elsewhere. To make the problem easier to solve, an important restriction has been made to treat mismatches and indels in exactly the same way, including their use in the definition of m and the requirement that they must be assigned the same penalty score. It is unclear whether the problem can still be solved in polynomial time if we use more general scoring schemes, such as when different penalties are used for different mismatches, when the mismatch penalty is different from the indel penalty, or when an affine gap penalty model (Altschul and Erickson 1986) is used, since the computation of shortest paths may no longer be adequate.

The above procedure requires the computation of shortest paths between pairs of vertices, which can be accomplished in two ways: either a single-pair shortest path algorithm can be applied for each pair, which takes $O(|E|+|V|\log|V|)$ time for each pair if the Dijkstra's algorithm is used with Fibonacci heaps, or an all-pairs shortest path algorithm can be used to preprocess G before path queries are applied, which takes $O(|V|^3)$ time for the entire input graph G if the Floyd-Warshall algorithm is used (Cormen et al. 2001). This latter option is especially suitable for biological data since G usually represents a known network.

When we assume that each edge in G' represents only mismatches and indels and ignore different variations of mismatches or indels that can appear in a path, we can find a set of k highest scoring paths in G' by reducing the problem to finding k shortest paths from s to t in the following modified graph with edge weights only: first negate all the vertex and edge weights in G', then move each vertex weight into all its outgoing edges by changing each $w(u, v)$ to $w(u)+w(u, v)$ and setting $w(v) = 0$ for all vertices v. Note that the modified graph may have negative edge weights.

The problem of finding k shortest paths between two vertices in a directed graph $G' = (V', E')$ (not necessarily acyclic but without negative edge weights) is well known to be solvable in polynomial time (Lawler 1972; Fox 1975). Eppstein (1998) gave two algorithms that are based on creating an implicit representation first: the basic algorithm takes $O(|E'| + |V'| \log |V'| + k \log k)$ time to create an implicit representation and is simple enough to implement, while the full algorithm computes a different implicit representation with an improved time complexity of $O(|E'| + |V'| \log |V'| + k)$ but may be difficult to implement in practice. It then takes $O(\log i)$ time to find the ith path from the implicit representation.

Since our graph is acyclic and all paths must be simple, negative edge weights do not pose any problems. We started with an implementation of the basic algorithm (Jiménez and Marzal 2003) and modified it to allow negative edge weights by replacing the Dijkstra's algorithm (Cormen et al. 2001), which was used to compute a shortest path tree during an intermediate step, by an algorithm that recursively determines the tree edge to a vertex v after the tree edges to all the incoming vertices of v are obtained. Although this reduces the time complexity of computing a shortest path tree from $O(|E'| + |V'| \log |V'|)$ to $O(|E'|)$, it does not change the time complexity of the basic algorithm. Although we did not implement the full algorithm, the above replacement of the Dijkstra's algorithm results in a reduction in time complexity of computing the implicit representation in the full algorithm to $O(|E'| + |V'| + k)$, making it optimal if we ignore the output of paths. Either algorithm can be used to solve the path matching problem in polynomial time.

C. Exact Algorithm for Graph Matching

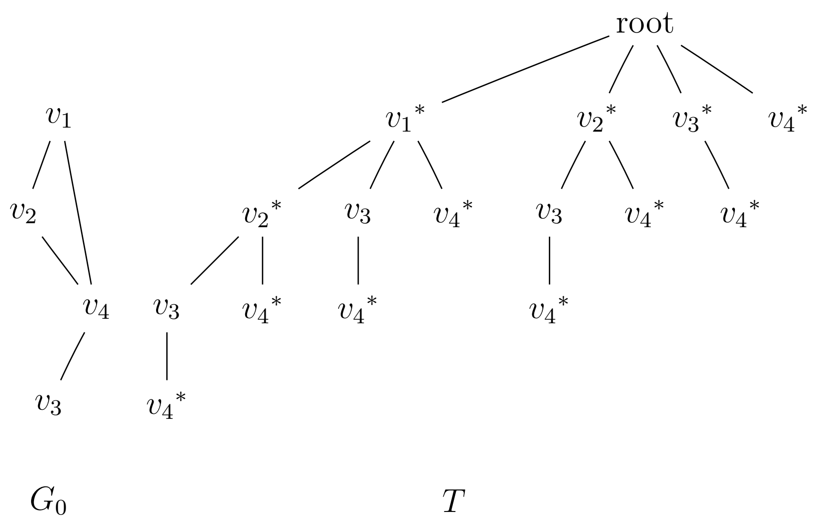
We formulate the graph matching problem as follows: given a query graph $G_0 = (V_0, E_0)$, a graph $G = (V, E)$, and for each $v_i \in V_0$, a set of correspondences $V_i = \{v_{i1}, v_{i2}, \ldots, v_{i,t_i}\}$ defining vertices in V that can be associated with v_i, find a graph ${G_0}'$ in G that aligns best to G_0. Note that the above graphs can either be undirected or directed, and we use the notation that a directed graph is connected if its underlying undirected graph is connected. We define a graph alignment between G_0 and ${G_0}'$ by assuming that G_0 has its vertex set V_0 partitioned into two subsets V_0^+ and V_0^-, where V_0^+ contains all vertices $v_i \in V_0$ that form an association (v_i, v_{ij}) with one vertex v_{ij} in ${G_0}'$, and V_0^- contains all other vertices in V_0 that do not have such an association. In order to preserve the structure of G_0 in ${G_0}'$, we require that for each pair of associations (v_i, v_{ij}) and (v_k, v_{kl}) with $(v_i, v_k) \in E_0$ (representing adjacent matches in G_0), the number of (indel) vertices between v_{ij} and v_{kl} in ${G_0}'$ is at most m, where m is a given parameter. We further require that the subgraph induced by V_0^+ in G_0 is connected so that ${G_0}'$ is also connected. To score a graph alignment, we assume that a similarity-based score s_{ij} is given for each associating vertex pair (v_i, v_{ij}) that will serve as the match score. We penalize each non-associated vertex in V_0^- by Δ_0 and each indel vertex in ${G_0}'$ by Δ_1. Note that our formulation ignores mismatches for simplicity and it is different from the notion of network alignment in Koyutürk et al. (2006) and Sharan et al. (2005).

To solve the graph matching problem, we enumerate all connected induced subgraphs of G_0, with each of them representing one way to obtain V_0^+ from V_0. This is possible since the query graph G_0 represents a small functional module and is much smaller than the full interaction network G of an organism. We grow an initially empty subgraph W one vertex at a time by adding each vertex $v \in V_0 \backslash W$ such that the subgraph $G_0[W \cup \{v\}]$ induced by $W \cup \{v\}$ is connected. To avoid repeatedly generating each connected induced subgraph of G_0 many times, we represent a connected induced subgraph with $|W|$ vertices as a path of length $|W|$ in a tree T that contains these $|W|$ vertices in sorted order from the root and mark the last vertex on this path (see Figure 4). Whenever a new subgraph is generated, we check if it has already been represented in T in $O(|V_0|)$ time. If so, the entire subtree for that search branch is pruned. Otherwise, we add the subgraph to T by creating new vertices if necessary. Since there are at most $2^{|V_0|}$ distinct induced subgraphs, the number of vertices in T is at most $O(2^{|V_0|}|V_0|)$ with each vertex occupying $O(|V_0|)$ memory for storing each potential next vertex in V_0, and the algorithm has worst case time complexity $O(2^{|V_0|}|V_0|^2)$. The algorithm is thus practical even for $|V_0|$ as large as 20, which is sufficient to represent most functional modules. In reality, G_0 is usually a sparse graph and we can replace the $2^{|V_0|}$ term above by the total number of connected induced subgraphs in G_0, which is usually much smaller than $2^{|V_0|}$.

To enumerate all solution graphs ${G_0}'$, we construct a graph $G' = (V', E')$ as follows:

> Let $V' = \bigcup_{i=1}^{|V_0|} V_i$, where V_i denotes the correspondence list for each $v_i \in V_0$. For each $v_{ij} \in V_i$ and $v_{kl} \in V_k$, add an edge (v_{ij}, v_{kl}) to E' if $(v_i, v_k) \in E_0$ and the length of the shortest path from v_{ij} to v_{kl} in G is at most $m + 1$.

Given $V_0^+ = \{v_{p_1}, \ldots, v_{p_s}\}$, a valid solution is represented as a set of vertices $\{v_{p_1,q_1}$, …, $v_{p_s,q_s}\}$, one from each correspondence list of v_{p_i}, such that whenever we have



```
Algorithm GraphMatch($W$,$W'$) {
  for each $v_i \in V_0 \backslash W$ do {
    if $G_0[W \cup \{v_i\}]$ is connected and is not in $T$ then {
      $V_0^+ \leftarrow W \cup \{v_i\}$;
      for each $v_{ij} \in V_i$ do {
        if $W' \cup \{v_{ij}\}$ is a valid solution of $V_0^+$ then {
          record $W' \cup \{v_{ij}\}$ and compute its score;
          GraphMatch($W \cup \{v_i\}$,$W' \cup \{v_{ij}\}$); } }
      add $G_0[W \cup \{v_i\}]$ to $T$; } } }
```

Fig. 4. Illustration of the exact graph matching algorithm. Tree T represents each connected induced subgraph of G_0 (which acts as a possible V_0^+) as a path from the root with the last vertex marked by a star. Algorithm GraphMatch is applied with $W = W' = \phi$ initially.

$(v_{p_j}, v_{p_k}) \in E_0$ for some (j,k), it must be true that $(v_{p_j,q_j}, v_{p_k,q_k}) \in E'$. We integrate both the enumerations of V_0^+ and the valid solutions together into a single depth-first search that grows both sets at the same time to avoid repeating the procedure for each V_0^+ separately (see Figure 4). This search is practical as long as the correspondence lists are not particularly large. Since only the top k solutions are of interest, we only keep the current top k solutions at any given time and use the score of the current kth solution as a lower bound to prune a search branch whenever this score can never be reached along that branch.

D. Application to Protein Interaction Networks

We represent a protein interaction network from DIP (Xenarios et al. 2000) by an undirected graph G in which each vertex represents a protein and each edge represents interactions between two proteins, where G can have thousands of vertices and tens of thousands of edges. For each protein in a query, we use BLAST (Altschul et al. 1990) to locate similar proteins in G and establish a correspondence if the resulting E-value is below a cutoff (Kelley et al. 2003). A similarity-based score for matches can then be defined as the minus-log E-value, which is a positive number and is almost the same as the minus-log P-value for the cutoff values we use (Karlin and Altschul 1990). Following Kelley et al. (2003), one way to easily visualize a set of solutions is to combine them to form a larger graph: a vertex or an edge appears in this graph whenever it appears in one of the solutions, with each edge replaced by its corresponding shortest path in G. Note that some suboptimal solutions may become the same after this procedure since a path alignment or a graph alignment may be represented more than once.

Figure 5 shows the result of applying PathMatch to query the protein inter-

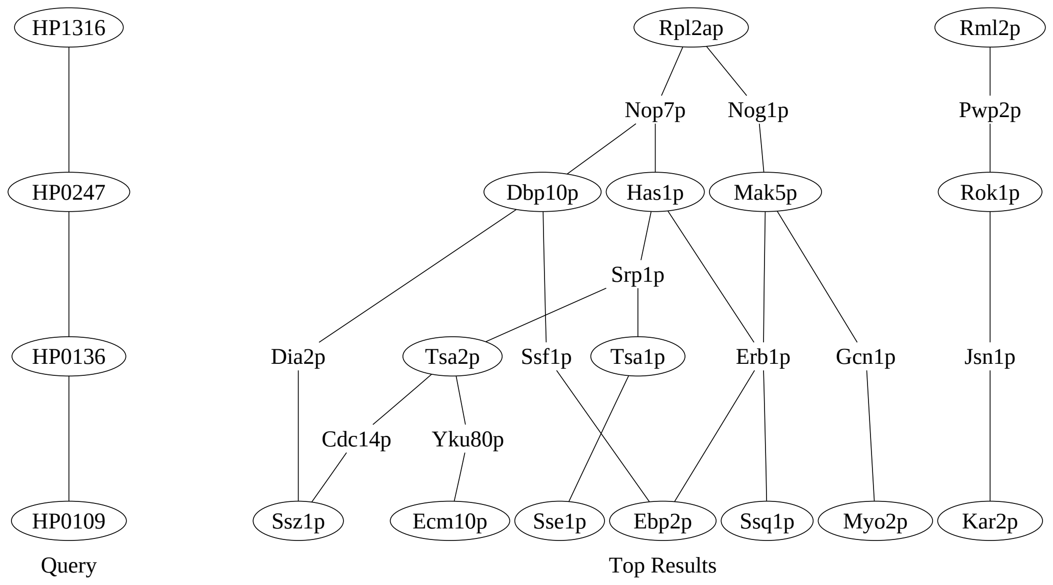


Fig. 5. Result of using PathMatch to query the protein interaction network of *S. cerevisiae* with a pathway from *H. pylori*. The left represents the query pathway from *H. pylori*, and the right shows the result from yeast, formed by combining the top ten suboptimal path alignments into a graph. The result contains a set of path alignments in which each yeast protein in an oval has correspondence to the *H. pylori* protein on the same row (matches) and other proteins not in ovals are mismatches or indels. At most $m = 1$ mismatches or indels are allowed between two matches in a path alignment. The BLAST E-value cutoff for matches is 10^{-5} and the match score is $-\ln(\text{E-value})$. The mismatch and indel penalty Δ is five times the ln E-value cutoff, which is a negative number.

action network of *S. cerevisiae* with a pathway from *H. pylori*. This *H. pylori* pathway was returned as a hit when Kelley et al. (2003) searched for conserved pathways between the two networks and they reported three pathways from yeast that align with it (Figure 2b, Kelley et al. 2003): Rpl2ap···Has1p···Tsa1p—Sse1p, Rpl2ap···Has1p···Tsa2p, and Rpl2ap···Has1p···Ssq1p, where a dotted line denotes a mismatch or an indel between two matching proteins and a solid line denotes direct interaction between two matching proteins. All these yeast pathways were also found by PathMatch. In addition, PathMatch found many others among the top suboptimal paths. A search in PIR (Barker et al. 2000) revealed that most matching proteins on the same row in Figure 5 share the same function: HP1316, Rpl2ap and Rml2p belong to the L2 family of ribosomal proteins; HP0247, Dbp10p, Has1p, Mak5p and Rok1p are ATP-dependent RNA helicases (some of them are putative); Tsa2p and Tsa1p are antioxidant proteins; while HP0109, Ssz1p, Ecm10p, Sse1p, Ssq1p and Kar2p either belong to the heat shock protein 70 family or are homologs. Thus, many of the yeast pathways found are likely to play similar roles as the given *H. pylori* pathway.

To test the ability of PathMatch to analyze related paths in the same organism, we query the protein interaction network of *S. cerevisiae* with the mitogen-activated protein (MAP) kinase cascade in the mating-pheromone response pathway of *S. cerevisiae* (Gustin et al. 1998) in order to find other MAP kinase cascades in yeast (Figure 6). This cascade is identical to the one in the filamentation/invasion pathway except that Fus3p is replaced by Kss1p (Gustin et al. 1998). The top results include known MAP kinase cascades of the cell integrity pathway (with two variants) and the high osmolarity growth pathway, in which the correspondences between the MAPKK, MAPK and MAP kinases are all correct (Gustin et al. 1998).

To demonstrate the ability of PathMatch to analyze long paths from a well

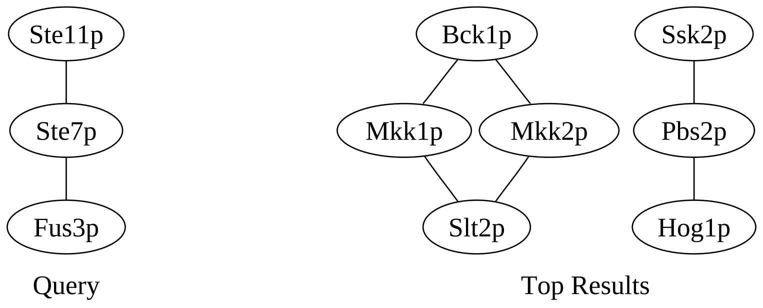


Fig. 6. Top results of using PathMatch to query the protein interaction network of *S. cerevisiae* with a pathway from *S. cerevisiae*. Vertices that appear in the query (Ste11p, Ste7p and Fus3p) and Kss1p (which is very similar to Fus3p) are removed from the interaction network before the search. Notations and settings are the same as in Figure 5, except that only the top three suboptimal path alignments are shown and the BLAST E-value cutoff for matches is 10^{-2}.

characterized organism, we extracted the longer mating-pheromone response pathway from the protein interaction network of *S. cerevisiae* (Gustin et al. 1998), which contains the MAP kinase cascade in our previous query. Figure 7 shows the result of using PathMatch to query the protein interaction networks of *C. elegans* and *D. melanogaster* with this pathway. Most notably, around the yeast MAP kinase module, the corresponding proteins all share similar functions: Ste11p, Ste7p, mig-15, Pak-PB and Mekk1-PA all belong to the serine/threonine protein kinase family; while Fus3p, mpk-1 and ERKA are all MAP kinases (Gustin et al. 1998; Stein et al. 2001; The Flybase Consortium 1996). Similar to Ste11p, Ste7p and Fus3p that work together in a MAP kinase cascade, Pak-PB, Mekk1-PA and ERKA work together in a transforming growth factor pathway (Luettich and Schmidt 2003). Only two repeating occurrences were found: mig-15, which indicates that Ste11p and Ste7p play very similar roles; and CG12045, which has hits in two distinct regions in the results of BLAST from Ste12p and Mat1ap, suggesting potential multi-functionality. We have also performed the same query on the protein interaction networks of *H.*

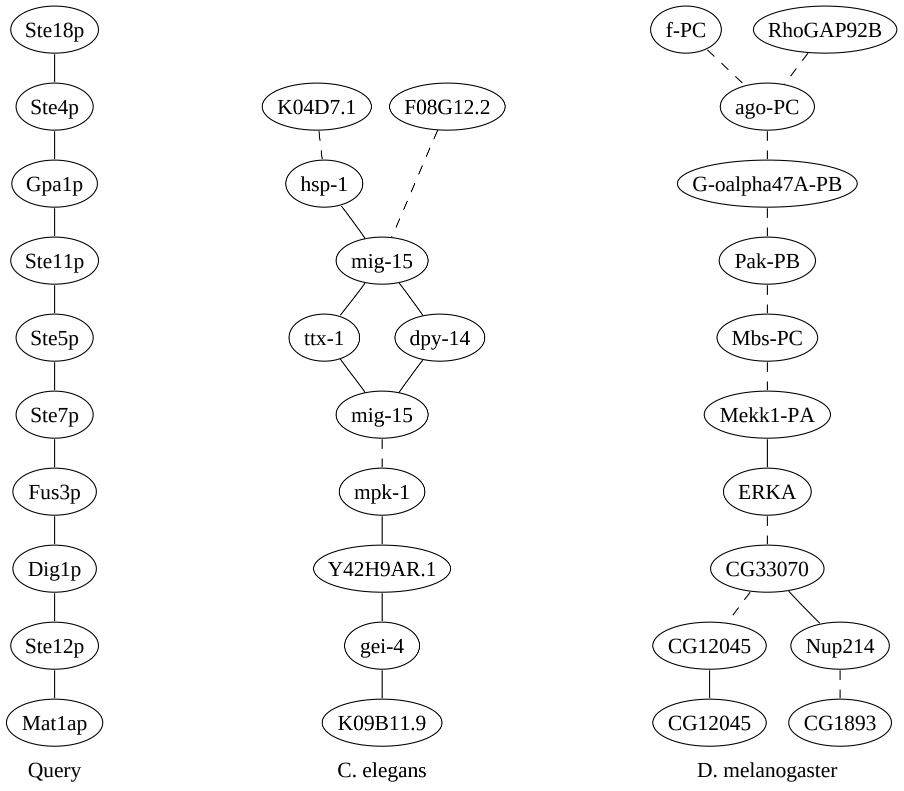


Fig. 7. Top results of using PathMatch to query the protein interaction networks of *C. elegans* and *D. melanogaster* with a long pathway from *S. cerevisiae*. Notations and settings are the same as in Figure 6. Indels are omitted and indicated by dashed edges.

pylori and *E. coli* (data not shown). The scores of the top paths (sum of minus-log E-value of matches plus the mismatch and indel penalties) were of the order 10^1, which, when compared to the previous top scores of the order 10^3 on the networks of *C. elegans* and *D. melanogaster*, are largely insignificant. This is consistent with the fact that there are no known MAP kinase pathways in bacteria (Chang and Stewart 1998).

Figure 8 shows the result of using GraphMatch to query the same networks with a related functional module from Spirin and Mirny (2003). Substantial correspondences are found around the MAP kinase module: Mkk1p, Mkk2p, sek-1 and lic-PA are all MAPK kinases; while Fus3p, Kss1p and mpk-1 and ERKA are all MAP kinases (Gustin et al. 1998; Stein et al. 2001; The Flybase Consortium 1996). The repeated occurrences of sek-1, lic-PA, mpk-1 and ERKA are due to significant similarities in the yeast proteins: Mkk1p and Mkk2p are functionally redundant, while Fus3p can be replaced by Kss1p to obtain the filamentation/invasion pathway from the almost identical mating-pheromone response pathway that has a different activation mechanism (Gustin et al. 1998). In addition, Hsc82p, daf-21 and Hsp83 are all heat shock proteins (Barker et al. 2000). There were also some hypothetical proteins with unknown function in the results. This indicates a potential use of PathMatch and GraphMatch to predict function of unknown proteins through their interaction networks.

E. Application to Metabolic Networks

We represent a metabolic network by a directed graph with two types of vertices: a compound vertex represents a chemical compound while a reaction vertex represents each reaction involving one enzyme with potentially multiple substrates and products. For each reaction vertex, a directed edge is created from each of its substrates and to each of its products. An enzyme may occur many times as distinct vertices with each of them representing one reaction and a pathway is represented by an ordered list of vertices of alternate types. A reversible reaction is modeled by two vertices, one for each direction of the reaction. Unlike many previous approaches that only use enzymes to characterize pathways (Dandekar et al. 1999; Ogata et al. 2000; Tohsato et al. 2000; Koyutürk et al. 2004), our model follows the more accurate representation in biological databases (Goto et al. 1997) and allows both enzymes and

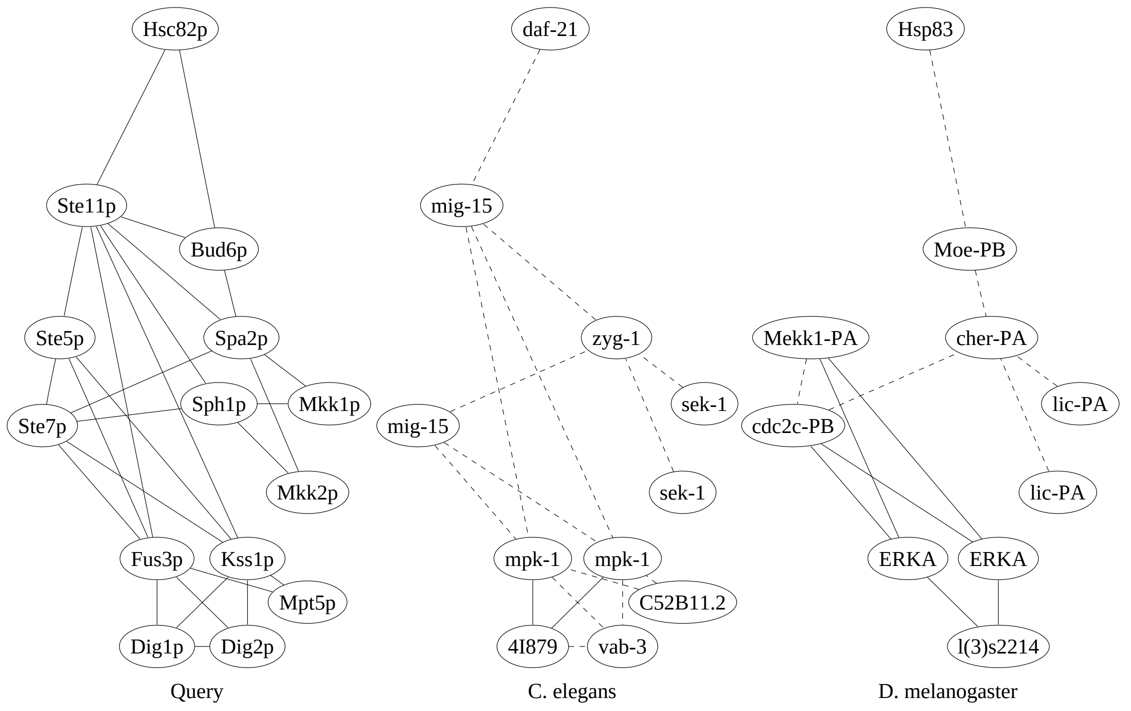


Fig. 8. Result of using GraphMatch to query the protein interaction networks of *C. elegans* and *D. melanogaster* with a *S. cerevisiae* functional module from Spirin and Mirny (2003) that is related to the query pathway in Figure 7. Only the top result is shown with associating vertices drawn in the same relative positions. Solid edges indicate no indels between the matches while dashed edges indicate one indel (at most $m = 1$ indels are allowed along each edge). The BLAST E-value cutoff for matches is 10^{-2} and the match score is $-\ln(\text{E-value})$. The non-associated vertex penalty Δ_0 and the indel penalty Δ_1 are -7.0 and -0.2 respectively ($|\Delta_1|$ is set to be small relative to $|\Delta_0|$ to allow for many indels).

substrates to be included via their interactions in reactions (which doubles the path length that needs to be handled). To establish vertex correspondences for enzymes, we follow the information content approach in Tohsato et al. (2000) utilizing similarity between EC numbers of enzymes but with a slight difference: we also impose a match cutoff and treat mismatches in the same way as indels. To establish vertex correspondences for compounds, we use the SIMCOMP package from Hattori et al. (2003) and impose a score cutoff for matches. We set the match score for enzymes to the information content value based on proximity of EC numbers and the match score for compounds to zero (thus only utilizing a similarity cutoff for compounds).

Figure 9 shows the result of using PathMatch to query a combined network of glycolysis, gluconeogenesis, the citrate cycle and the glyoxylate metabolism in *E. coli* constructed from the KEGG database (Goto et al. 1997), which contains 199 vertices (with 117 reaction vertices including 55 unique enzymes and 82 compound vertices) and 270 directed edges, with a pathway from the propanoate metabolism constructed from KEGG (Goto et al. 1997) and EcoCyc (Karp et al. 2002). The result pathways are mainly involved in transforming acetate into succinate or oxaloacetate. In particular, the top result pathway that transforms acetate into succinate had a high resemblance to the query pathway that transforms propionate into succinate, with substantial similarities between corresponding enzymes and compounds. It has been observed that these two sets of pathways might have evolved from a common origin by enzyme (and pathway) duplications (Gerike et al. 1998). Figure 10 shows the result of using GraphMatch to query the same network with a tree-like network containing part of the α-aminoadipic pathway in *T.thermophilus* (Kobashi et al. 1999). The result lied in a region that partially overlaps with the previous result in Figure 9 and again had substantial similarities between corresponding enzymes and compounds. This is consistent with the hypothesis from Nishida et al. (1999) that

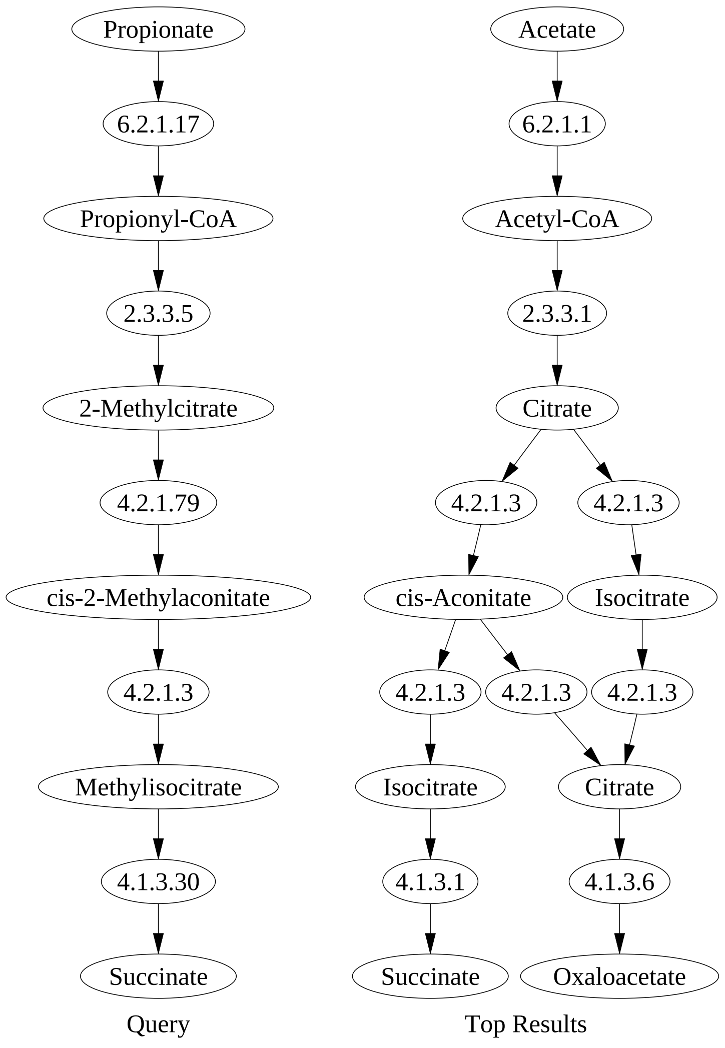


Fig. 9. Top results of using PathMatch to query a combined network of glycolysis, gluconeogenesis, the citrate cycle and the glyoxylate metabolism in *E. coli* with a pathway from the propanoate metabolism. Notations are the same as in Figures 7 and 8. Each reaction vertex is labeled by the EC number of the enzyme involved (the same enzyme does not have to come from the same reaction), while each compound vertex is labeled by the name of the compound. At most $m = 2$ mismatches or indels are allowed between two matches. The information content cutoff for enzymes is set so that a match of two enzymes is guaranteed when the first two parts of their EC numbers are the same. The score cutoff for compound similarity is 0.5. Only the information content value for enzymes contributes to the match score. The mismatch and indel penalty Δ is set to an information content value of -1.0.

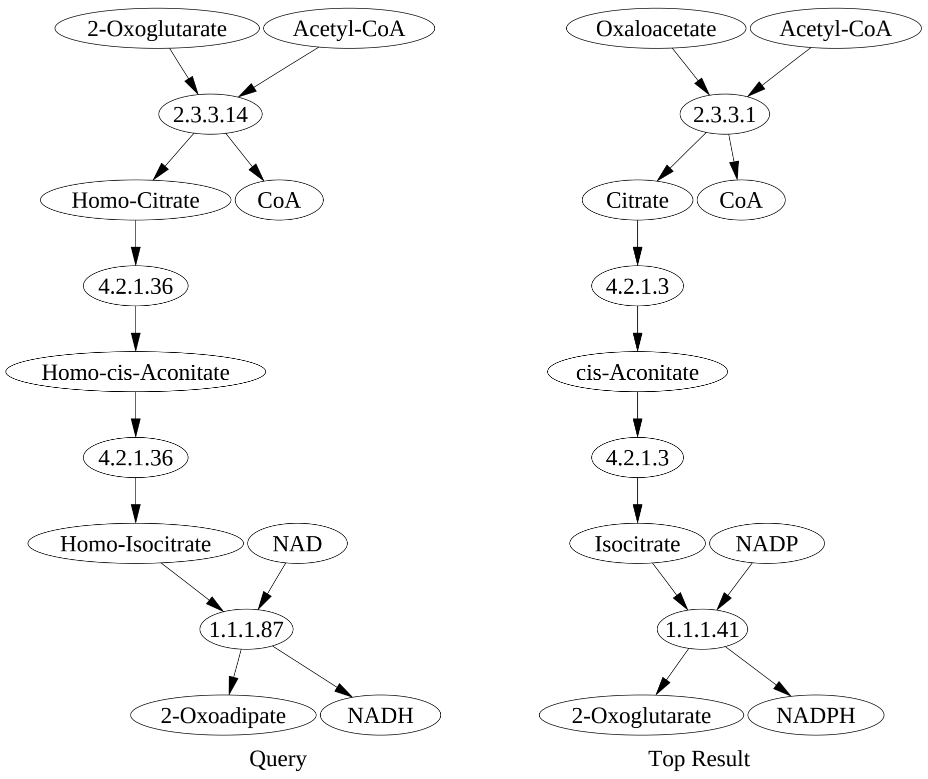


Fig. 10. Top result of using GraphMatch to query a combined network of glycolysis, gluconeogenesis, the citrate cycle and the glyoxylate metabolism in *E. coli* with part of the α-aminoadipic pathway in *T. thermophilus*. Notations are the same as in Figures 7 and 8. Each reaction vertex is labeled by the EC number of the enzyme involved (the same enzyme does not have to come from the same reaction), while each compound vertex is labeled by the name of the compound. At most $m = 2$ mismatches or indels are allowed between two matches. The information content cutoff for enzymes is set so that a match of two enzymes is guaranteed when the first two parts of their EC numbers are the same. The score cutoff for compound similarity is 0.5. Only the information content value for enzymes contributes to the match score. The non-associated vertex penalty Δ_0 is -1.0, while the indel penalty Δ_1 is -0.1.

the two pathways might be evolutionarily related.

F. Discussion

We have shown that PathMatch and GraphMatch are able to find meaningful pathways from biological networks. Due to the polynomial time complexity, PathMatch is generally very fast, taking only seconds to minutes to process a query on a network G with thousands of vertices and tens of thousands of edges when all shortest paths in G were pre-computed, while GraphMatch has exponential time complexity in the worst case and can take many hours in some cases depending on the sizes of the graphs and the parameter settings, which can be adjusted to obtain reasonably sized correspondence lists. Nevertheless, the model employed by GraphMatch has more expressive power and can return more accurate results that are appropriately constrained. For protein interaction networks, since the false positive rate of edges can be quite high (Deane et al. 2002; Deng et al. 2002; Steffen et al. 2002; Scott et al. 2006), it may be desirable to incorporate edge reliability into the score. This can be done in a similar way as in Kelley et al. (2003) and Sharan et al. (2005). To further improve the model for metabolic networks, one needs to investigate how to combine enzyme and compound similarities into a single score. To compute the statistical significance of a path alignment or a graph alignment, one can follow the idea in Kelley et al. (2003) to compare the score of a solution to the scores of random alignments between the query and the input graph G. It is also possible to use other methods besides BLAST or information content, such as using expression data and gene ontology annotations, to establish vertex correspondences (Sharan et al. 2005).

CHAPTER IV

IDENTIFYING GENE CLUSTERS OF CONSTRAINED SIZES IN MULTIPLE GENOMES

A. Introduction

With the increased availability of complete genome sequences, there has been considerable interest to study genome evolution by investigating the clustering of sets of orthologous genes among multiple genomes. There are many attempts to identify conserved clusters of genes through different formulations of gene clusters. Fujibuchi et al. (2000) employed graph-theoretic techniques to detect conserved gene clusters in multiple genomes by identifying highly connected structures in a graph. Uno and Yagiura (2000) represented sets of genes as permutations and developed algorithms to find common intervals within these permutations that represent gene clusters, while Heber and Stoye (2001a,2001b) generalized this model to compare more than two genomes. Vandepoele et al. (2002) and Calabrese et al. (2003) restricted their attention to identify almost colinear regions of similarities in two genomes. Bergeron et al. (2002) and Luc et al. (2003) allowed intervening genes to appear between genes in a cluster and proposed polynomial time algorithms that allow more than two genomes to be analyzed. Didier (2003) and Schmidt and Stoye (2004) generalized the common interval formulation to allow for the inclusion of paralogous genes within a genome by representing sets of genes as sequences rather than permutations and proposed efficient algorithms to find conserved clusters. Eres et al. (2004) suggested a general notion of permutation pattern discovery and developed algorithms to identify maximal clusters that do not allow intervening genes. He and Goldwasser (2005) developed an algorithm that allows both intervening genes and paralogous genes in their formulation. Rahmann and Klau (2006) reduced the problem to an integer linear programming problem that represents common intervals with errors. Parida (2007) generalized the notion of permutation patterns to allow both intervening genes and paralogous genes.

When intervening genes are allowed, the most popular approaches require only that the distance between adjacent genes in a cluster to be small (Bergeron et al. 2002; Luc et al. 2003; He and Goldwasser 2005; Parida 2007), and thus can return extremely large clusters even when the distance allowed is small. We investigate a different formulation of gene clusters which requires that the distance between any pair of genes within a cluster to be small (Hoberman et al. 2004), thus placing a constraint on the overall cluster size. To illustrate how this approach can find different clusters, we run HomologyTeams (He and Goldwasser 2005) on two bacterial genomes *Streptococcus pyogenes* and *Streptococcus pneumoniae* while requiring that the distance between adjacent genes in a cluster to be at most 25,000 base pairs. The largest cluster found contains 131 genes that are within a region of 566 genes in *S. pyogenes* and 584 genes in *S. pneumoniae.* This cluster has an e-value of 1.1×10^{67} in our formulation and thus will not be found by our approach. Instead, our approach finds a much smaller cluster with an e-value of 2.5×10^{-6} that contains 61 genes within a shorter region of 208 genes.

By reducing from the clique problem (Garey and Johnson 1979), we show that the problem is *NP*-hard when multiple genomes are given, thus it is unlikely that the problem can be solved in polynomial time. Nevertheless, we show that a restricted version which requires that orthologous genes are strictly ordered within each cluster can be solved in polynomial time. This simplified model is suitable for finding clusters in regions with a small number of gene rearrangements, after which a detailed analysis can be performed in the localized area. We then consider the unrestricted problem and develop practical exact algorithms that allow paralogous genes within a genome and clusters that may not appear in every genome.

Since a protein may have more than one function determined by different domains, we allow a gene to appear in more than one orthologous group, which is biologically more accurate as an increasing number of genes are assigned to more than one group in the COG database (Tatusov et al. 1997). For example, in *Clostridium acetobutylicum*, the gene *gltA* belongs to three COGs that correspond to distinct domains glut_synthase, GltS_FMN and gltb_C with different functions. We test our algorithms on a set of four bacterial genomes *Bacillus subtilis*, *Streptococcus pyogenes*, *Streptococcus pneumoniae* and *Clostridium acetobutylicum*, and a set of four yeast genomes *Saccharomyces cerevisiae*, *Saccharomyces paradoxus*, *Saccharomyces mikatae* and *Saccharomyces bayanus*. We show that the clusters with the best statistical significances correspond to biologically relevant gene clusters.

A software program (GCFinder) and a list of gene clusters found on the bacterial and the yeast genomes are available at http://faculty.cs.tamu.edu/shsze/gcfinder.

B. Problem Formulation

Let $C = \{c_1, \ldots, c_k\}$ be a set of k chromosomes, one from each genome under consideration. These chromosomes can either be all linear or all circular. We first assume that each gene appears in each sequence exactly once that represents orthologs of each other without allowing paralogous copies. We represent each gene by a number (while ignoring its orientation) so that the same number represents the same gene (or an orthologous gene) in different genomes and each chromosome c_i is represented by a sequence of gene numbers. With this assumption, the length of each sequence is the same and each sequence represents a permutation of the same set of genes. This representation is similar to the ones used in Uno and Yagiura (2000) and Heber and Stoye (2001a,2001b).

Let $G = \{g_1, \ldots, g_n\}$ be the set of n genes that appear in each chromosome. Denote the position of gene g on chromosome c_i by $p_i(g)$, which is the index of g in the sequence of genes on c_i (on circular chromosomes, it is interpreted in a circular fashion). Note that this definition considers each gene as a basic unit instead of each base pair. Given a parameter d, we can think of each subset G' of G with $|G'| \leq d$ as a potential gene cluster. Let $G' = \{g'_1, \ldots, g'_{d'}\} \subseteq G$ with d' genes in the cluster. We say that G' appears in c_i if $|p_i(g'_r) - p_i(g'_s)| < d$ for all pairs of genes g'_r and g'_s in G'. Thus, in order for G' to appear in c_i, all its genes have to be within a region of size at most d on c_i. To evaluate the statistical significance of G', we assume that each permutation is equally likely to occur. The probability of G' appearing in a linear chromosome with n genes was given in Durand and Sankoff (2003) as

$$p(n, d, d') = \left((n-d)\binom{d-1}{d'-1} + \binom{d}{d'}\right) \Big/ \binom{n}{d'}. \tag{4.1}$$

On a circular chromosome, we consider $d \leq n/2$ and this probability is given by

$$p(n, d, d') = n\binom{d-1}{d'-1} \Big/ \binom{n}{d'}. \tag{4.2}$$

Suppose that G' appears in k' chromosomes. We define the p-value of G' to be the probability of G' appearing in at least k' out of k chromosomes and it was given in Durand and Sankoff (2003) as

$$p(k, k', n, d, d') = \sum_{i=k'}^{k} \binom{k}{i} p(n, d, d')^i (1 - p(n, d, d'))^{k-i}. \tag{4.3}$$

We estimate the e-value of G' by

$$e(k, k', n, d, d') = \binom{n}{d'} p(k, k', n, d, d'). \tag{4.4}$$

For each fixed d, we are interested in finding all maximal clusters G' with e-value below a cutoff, where a cluster G' is maximal if there does not exist another cluster G'' such that $G'' \supset G'$ and G'' appears in the same set of chromosomes.

We now generalize the model to allow different paralogous copies of a gene to be assigned the same gene number and remove the requirement that each gene must appear in every chromosome. Each chromosome c_i is no longer represented by a permutation but by a sequence of gene numbers, possibly of different lengths, that allows some gene numbers to appear more than once. This representation is similar to the ones used in Didier (2003) and Schmidt and Stoye (2004). Bergeron et al. (2002), Luc et al. (2003) and He and Goldwasser (2005) used a slightly different notation to represent a gene that does not appear in every chromosome by a star symbol, while we use a gene number to represent each gene, including the ones that do not appear in every chromosome, since these genes can also appear in a gene cluster in our formulation.

A gene cluster G' that appears in k' chromosomes is represented by k lists $P'_1, \ldots, P'_k$, with k' of these lists non-empty and each list P'_i containing a set of positions on c_i, which together specify all the positions of genes that are in G'. To ensure that all genes in G' are clustered within a region of size at most d on each chromosome c_i for which P'_i is non-empty, we require that each gene number that appears in G' must appear at least once in each of the k' non-empty lists P'_i, and within each non-empty P'_i, $|r - s| < d$ for any pair of positions $r, s \in P'_i$. Note that in this definition, it is possible that a gene in G' may have other homologous copies outside G'. For each fixed d, we are interested in finding all maximal clusters G' with

e-value below a cutoff, where a cluster G' is maximal if there does not exist another cluster G'' with position lists $P''_1, \ldots, P''_k$ such that $P''_i \supseteq P'_i$ for all i with $1 \leq i \leq k$. To obtain approximate significance estimates, we use the original e-value formula and replace d' and n by the following averages: $d' = \sum_{P'_i \neq \emptyset} |P'_i|/k'$ and $n = \sum_{i=1}^{k} |c_i|/k$. To further allow a gene to appear in more than one orthologous group, we allow multiple gene numbers to appear at the same position that correspond to the same gene. Each P'_i now contains a set of (gene number, position) pairs and d' becomes the average number of distinct positions among all non-empty P'_i.

C. Problem Hardness

When the order of genes within a cluster is not constrained, we show that the problem is hard even in the simplest case when each gene appears in each chromosome exactly once and each position contains only one gene number.

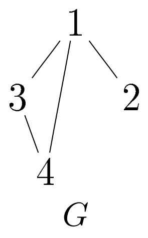
Theorem 1. *Given k sequences of gene numbers, each representing a linear chromosome, and a parameter d denoting the maximum length of a region that can contain a gene cluster, the problem of finding a gene cluster of size $d' \geq 2$ that appears in every sequence is NP-hard.*

Proof. We reduce from the clique problem (Garey and Johnson 1979). Let $G = (V, E)$ be an undirected graph and let $V = \{v_1, \ldots, v_d\}$. We construct a gene cluster problem from G as follows. Let $V' = \{v'_1, \ldots, v'_d\}$ and let $V \cup V'$ be the set of genes. We construct a set of $3d$ gene sequences, each of length $2d$, by defining three groups s_i, s'_i and s''_i, with $1 \leq i \leq d$, as follows (see Figure 11):

$$s_i = v_i \bigodot_{j=1}^{d-|N_{v_i}|-1} v'_j \bigodot_{v_j \in N_{v_i}} v_j \bigodot_{v_j \in V - \{v_i\} - N_{v_i}} v_j \bigodot_{j=d-|N_{v_i}|}^{d} v'_j$$

$$s'_i = v'_i \bigodot_{j=1}^{d} v_j \bigodot_{j \neq i} v'_j$$

$$s''_i = v'_i \bigodot_{j \neq i} v'_j \bigodot_{j=1}^{d} v_j$$



$$\begin{aligned}
s_1 &= (\,1\ 2\ 3\ 4\ 1'\,2'\,3'\,4'\,) \\
s_2 &= (\,2\ 1'\,2'\,1\ 3\ 4\ 3'\,4'\,) \\
s_3 &= (\,3\ 1'\,1\ 4\ 2\ 2'\,3'\,4'\,) \\
s_4 &= (\,4\ 1'\,1\ 3\ 2\ 2'\,3'\,4'\,)
\end{aligned}$$

Group 1

$$\begin{aligned}
s'_1 &= (\,1'\,1\ 2\ 3\ 4\ 2'\,3'\,4'\,) \\
s'_2 &= (\,2'\,1\ 2\ 3\ 4\ 1'\,3'\,4'\,) \\
s'_3 &= (\,3'\,1\ 2\ 3\ 4\ 1'\,2'\,4'\,) \\
s'_4 &= (\,4'\,1\ 2\ 3\ 4\ 1'\,2'\,3'\,)
\end{aligned}$$

Group 2

$$\begin{aligned}
s''_1 &= (\,1'\,2'\,3'\,4'\,1\ 2\ 3\ 4\,) \\
s''_2 &= (\,2'\,1'\,3'\,4'\,1\ 2\ 3\ 4\,) \\
s''_3 &= (\,3'\,1'\,2'\,4'\,1\ 2\ 3\ 4\,) \\
s''_4 &= (\,4'\,1'\,2'\,3'\,1\ 2\ 3\ 4\,)
\end{aligned}$$

Group 3

Fig. 11. Illustration of the reduction from the clique problem with $d = 4$ in the proof of *NP*-hardness of the unordered gene cluster problem, where i represents the original vertices in V and i' represents the additions to obtain the set of genes.

where $\bigodot_c v$ represents the concatenation of all v that satisfy the condition c in increasing index order and $N_v = \{u \in V \mid (u,v) \in E\}$ represents the set of all neighbors of a vertex v in G. Note that group 1 encodes G, while groups 2 and 3 ensure that a gene cluster of size at least two does not contain any genes in V'.

Consider a clique C of size $d' \geq 2$ in G. All vertices in C appear in the first d positions of sequences s_i for which $v_i \in C$. These vertices appear together between the genes in V' in sequences s_i for which $v_i \notin C$ and in sequences s'_i and s''_i. Thus the distance between these vertices must be less than d in all the $3d$ gene sequences and they form a gene cluster. Conversely, consider a gene cluster of size $d' \geq 2$. Since it has to appear in all the $3d$ gene sequences, sequences s'_i and s''_i prevent any genes in V' to be included. The cluster must form a clique in G, since otherwise, there exist two genes v_i and v_j in the cluster with $(v_i, v_j) \notin E$ and they are at least distance d apart in sequences s_i and s_j, a contradiction. ■

It is easy to see that the problem is of a similar difficulty on circular chromosomes and it is at least as hard (if not harder) when paralogous genes are allowed, when a gene cluster does not need to appear in every chromosome, or when some of the positions contain more than one gene number.

D. Finding Ordered Clusters

Since it is not likely that the unconstrained problem can be solved in polynomial time, we consider a restricted version which requires that a cluster appears in every input chromosome, none of the genes appear in more than one orthologous group and orthologous genes are strictly ordered within each cluster without allowing paralogous copies. In Figure 12, we describe an algorithm that finds an optimal cluster with

```
Algorithm OrderedClusters ($\{c_1, \ldots, c_k\}$,$d$) {
    $\mathcal{G} \leftarrow$ empty;
    for each gene $g$ that appears in all $k$ chromosomes do {
        for $i \leftarrow 1$ to $k$ do {
            $W_i \leftarrow$ window of size $d$ starting from $g$ on $c_i$; }
        construct a directed graph $G = (V, E)$, where $V$ is the set of genes that
            appear in every $W_i$, and $(g_1, g_2) \in E$ if $g_1$ is before $g_2$ in each $W_i$;
        $P \leftarrow$ longest path $(g_1, \cdots, g_{d'})$ in $G$;
        add $P$ to $\mathcal{G}$; } }
```

Fig. 12. Algorithm OrderedClusters to find a set of ordered gene clusters that include a maximal cluster with the lowest e-value when paralogous copies of genes are not allowed and none of the genes appear in more than one orthologous group. $\mathcal{G}$ is the set of clusters returned with each cluster represented by an ordered list of genes that appear in each of the k given chromosomes and are colinear among all chromosomes.

the lowest e-value. The idea is to start from each gene g that appears in every given chromosome and find all genes that are less than distance d to the right of g in every chromosome. Then construct a directed acyclic graph G with g and all these genes as vertices in which edges represent strict ordering of two genes within every chromosome. An optimal cluster that starts from g is represented by a longest path in G, which can be found by a procedure similar to topological sorting (Cormen et al. 2001). It is easy to see that an optimal cluster with the lowest e-value must be among one of the longest paths. Figure 13 illustrates one iteration of the algorithm on a small example. Note that windows cannot go beyond the right boundary on a linear chromosome, but they can wrap around on a circular chromosome.

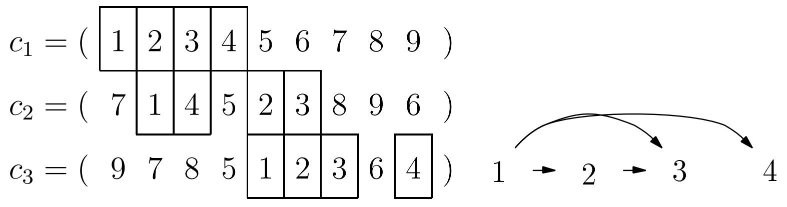

$c_1 = (\boxed{1}\,\boxed{2}\,\boxed{3}\,\boxed{4}\; 5\; 6\; 7\; 8\; 9 \;)$
$c_2 = (7\,\boxed{1}\,\boxed{4}\; 5\,\boxed{2}\,\boxed{3}\; 8\; 9\; 6 \;)$
$c_3 = (9\; 7\; 8\; 5\,\boxed{1}\,\boxed{2}\,\boxed{3}\; 6\,\boxed{4}\;)$
1
2
3
4

Fig. 13. Illustration of the construction of a directed graph G in the OrderedClusters algorithm when $g = 1$ and $d = 5$. Genes that appear in all windows W_i on each chromosome c_i are enclosed in boxes. G is shown to the right of the chromosomes. The longest path $(1, 2, 3)$ is added to $\mathcal{G}$.

To evaluate the time complexity, since each cluster must appear in every chromosome, it suffices to consider each pair of genes that are less than distance d apart on chromosome c_1 in order to determine all vertices and edges of G for each gene g (see Figures 12 and 13). Since there are a total of nd such pairs to consider over all g, it takes $O(knd)$ time to determine the ordering relationships between all pairs. For each gene g, since $|W_i| \leq d$, we have $|V| \leq d$ and $|E| \leq d^2$, and it takes $O(d)$ time and $O(d^2)$ time respectively to determine V and E from the above relationships. It then takes $O(d^2)$ time to find a longest path in each G and $O(k+d)$ time to compute its e-value. Thus the overall time complexity is $O(nd(k + d))$.

To obtain additional clusters that are not related to the optimal cluster, we mask the genes in the optimal cluster and rerun the algorithm to obtain the next cluster. Alternatively, for $l > 1$, we can find top l clusters in one run by finding l longest paths in each directed acyclic graph G, which is well known to be solvable in polynomial time (Lawler 1972; Fox 1975). In particular, we can use the basic algorithm in Eppstein (1998) to create an implicit representation of the l longest paths from the starting gene g to each other vertex in G in $O(|E| + |V| \log |V| + l|V| \log l)$ time and use $O(i)$ time to obtain the ith path from the representation for each G.

E. Finding Unordered Clusters

We now consider the more difficult problem when the order of genes within a cluster is not constrained and paralogous copies of genes are allowed. We first assume that a gene cluster must appear in each of the k given chromosomes. In Figure 14, we describe an algorithm that always guarantees that all maximal clusters are included in the results. The idea is to start with each window of size d on chromosome c_1 and find the locations of all occurrences of its gene numbers in the other chromosomes. We form windows of size d on each chromosome that contain these occurrences and intersect them, k windows at a time with one from each chromosome, to obtain a list of candidate gene clusters G' each containing a list of gene numbers. The positions of the genes in each cluster G' are recovered by investigating how the intersection was originally obtained. It is easy to see that all maximal clusters that appear in each of the k given chromosomes must be included in the results, but it is possible that some non-maximal clusters are also included. Figure 15 illustrates the algorithm on a small example.

To evaluate the time complexity, let t be the maximum number of paralogous copies allowed for a gene within a chromosome and let D be the maximum number of gene numbers within a window of size d (D can be larger than d when some genes appear in more than one orthologous group). For a given window W_1 (see Figures 14 and 15), since $|G_1| \leq D$ and $|G_{ij}| \leq D$, we have $|P_i| \leq tD$ and it takes $O(ktD^2)$ time to construct all G_{ij}. Also, there are a total of at most $(tD)^{k-1}$ intersections to perform with each taking $O(kD)$ time. Since there are $O(n)$ windows of size d on chromosome c_1, the overall time complexity is $O(knD(tD)^{k-1})$.

To allow for clusters that do not appear in every chromosome, for each $i > 1$ we add an extra universal set G_{i0} containing all possible gene numbers and change the

```
Algorithm UnorderedClusters ($\{c_1, \ldots, c_k\}$,$d$) {
    $\mathcal{G} \leftarrow$ empty;
    for each window $W_1$ of size $d$ on chromosome $c_1$ do {
        $G_1 \leftarrow$ set of gene numbers in $W_1$;
        for $i \leftarrow 2$ to $k$ do {
            $P_i \leftarrow$ set of positions on chromosome $c_i$ in which gene numbers in $G_1$
                appear;
            for $j \leftarrow 1$ to $|P_i|$ do {
                $W_j \leftarrow$ window of size $d$ starting from the $j$th position in $P_i$ on $c_i$;
                $G_{ij} \leftarrow$ set of gene numbers in $W_j$ that appear in $G_1$; } }
        for each tuple $(j_2, \ldots, j_k)$ with $1 \le j_i \le |P_i|$ do {
            add $G' = G_1 \cap (\bigcap_{i=2}^{k} G_{ij_i})$ to $\mathcal{G}$; } } }
```

Fig. 14. Algorithm UnorderedClusters to find a set of unordered gene clusters that include all maximal clusters when paralogous genes are allowed while requiring that each cluster appears in each of the k given chromosomes. $\mathcal{G}$ is the set of clusters returned with each cluster G' represented by a list of gene numbers.

$$
\begin{array}{ll}
c_1 = (\ \boxed{1\ 2\ 3}\ 4\ 5\ 6\ 6\ 7\ 8\ 9\ ) & G_1 = \{1,2,3\} \\
c_2 = (\ 9\ 5\ \boxed{2}\,\boxed{3}\ 6\ \boxed{1}\ 4\ \boxed{2}\ 8\ 7\ 9\ ) & G_{21} = \{2,3\}, G_{22} = \{1,3\}, G_{23} = \{1,2\}, G_{24} = \{2\} \\
c_3 = (\ 4\ 8\ 4\ 7\ 8\ 8\ \boxed{1}\ 8\ \boxed{2}\ ) & G_{31} = \{1,2\}, G_{32} = \{2\}
\end{array}
$$

$$
\begin{array}{lll}
G'_1 = G_1 \cap G_{21} \cap G_{31} = \{2\} & G'_4 = G_1 \cap G_{22} \cap G_{32} = \{\} & G'_7 = G_1 \cap G_{24} \cap G_{31} = \{2\} \\
G'_2 = G_1 \cap G_{21} \cap G_{32} = \{2\} & G'_5 = G_1 \cap G_{23} \cap G_{31} = \{1,2\} & G'_8 = G_1 \cap G_{24} \cap G_{32} = \{2\} \\
G'_3 = G_1 \cap G_{22} \cap G_{31} = \{1\} & G'_6 = G_1 \cap G_{23} \cap G_{32} = \{2\} &
\end{array}
$$

Fig. 15. Illustration of the UnorderedClusters algorithm when $d = 3$. On chromosome c_1, the window W_1 under consideration is enclosed in a box. The occurrences of the gene numbers in W_1 in chromosomes c_2 and c_3 are enclosed in boxes. The set of returned clusters is $\mathcal{G} = \{G'_1, \ldots, G'_8\}$.

range of j_i in Figure 14 to $0 \leq j_i \leq |P_i|$. Since G_{i0} has no effect on an intersection, a cluster that does not appear in chromosome c_i can be obtained by an intersection that includes G_{i0}. Note that the addition of G_{i0} is only for convenience of explanation and nothing needs to be done when intersecting with G_{i0}. After this modification, the total number of intersections is at most $(tD+1)^{k-1}$, which is still $O((tD)^{k-1})$. To remove the restriction that a cluster must appear in chromosome c_1, we run the algorithm k times, each time with chromosome set $\{c_i, \ldots, c_k\}$, for $1 \leq i \leq k$. This change increases the time complexity of the entire algorithm by at most a factor of k, resulting in an overall time complexity of $O(k^2 nD(tD)^{k-1})$, with no increase in the exponential part.

To compute the e-value for each of the candidate clusters, since k, n and d are fixed in a particular run, we preprocess and store all values of $e(k, k', n, d, d')$ for different combinations of k' and d' so that each e-value can be obtained in constant time. The preprocessing can be completed in polynomial time and thus takes negligible time. Since not many clusters remain after applying the e-value cutoff, it is easy to remove non-maximal clusters from the truncated list and this also takes negligible time. Although the overall time complexity is exponential, $O((tD)^{k-1})$ is much better than $O(n^{k-1})$ when tD is small relative to n. In this case, our time complexity is better than the $O(n^k)$ time complexity obtained in He and Goldwasser (2005) when their algorithm is applied to k chromosomes with a different formulation of gene clusters.

We next describe a few improvements that do not change the worst case time complexity but will significantly improve the running time in practice while still guaranteeing that no maximal clusters are missed. The simplest such improvement is to remove each G_{ij} that satisfies $G_{ij} \subset G_{ij'}$ for some $j \neq j'$ from among $G_{i1}, \ldots, G_{i|P_i|}$ before intersection. This step does not increase the time complexity since it can be included easily while constructing the sets. The second improvement takes advantage of significant overlaps among windows W_1 on chromosome c_1. Except for the leftmost window, each window contains at most one new position p when compared to the overlapping window that is immediately to its left. For each such window, it suffices to consider only those G_{ij} that contain a gene number at position p during intersection, since other clusters that do not contain a gene number at position p are already included.

F. Application to Bacterial Genomes

We test our algorithm (GCFinder) on a set of four circular bacterial genomes, including *B. subtilis*, *S. pyogenes*, *S. pneumoniae* and *C. acetobutylicum* from the COG database (Tatusov et al. 1997). For genes that have the same COG number, those that are on different genomes are considered orthologs while those that are on the same genome are considered paralogs. We first test the unrestricted version of GCFinder that does not impose gene order restriction within a cluster. We ran GCFinder over $1 \leq d \leq 50$ and found all maximal gene clusters. We examined clusters with low e-values that appear in all the four genomes. Note that there are also many other clusters that have low e-values but do not appear in all the four genomes.

The cluster with the lowest e-value (2.8×10^{-131}) consists of a highly conserved superoperon containing ribosomal proteins and two conserved genes adenylate kinase and initiation factor IF-1 (Figure 16(a)). Except for some intervening genes, the genes in the cluster are perfectly ordered. The superoperon consists of *S10*, *spc* and *alpha* operons. The *S10* operon includes genes *rpsJ*, *rplW*, *rplB*, *rpsS*, *rpsC*, *rplP* and *rpsQ*. The *spc* operon includes genes *rplX*, *rplE*, *rpsN*, *rplF*, *rplR*, *rpsE*, *rpmD*,

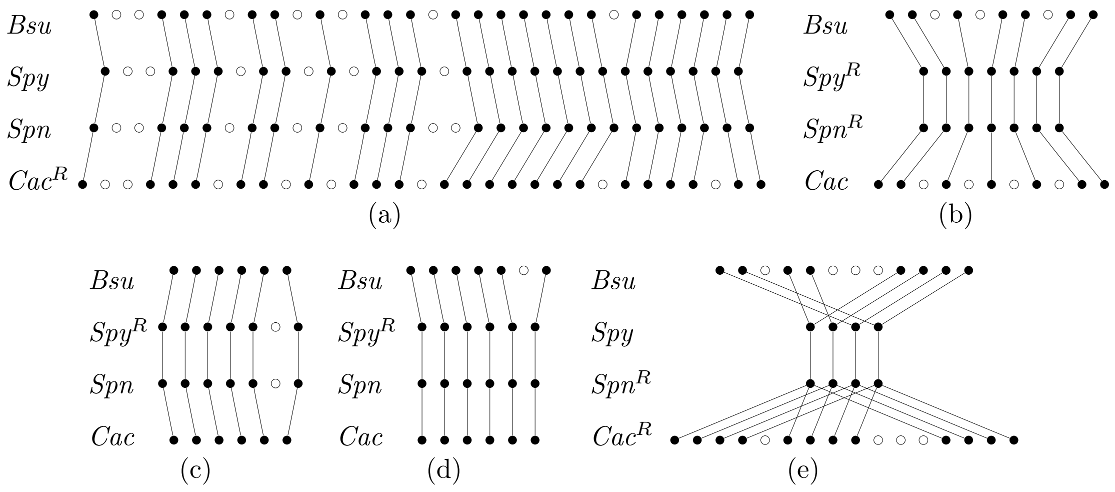


Fig. 16. Top five gene clusters found by GCFinder that appears in all genomes on four bacteria *B. subtilis* (*Bsu*), *S. pyogenes M1 GAS* (*Spy*), *S. pneumoniae TIGR4* (*Spn*), and *C. acetobutylicum* (*Cac*). A sequence of genes are shown from each genome on each row. Filled circles denote genes in a cluster, while hollow circles denote intervening genes. Genes that are in the same COG are connected by lines. Some of the sequences are shown in reversed orientation (denoted by the superscript R) to reduce the number of crossover lines. (a) *S10-spc-alpha* superoperon. (b) *prpC-prkC* operon. (c) *atp* operon. (d) *nusA-infB* operon. (e) Oligopeptide ABC transporter system.

rplO and *secY*. The *alpha* operon contains genes *rpmJ*, *rpsM*, *rpsK*, *rpoA* and *rplQ*. Except for *secY* and *rpoA*, all genes in these operons encode ribosomal proteins. The *S10-spc-alpha* superoperon is among a few regions with extremely conserved gene order in bacterial genomes (Coenye and Vandamme 2005).

The cluster with the second lowest e-value (3.7×10^{-39}) contains genes *gmk*, *yloH*, *priA*, *fmt*, *yloM*, *prpC* and *prkC* in *B. subtilis* (Figure 16(b)). The corresponding genes from the other genomes are either orthologs or are predicted to have similar functions. Although *prpC* and *prkC* have different physiological roles, Iwanicki et al. (2005) showed that they are in the same operon.

The cluster with the third lowest e-value (1.7×10^{-37}) contains the *atp* operon that includes genes *atpD*, *atpG*, *atpA*, *atpH*, *atpF* and *atpE*, which consists of subunits of the F_1F_o ATP synthase (Figure 16(c)). The F_1F_o ATP synthase is an enzyme complex that is important in the free energy metabolism in almost all cells (Koebmann et al. 2000) and is found extensively in bacteria (Das and Ljungdahl 2003).

The cluster with the fourth lowest e-value (1.7×10^{-37}) contains components of the *nusA-infB* operon that includes genes *ylxS*, *nusA*, *ylxR*, *ylxQ*, *infB* and *rbfA* in *B. subtilis* (Figure 16(d)), which was studied in Shazand et al. (1993). The corresponding genes from the other genomes are either orthologs or are predicted to have similar functions.

The cluster with the fifth lowest e-value (1.2×10^{-33}) consists of operons from the oligopeptide ABC transporter system. Multiple paralogous operons were found in *B. subtilis* and *C. acetobutylicum* that are separated by intervening genes (Figure 16(e)). In *B. subtilis*, components of the *app* operon (Koide and Hoch 1994), including genes *appD*, *appF*, *appB* and *appC* with one intervening gene between *appB* and *appC*, and components of the *opp* operon (Steiner and Malke 2001), including genes *oppB*, *oppC*, *oppD* and *oppF*, were found. In *C. acetobutylicum*, components of the *opp* operon were found in addition to two other blocks *CAC3635*, *CAC3636*, *CAC3637*, *CAC3638* and *CAC3641*, *CAC3642*, *CAC3643*, *CAC3644*. These operons are associated with components of the *opp* operon in *S. pyogenes*, and with components of the *ami* operon (Alloing et al. 1990) in *S. pneumoniae*, including genes *amiC*, *amiD*, *amiE* and *amiF*. The *app* operon in *B. subtilis* does not have conserved gene order. Each of these operons includes four components of the five-component oligopeptide ABC transporter system, including two ATPases and two permeases.

We found another cluster with an e-value of 1.5×10^{-20} that contains a paralog

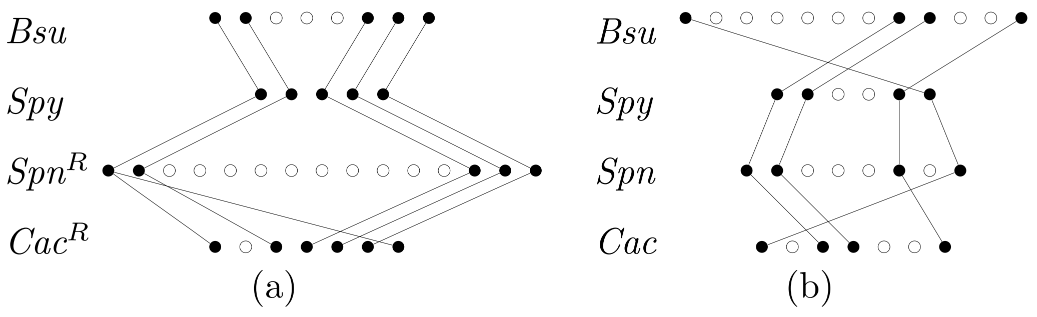


Fig. 17. Two gene clusters found by GCFinder on four bacteria *B. subtilis* (*Bsu*), *S. pyogenes M1 GAS* (*Spy*), *S. pneumoniae TIGR4* (*Spn*), and *C. acetobutylicum* (*Cac*) that have low e-values and many intervening genes. (a) *ihk*-*irr* system. (b) *pur* operon.

with unconserved gene order and many intervening genes (Figure 17(a)). The cluster contains five COGs, with two of them containing putative histidine kinases and response regulators which are related to signal transduction. In particular, the two genes *ihk* and *irr* in *S. pyogenes* constitute a two-component gene regulatory system that is essential for its survival (Voyich et al. 2003). We also found a conserved cluster containing genes *purE*, *purF*, *purM* and *purD* with an e-value of 2.1×10^{-15} and many intervening genes, in which all genes are involved in the purine pathway (Figure 17(b)). All genes except *purE* are in the same order in the cluster, with variable number of intervening genes between them. This *pur* operon was studied in Overbeek et al. (1999).

We were also able to find gene clusters in which some genes appear in more than one orthologous group. Figure 18(a) shows a gene cluster with an e-value of 8.7×10^{-14} that contains three genes *fruR*, *fruK* and *fruA*, which were shown to form an operon in *S. citri* (Gaurivaud et al. 2000). The gene *fruA* was assigned two COG numbers except in *C. acetobutylicum*, where there are two different genes.

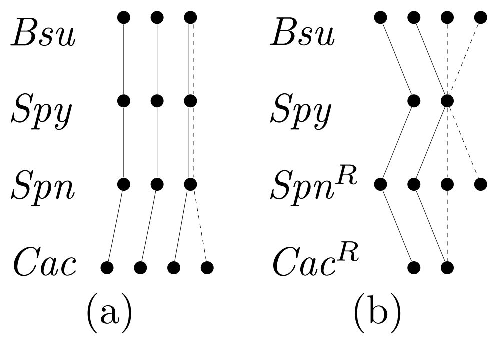


Fig. 18. Two gene clusters found by GCFinder on four bacteria *B. subtilis* (*Bsu*), *S. pyogenes M1 GAS* (*Spy*), *S. pneumoniae TIGR4* (*Spn*), and *C. acetobutylicum* (*Cac*) that have low *e*-values in which some genes appear in two orthologous groups, with one connected by solid lines and the other connected by dashed lines. (a) *fru* operon. (b) Glutamine ABC transporter system.

Figure 18(b) shows a gene cluster with an *e*-value of 8.7×10^{-14} that contains three COGs which form the glutamine ABC transporter system. In *B. subtilis* and *S. pneumoniae*, there are four genes *glnQ*, *glnH*, *glnM* and *glnP* (Caldwell et al. 2001), while in *S. pyogenes* and *C. acetobutylicum*, there are only two genes.

We then apply the restricted version of GCFinder which requires that none of the genes appear in more than one orthologous group and orthologous genes are strictly ordered within each cluster without allowing paralogous copies. We split each COG that has paralogs into potentially more than one orthologous group, by finding a group of genes, one from each genome, that has the best sum of pairwise BLAST log *e*-values (Altschul et al. 1990), then removing the genes in this group and repeating the process. In the *S10-spc-alpha* superoperon, the gene adenylate kinase was excluded due to splitting of its COG during paralog removal. The paralog with unconserved gene order was excluded from the *ihk-irr* system, while *purE* was excluded from the *pur* operon. To obtain the results, it was necessary to reverse the orientation of some of the chromosomes. The oligopeptide ABC transporter system cluster was completely missed. Thus, although this version of GCFinder is much faster and is able to find clusters in regions with a small number of gene rearrangements, the unrestricted version is still needed in other cases.

G. Application to Yeast Genomes

each gene can only appear in one orthologous group due to the way the orthologous groups were generated in Kellis et al. (2003) allow arbitrary windows possible for two species e-value cutoff 10^{-10} surprising study of evolution $1 \leq d \leq 50$ pair of yeasts all d further analysis for unordered clusters running time

We apply the unrestricted version of GCFinder on four yeast genomes, including *S. cerevisiae*, *S. paradoxus*, *S. mikatae* and *S. bayanus*. While the complete genome sequences were used for *S. cerevisiae*, scaffolds were used for the other yeasts, and the orthologous groups were from Kellis et al. (2003). Figure 19 shows the top gene cluster that appears in all genomes with an e-value of 4.1×10^{-95}. In *S. cerevisiae*, the segment that contains this cluster is located on chromosome 13 between genes *YML068W* and *YML035C*. Except for some intervening genes and paralogs, the genes in the cluster are perfectly ordered. We use p-values from the GO term finder in the *Saccharomyces* Genome Database (SGD, Cherry et al. 1998) to evaluate whether genes within this cluster tend to have related functions. Although there are not many common hits in the Molecular Function Ontology, 26 of the 31 genes in the *S. cerevisiae* cluster have common parent GO term “cellular physiological process” in the Biological Process Ontology with a p-value of 0.02, while 22 genes are annotated to the GO term “organelle” in the Cellular Component Ontology with a p-value of 0.04. We also found the gene cluster that appears in all genomes in Figure 1 of Kellis

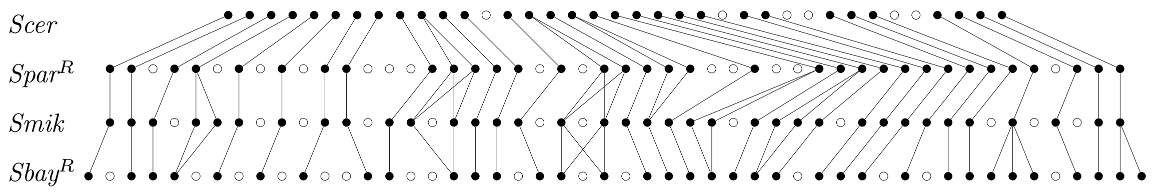


Fig. 19. Top gene cluster found by GCFinder that appears in all genomes on four yeasts *S. cerevisiae* (*Scer*), *S. paradoxus* (*Spar*), *S. mikatae* (*Smik*), and *S. bayanus* (*Sbay*).

et al. (2003) with an e-value of 4.4×10^{-74} with minor differences. For both of these clusters, the restricted version of GCFinder found almost identical structures with paralogous copies omitted.

H. Discussion

We have developed two algorithms: the first finds colinear clusters that appear in all input chromosomes without allowing paralogous genes and is very fast, taking only seconds to minutes to process a set of chromosomes with thousands of genes. When the number of chromosomes k is not very large, it is also possible to relax the requirement to allow a colinear cluster to appear in an arbitrary number of input chromosomes with each chromosome in either one of the two orientations, with a total of 3^k combinations to check. The second is an exact algorithm for the unrestricted problem that guarantees that all maximal unordered clusters with the lowest e-values are found while allowing paralogous genes and clusters that appear in an arbitrary number of input chromosomes. Although it has exponential time complexity and can take hours or days to process a small set of chromosomes, the unrestricted model allows a much wider range of clusters to be identified.

There are other existing software implementations that use different models to predict conserved gene clusters, including FISH (Calabrese et al. 2003), GeneTeams (Bergeron et al. 2002; Luc et al. 2003), HomologyTeams (He and Goldwasser 2005), a generalized version of GeneTeams and HomologyTeams that does not require that a cluster appears in every input chromosome (Kim et al. 2005), and DomainTeams (Pasek et al. 2005). FISH uses a general model to establish correspondences between segments on two chromosomes that may not simply be genes, but imposes an almost colinear ordering of pairwise homologous regions. DomainTeams is a variant of GeneTeams that considers each domain as a unit rather than each gene, while the other algorithms allow multiple genomes and use a gene proximity parameter that restricts the number of intervening genes between adjacent genes in a cluster, which is different from the constraint used in our algorithm GCFinder that restricts the overall cluster size. GeneTeams further requires that there are no paralogous gene copies within a genome, while HomologyTeams uses the number of base pairs as the distance between genes in a cluster instead of using the number of intervening genes. Among these algorithms, FISH, HomologyTeams and GCFinder are the only ones that can provide statistical significance estimates, while the other algorithms report the clusters in no particular order, making it hard to evaluate whether a cluster is significant or not.

The clusters returned by GCFinder are, in most cases, similar to the ones returned by these algorithms. Due to its different formulation, GCFinder can identify clusters that are not found by these algorithms and is thus complementary to these algorithms. One advantage of GCFinder is that it considers the biologically more accurate model in which a gene is allowed to appear in more than one orthologous group that reflects its multiple functions due to different domains. The ability of GCFinder to provide statistical significance estimates while allowing clusters that may not appear in every genome is very important for biologists to identify the most important clusters. Note that other than using COG (Tatusov et al. 1997) or the method in Kellis et al. (2003), there are other ways to establish orthologous and paralogous correspondences (Fujibuchi et al. 2000; Remm et al. 2001; Vandepoele et al. 2002; Calabrese et al. 2003; Luc et al. 2003). To improve the accuracy of the statistical significance estimates, more elaborate estimates that were developed in Durand and Sankoff (2003) can be used.

CHAPTER V

GENE CLUSTER QUERYING

A. Introduction

In bacteria, one of the main mechanisms to facilitate control of gene expression is the organization of genes into operons, in which a number of algorithms are available for their predictions (Salgado *et al.*, 2000; Price *et al.*, 2005; Che *et al.*, 2006). An important strategy to study operons and their evolution is to investigate clustering of related genes within localized regions across multiple bacterial genomes. Since operon structures can be altered by genome rearrangements (Coenye and Vandamme, 2005), it is important to allow the investigation of unrestricted gene clusters that may not correspond to single operons across bacterial genomes. Although existing algorithms are available that can identify gene clusters across two or more genomes, including FISH (Calabrese *et al.*, 2003), GeneTeams (Bergeron *et al.*, 2002; Luc *et al.*, 2003), HomologyTeams (He and Goldwasser, 2005), and a generalized algorithm of GeneTeams and HomologyTeams in Kim *et al.* (2005), very few algorithms are efficient enough to study gene clusters across hundreds of genomes.

To overcome this difficulty, Lee and Sonnhammer (2003) first analyzed each genome separately by identifying clusters of genes that belong to the same metabolic pathway and then compared the results across a large number of genomes. One drawback of such a strategy is that it is not possible to utilize comparative data during the initial analysis. We observe that the following querying strategy can be used to analyze gene clusters across a large number of genomes. Suppose that a list of clusters is given on one of the genomes. For each given cluster Q, first find the locations of all the related genes on each chromosome c. By considering c as a sequence of genes, the distribution of related genes within any window on c can be modeled by the hypergeometric distribution. The list of windows in c with e-values below a cutoff then give rise to a list of clusters on c. An important advantage of such a querying strategy is that it is possible to obtain very high accuracy by choosing initial clusters that have been experimentally confirmed. It is also more sensitive than general-purpose algorithms that do not assume that an initial list of clusters is given, since some of the orthologous genes that belong to a cluster may not appear frequent enough in multiple genomes in order to be detected by these algorithms.

In this paper, we study gene clustering in 400 bacterial genomes by starting from a well-characterized list of operons in *Escherichia coli* K12. We first validate our algorithm by performing queries within its own genome. We then perform comparative analysis of operon occurrences among bacterial groups and study gene orientations within predicted clusters. We also study distributions of rearrangements, both within and across clusters. We show that our algorithm is well suited for analyzing gene clusters across a large number of genomes.

A software program implementing the algorithm (GCQuery) and supplementary data are available at http://faculty.cs.tamu.edu/shsze/gcquery.

B. Methods

We represent each chromosome c by an ordered sequence of genes $(g_1, g_2, \ldots, g_n)$. Given a query cluster Q, our algorithm GCQuery first identifies the set of all related genes on c. This defines a subsequence $c' = (g'_1, g'_2, \ldots, g'_{n'})$ of c. We think of each substring $(g'_j, g'_{j+1}, \ldots, g'_{j+k'-1})$ on c' between the jth gene and the $(j+k'-1)$th gene as a potential gene cluster that spans the window $(g_i, g_{i+1}, \ldots, g_{i+k-1})$ on c between the ith gene and the $(i+k-1)$th gene, where $g_i = g'_j$ and $g_{i+k-1} = g'_{j+k'-1}$ (see

Figures 20 and 21). The probability of finding such a cluster of size at least k' is given by the hypergeometric distribution as

$$p(n, n', k, k') = \sum_{i=k'}^{k} \binom{n'}{i} \binom{n-n'}{k-i} \Big/ \binom{n}{k}. \tag{5.1}$$

The expected number of such clusters that span a window of length k on c is given by the e-value

$$e(n, n', k, k') = (n - k + 1)p(n, n', k, k'). \tag{5.2}$$

Note that windows cannot extend beyond the right end on a linear chromosome (see Figures 20 and 21), but they can wrap around on a circular chromosome. Since n and n' are fixed, we precompute and store all the $O(n)$ binomial coefficients. For fixed k', $p(n, n', k, k')$ can then be obtained from $p(n, n', k, k' - 1)$ in constant time, and it takes $O(n^2)$ time and space to obtain all the possible e-values. Since each cluster $(g'_j, g'_{j+1}, \dots, g'_{j+k'-1})$ can be obtained from the previous one in constant time by removing g'_{j-1} and adding $g'_{j+k'-1}$, the time complexity of the algorithm is $O(n^2)$.

We use the above algorithm to study the organization of bacterial gene clusters by starting from a list of 123 *E. coli* K12 operons that are experimentally validated and contain at least four genes from the RegulonDB database (Huerta *et al.*, 1998), with protein sequences from the MG1655 strain of *E. coli* K12 (Blattner *et al.*, 1997). We analyze related clusters in 400 completely sequenced bacterial genomes with taxonomy information (Wheeler *et al.*, 2000; see Supplementary Figure S1 for all results). We follow the classification approach on the NCBI web site (http://www.ncbi.nlm.nih.gov/genomes/lproks.cgi) and divide the genomes into 18 groups (Table V left). While *E. coli* K12 belongs to the Gammaproteobacteria class, Alphaproteobacteria, Betaproteobacteria, Gammaproteobacteria, Deltaproteobacte-

```
Algorithm GCQuery($Q$,$c$) {
  $c' \leftarrow$ subsequence $(g'_1, g'_2, \ldots, g'_{n'})$ of $c = (g_1, g_2, \ldots, g_n)$ such that each $g'_i$
    is related to at least one gene in $Q$;
  for $k' \leftarrow 1$ to $n'$ do {
    for $j \leftarrow 1$ to $n' - k' + 1$ do {
      compute $e(n, n', k, k')$ of the cluster $(g'_j, g'_{j+1}, \ldots, g'_{j+k'-1})$ on $c'$
        that spans the window $(g_i, g_{i+1}, \ldots, g_{i+k-1})$ on $c$,
        where $g_i = g'_j$ and $g_{i+k-1} = g'_{j+k'-1}$; } } }
```

Fig. 20. Algorithm GCQuery to find all related gene clusters on a linear chromosome c from a query cluster Q.

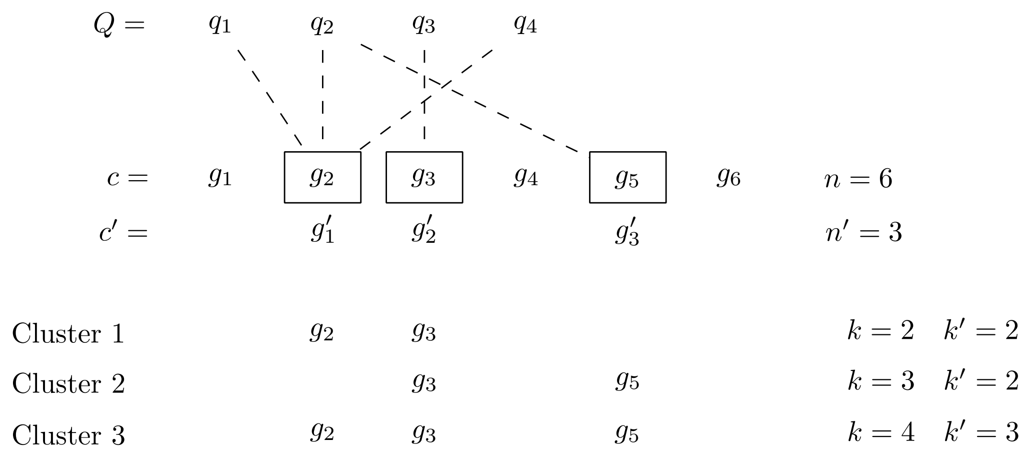

$Q =$ q_1 q_2 q_3 q_4
$c =$ g_1 g_2 g_3 g_4 g_5 g_6 $n = 6$
$c' =$ g'_1 g'_2 g'_3 $n' = 3$
Cluster 1 g_2 g_3 $k = 2$ $k' = 2$
Cluster 2 g_3 g_5 $k = 3$ $k' = 2$
Cluster 3 g_2 g_3 g_5 $k = 4$ $k' = 3$

Fig. 21. Illustration of all clusters of size greater than one on a linear chromosome c from a query cluster Q. Dashed lines denote related genes. It is possible that each gene in Q can be related to more than one gene in c and vice versa.

Table V. Number of genomes in each group (#) and the minimum, maximum and overall percentage under four categories in each group. The minimum and maximum percentages are computed over all genomes in each group, while the overall percentage is computed by dividing the number of entries that satisfy the condition within a category by the total number of entries considered. For each pair of *E. coli* K12 operon and bacterial genome, only one significant cluster (if it exists) with the lowest e-value is considered. The four categories are as follows (see the main text for details). (a) Percentage of occurrences of operons that are significant. (b) Percentage of significant clusters in which all genes share the same orientation. (c) Percentage of conserved neighboring gene pairs within significant clusters. (d) Percentage of conserved neighboring cluster pairs.

		(a) Occurrence rate			(b) Gene orientation			(c) Gene neighbors			(d) Operon neighbors		
Group	#	Min	Max	All	Min	Max	All	Min	Max	All	Min	Max	All
Acidobacteria	1	22.0	22.0	22.0	96.3	96.3	96.3	90.4	90.4	90.4	22.2	22.2	22.2
Actinobacteria	34	8.9	27.6	18.5	70.6	100.0	83.2	81.1	98.1	87.6	11.1	54.5	21.0
Alphaproteobacteria	52	4.9	42.3	19.8	63.5	100.0	86.0	78.2	100.0	87.3	7.7	66.7	20.7
Betaproteobacteria	36	17.1	49.6	33.9	79.5	100.0	91.9	89.3	96.4	93.2	8.0	33.3	16.8
Gammaproteobacteria	98	13.0	100.0	45.8	81.1	100.0	94.8	88.6	99.8	96.6	6.1	94.8	43.0
Deltaproteobacteria	14	10.6	31.7	23.5	68.6	100.0	88.1	86.1	98.4	90.5	8.0	38.5	14.6
Epsilonproteobacteria	11	9.8	19.5	14.1	70.6	100.0	93.2	82.4	92.6	86.8	8.7	35.7	18.8
Aquificae	1	8.1	8.1	8.1	90.0	90.0	90.0	82.0	82.0	82.0	10.0	10.0	10.0
Bacteroidetes/Chlorobi	10	8.1	17.9	15.0	90.0	100.0	97.3	80.2	95.7	88.5	18.8	52.9	29.3
Chlamydiae/Verrucomicrobia	11	8.1	15.4	10.3	90.0	100.0	92.8	89.4	98.0	96.2	15.8	38.5	30.2
Chloroflexi	2	13.8	15.4	14.6	73.7	88.2	80.6	89.0	91.7	90.3	10.5	11.8	11.1
Cyanobacteria	19	7.3	16.3	9.5	60.0	100.0	79.6	83.3	100.0	90.4	9.1	50.0	33.0
Deinococcus-Thermus	4	12.2	15.4	13.4	81.2	100.0	89.4	85.1	89.3	86.7	25.0	42.1	33.3
Firmicutes	95	2.4	31.7	14.8	64.0	100.0	92.0	79.9	100.0	90.5	7.7	87.5	23.4
Fusobacteria	1	12.2	12.2	12.2	93.3	93.3	93.3	91.2	91.2	91.2	13.3	13.3	13.3
Planctomycetes	1	11.4	11.4	11.4	100.0	100.0	100.0	89.8	89.8	89.8	14.3	14.3	14.3
Spirochaetes	9	6.5	13.0	10.5	80.0	100.0	94.0	93.0	100.0	96.5	13.3	50.0	28.3
Thermotogae	1	16.3	16.3	16.3	90.0	90.0	90.0	87.1	87.1	87.1	40.0	40.0	40.0
Total	400	2.4	100.0	24.9	60.0	100.0	91.3	78.2	100.0	93.1	6.1	94.8	31.0

ria Epsilonproteobacteria belong to the Proteobacteria phylum. The largest group contains 98 bacterial genomes, while seven groups contain four or less genomes. We consider a gene in a query cluster to be related to a gene in a genome if their protein-protein BLAST *e*-value is less than 10^{-5} when the set of all genes in the genome is used as a database (Altschul *et al.*, 1990). We consider a predicted cluster to be significant if its *e*-value from the GCQuery algorithm is less than 10^{-5}. Note that a predicted cluster does not necessarily correspond to a single operon since the orientations of genes within the cluster may not be the same and there may be intervening genes in the cluster.

C. Results

1. Gene clusters on *E. coli* K12

To validate our algorithm, we consider each of the 123 *E. coli* K12 operons and apply GCQuery within its own genome to find significant clusters (see Supplementary Figure S2 for complete results). All operons were found in their entirety except for the *thrLABC* operon, which is involved in threonine biosynthesis (Cossart *et al.*, 1981). The gene *thrL* was missing since its protein consists of 21 amino acids and is thus difficult to detect with a low *e*-value. Within *E. coli* K-12, a few sets of homologous gene clusters were found, including operons *narGHJI* and *narZYWV*, which encode the alpha, beta, delta and gamma units of nitrate reductase 1 and nitrate reductase 2 respectively (Blasco *et al.*, 1990). Both operons were found when starting from either operon.

The four-component operon *fixABCX* of the anaerobic carnitine metabolism consists of *fixA*, *fixB*, *fixC* and *fixX* (Walt and Kahn, 2002). The putative five-component operon *ydiQRST-fadK* contains *ydiQ*, *ydiR*, *ydiS*, *ydiT* and *fadK*. The proteins *fixA*, *fixB*, *fixC* and *fixX* have high sequence similarity with *ydiQ*, *ydiR*, *ydiS* and *ydiT* respectively (Campbell *et al.*, 2003). When *fixABCX* is used as a query, both *fixABCX* and a cluster containing *ydiQ*, *ydiR*, *ydiS* and *ydiT* were found. When *ydiQRST-fadK* is used as a query, both *ydiQRST-fadK* and *fixABCX* were found.

In the query results from the type 1 fimbrial operon *fimAICDFGH* (Lane *et al.*, 2007), in addition to itself, two additional clusters were found, including *ycbQR-SUVF* with e-value 1.49×10^{-11} and *yfcPSTUV* with e-value 1.31×10^{-8} (Figure 22). Although the genes within these two clusters are of unknown function, the clusters strongly resemble the two predicted transcription units *ycbRSTUVF* and *yfcOPQRSTU* in Moreno-Hagelsieb and Collado-Vides (2002). This shows that GCQuery can be used to search for potential new operons or gene clusters.

2. Comparative analysis of bacterial groups

We study the distribution of occurrences of the 123 *E. coli* K12 operons in 18 bacterial groups. We define the occurrence rate in a genome to be the percentage of *E. coli* K12 operons that have a significant predicted cluster in the genome. We define the overall occurrence rate in a group of genomes to be the percentage of pairs of operon and genome in the group that have significant occurrences. Table V(a) shows the minimum, maximum and overall occurrence rates within each group (see Supplementary Table S1 for detailed results). In *Mycoplasma mobile* 163K, only three operons (2.4%) had significant occurrences. In fact, these three operons were the only operons that occurred in all the 400 bacterial genomes (see below). In *E. coli* W3110, all the operons had significant occurrences, which is not surprising since it is also one of the *E. coli* K12 strains. In addition to *E. coli* K12 strains, six other *E. coli* strains are completely sequenced in our data set, including *E. coli* 536, *E. coli*

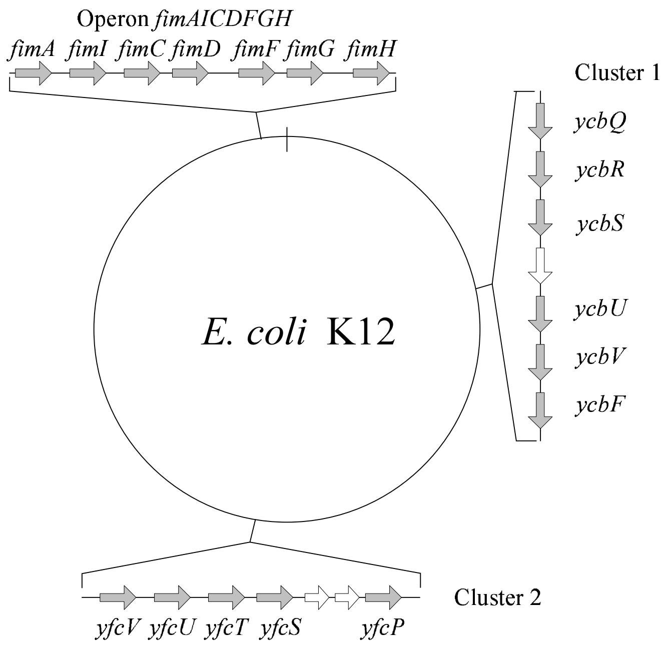


Fig. 22. Query results from the *fimAICDFGH* operon and their positions on the *E. coli* K12 genome. Genes within the clusters are represented by filled arrows while intervening genes are represented by hollow arrows.

APEC O1, *E. coli* CFT073, *E. coli* O157:H7 EDL933, *E. coli* O157:H7 str. Sakai and *E. coli* UTI89. The occurrence rates for these six *E. coli* strains were 90.2%, 90.2%, 91.1%, 94.3%, 95.1% and 90.2% respectively, which indicate that only a small number of *E. coli* operons are not conserved during evolution. The overall occurrence rate in the 400 bacterial genomes was 24.9%. Only two groups had overall occurrence rate over 24.9%. The highest was the group Gammaproteobacteria, to which *E. coli* K12 belongs, with overall occurrence rate 45.8%. The variations in occurrence rates within the group were very high, ranging from 13.0% to 100.0%. Another group was Betaproteobacteria, with overall occurrence rate 33.9%.

3. Comparative analysis of operon occurrences

We study occurrences of the 123 *E. coli* K12 operons in the 400 bacterial genomes. Overall, 104 operons (84.6%) had occurrences in less than half of the 400 bacterial genomes (Figure 23; see Supplementary Table S2 for detailed results). Only three operons (2.4%) had occurrences in all the 400 bacterial genomes, including *S10*, *spc* and *alpha*, which have high similarity among bacterial genomes (Watanabe *et al.*, 1997). Coenye and Vandamme (2005) studied the organization of these operons in 99 bacterial genomes and found that many bacterial genomes miss ribosomal proteins that appear in *E. coli*.

In the *S10* operon, *S10*, *L3*, *L4*, *L23*, *L2*, *S19*, *L22*, *S3*, *L16*, *L29* and *S17* encode ribosomal proteins. While *S3* and *L16* occurred within significant clusters in all the 400 bacterial genomes, the occurrence rates of *S10*, *L3*, *L4*, *L2*, *S19*, *L22* and *S17* were all over 97%. The genes *L23* and *L29* were much less conservative with occurrence rates 63.2% and 62.8% respectively (Figure 24 left; see Supplementary Table S3 for detailed results). The smallest significant cluster found contains four consecutive genes in *Silicibacter pomeroyi* DSS-3 with *e*-value 1.4×10^{-7}, which are

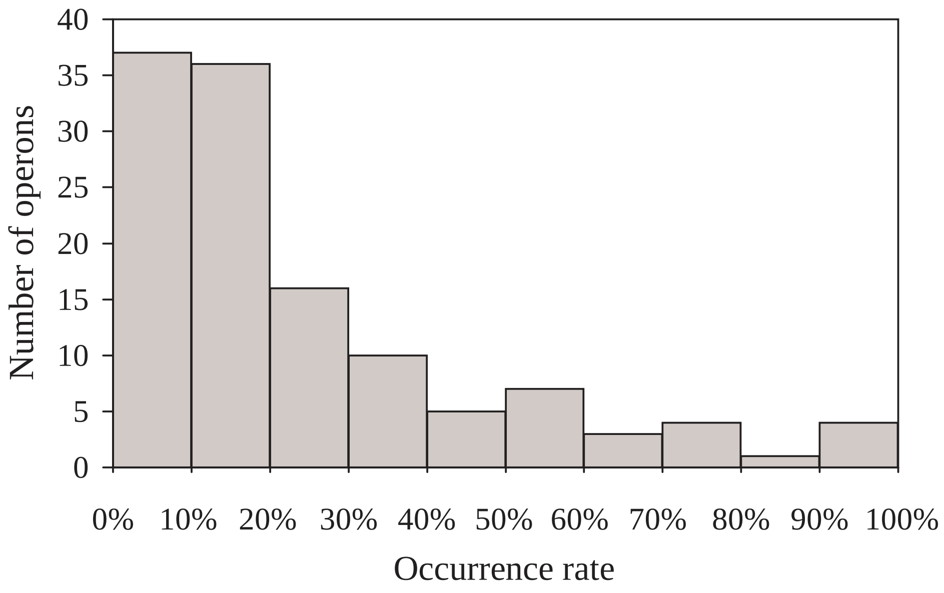


Fig. 23. Distribution of the occurrence rate of the 123 *E. coli* K12 operons in the 400 bacterial genomes.

orthologs of *S10*, *L3*, *L4* and *L23*. In *S. pomeroyi* DSS-3, a cluster of seven genes was also found with *e*-value 1.8×10^{-13}, which are orthologs of the rest of the operon. Interestingly, these two fragments together were similar to the operon, but there were 1077 intervening genes between them.

In the *spc* operon, *L14*, *L24*, *L5*, *S14*, *S8*, *L6*, *L18*, *S5*, *L30*, *L15* and *L36* encode ribosomal proteins, while *secY* encodes a preprotein translocase membrane subunit. Only *S8* occurred within significant clusters in all the 400 bacterial genomes, while the occurrence rates of *L14*, *L24*, *L5*, *S14*, *S8*, *L6*, *L18*, *S5* and *secY* were all over 98%. The genes *L30*, *L36* and *L15* were much less conservative with occurrence rates 78.8%, 67.0% and 38.2% respectively (Figure 24 middle; see Supplementary Table S4 for detailed results).

The *alpha* operon consists of *S13*, *S11*, *S4*, *rpoA* and *L17*. While the three genes *S11*, *rpoA* and *L17* occurred within significant clusters in all the 400 bacterial

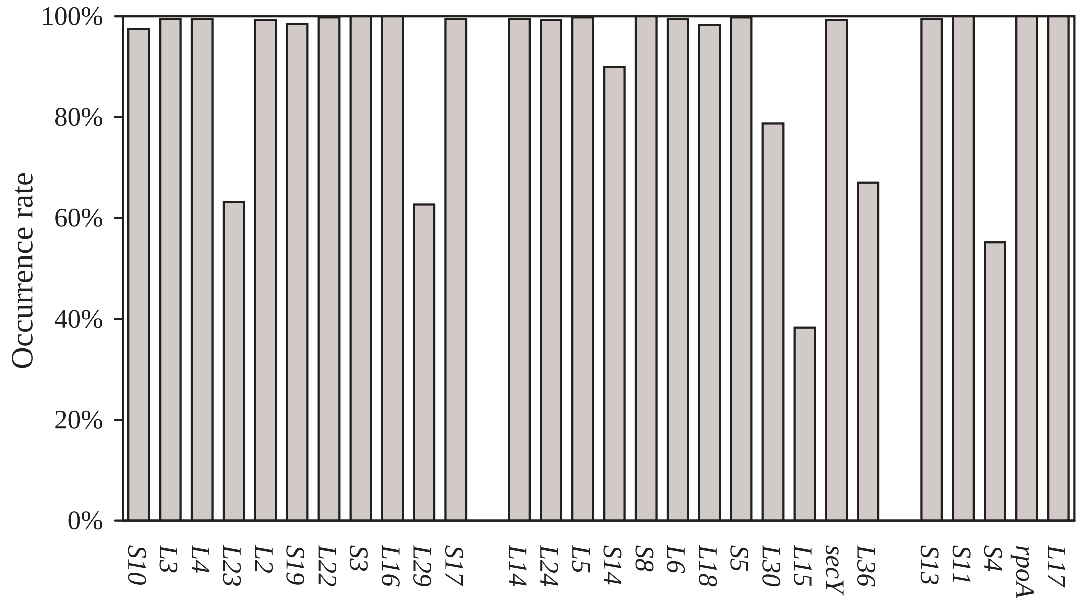


Fig. 24. Distribution of the occurrence rate of genes within significant clusters in *S10*, *spc* and *alpha* operons in the 400 bacterial genomes.

genomes, the occurrence rates of *S13* and *S4* were 99.5% and 55.2% respectively (Figure 24 right; see Supplementary Table S5 for detailed results). The smallest significant cluster found contains three genes in *Magnetospirillum magneticum* AMB-1 with *e*-value 1.2×10^{-6}, which are orthologs of *S11*, *rpoA* and *L17*. Significant clusters in other bacterial genomes contain at least four genes.

The operon with the lowest occurrence rate (1.5%) in the 400 bacterial genomes was the *frl* operon, whose genes are mainly involved in frutoselysine related metabolism (Wiame and Van Schaftingen, 2004). The operon occurred in six bacterial genomes only, including *E. coli* O157:H7 EDL933 with *e*-value 4.9×10^{-8}, *E. coli* O157:H7 str. Sakai with *e*-value 5.1×10^{-8}, *E. coli* W3110 with *e*-value 1.4×10^{-7}, *Shigella boydii* Sb227 with *e*-value 7.0×10^{-8}, *Shigella sonnei* Ss046 with *e*-value 7.6×10^{-8} and *Clostridium acetobutylicum* ATCC 824 with *e*-value 7.4×10^{-6}. The first five genomes belong to Gammaproteobacteria, while *C. acetobutylicum* ATCC 824 belongs to Firmicutes (see Supplementary Table S6 for detailed results).

The above results indicate that although very few *E. coli* K12 operons are shared by all the 400 bacterial genomes, the counterparts of most operons can be found in many bacteria. The GCQuery algorithm allows the evaluation of various hypotheses concerning evolution and conservation of gene clusters.

4. Gene orientations within clusters

One of the most important characteristics of an operon is that all genes are transcribed in the same direction. Although the GCQuery algorithm does not require that genes in a cluster must all have the same orientation, the fact that all genes within a predicted cluster have the same orientation provides additional evidence that it is probably an operon.

We study the distribution of gene orientations within these clusters. With the requirement that at most one significant cluster with the lowest e-value is considered for each pair of *E. coli* K12 operon and bacterial genome, there were a total of 12231 significant clusters. We define the percentage of clusters in which all genes share the same orientation in a genome only among these significant clusters. We define the overall percentage in a group of genomes to be the percentage over all significant clusters in the group of genomes. Within 91.3% of the 12231 clusters, all genes share the same orientation (Table V(b); see Supplementary Table S7 for detailed results). Within 13 groups (72.2%), one or more bacterial genomes had clusters in which all genes share the same orientation. In fact, in 93 bacterial genomes (23.3%), all clusters contain only one gene orientation.

Among the bacterial genomes, *Synechococcus sp.* CC9311 had the lowest percentage (60.0%) of clusters that contain only one gene orientation. While there were a total of ten predicted clusters in *Synechococcus sp.* CC9311, four of them had genes with different orientations. The operon *atpIBEFHAGDC* consists of nine genes that are subunits of ATP synthases (Kasimoglu *et al.*, 1996). In *Synechococcus sp.* CC9311, the predicted cluster consists of six genes that are separated into two parts with 26 intervening genes between them, with homologs of *atpD* and *atpC* in a different orientation from the other genes that are homologs of *atpB*, *atpH*, *atpA* and *atpG*. The operon *nuoABCEFGHIJKLMN* encodes the subunits of an NADH dehydrogenase (Archer and Elliott, 1995). In *Synechococcus sp.* CC9311, a smaller cluster of four genes was found that are separated into two parts, with the first part containing homologs of *nuoH* and *nuoK*, the second part containing homologs of *nuoL* and *nuoM*, and each part in a different orientation. The operon *cyoABCDE* encodes the cytochrome *o* oxidase complex (Cotter *et al.*, 1990). In *Synechococcus sp.* CC9311, a cluster of four genes was found that include homologs of *cyoA*, *cyoB*, *cyoC* and *cyoE*, with the homolog of *cyoE* in a different orientation from the other genes. The operon *moaABCDE* consists of genes that are involved in molybdopterin synthesis (Tao *et al.*, 2005). In *Synechococcus sp.* CC9311, a cluster of four genes was found that include homologs of *moaA*, *moaC*, *moaE* and *moaB* in the given order, with homologs of *moaE* and *moaA* in a different orientation from homologs of *moaC* and *moaB*.

We are also interested in the effect of varying the BLAST *e*-value cutoff that defines related genes on gene orientations. As the cutoff decreases from 10^{-5} to 10^{-60}, the total number of related genes decreases and the overall percentage of clusters that contain only one gene orientation increases from 91.3% to 98.3% (Figure 25). This shows that a more stringent requirement can affect the results.

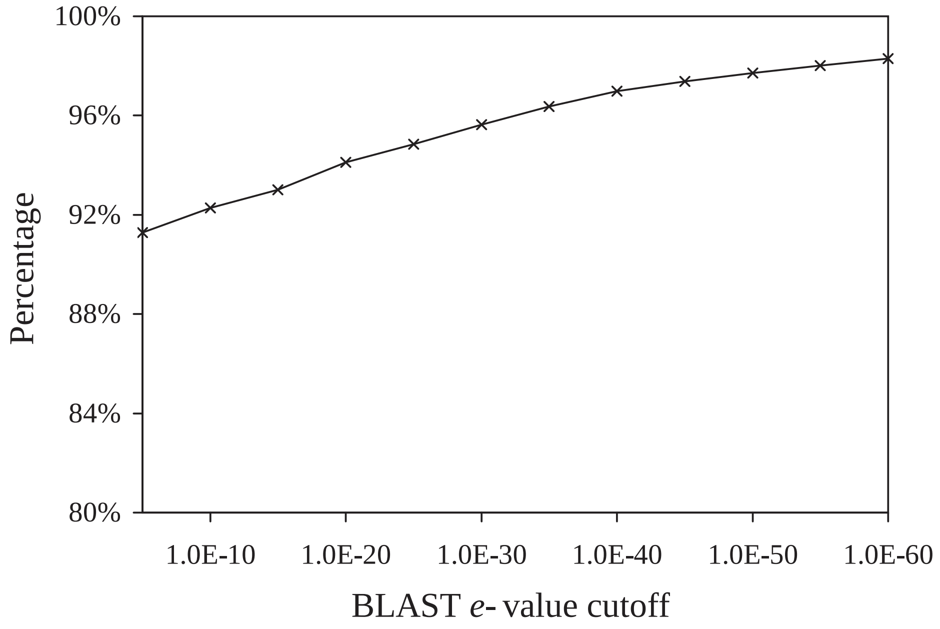


Fig. 25. Percentage of clusters in which all genes share the same orientation for different BLAST e-value cutoffs.

5. Rearrangements within clusters

In addition to common gene orientations, the spatial arrangement of genes within bacterial operons is important for function, expression and regulation of these genes (Itoh *et al.*, 1999; Tamames, 2001). We study the distribution of gene order within the 12231 clusters described above. For a given pair of *E. coli* K12 operon Q and predicted cluster C and a set of correspondences with each of them linking a gene in Q to a related gene in C, we obtain a subset of one-to-one corresponding pairs of links as follows: if there are more than one link for a gene in Q, only retain the one with the lowest BLAST e-value. After this step, if there are more than one link for a related gene in C, only retain the one with the lowest BLAST e-value. In the remaining set of k genes in Q and k related genes in C, assign a label from $+1$ to $+k$ to each gene in Q according to the order of genes in *E. coli* K12, then assign the

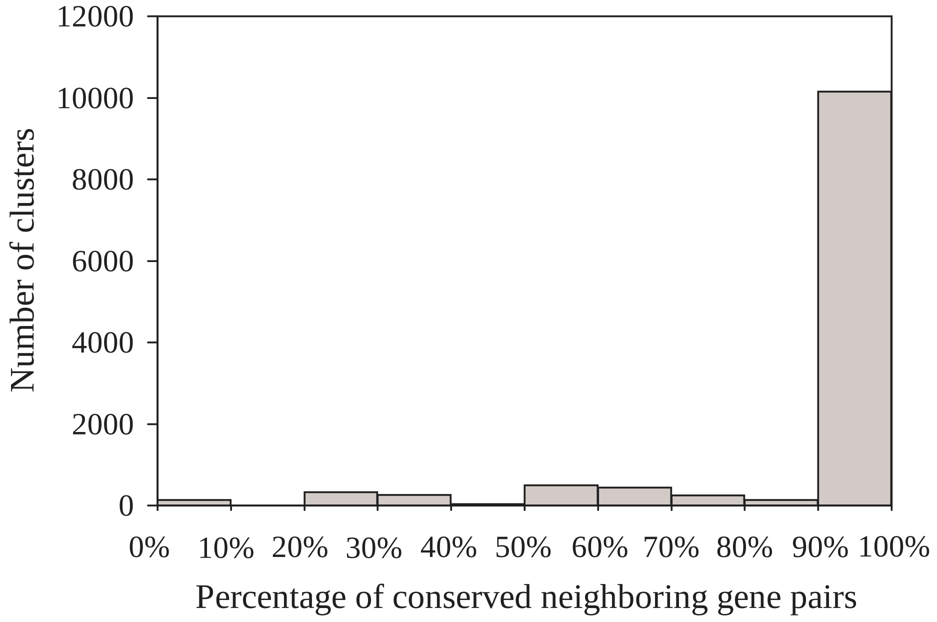


Fig. 26. Distribution of the percentage of conserved neighboring gene pairs over all clusters.

corresponding label for each related gene in C, while giving it a '+' direction if the related gene is on the forward strand within the cluster and a '$-$' direction if the related gene is on the reverse strand within the cluster. The sequence of k genes in C then corresponds to a signed permutation, in which each neighboring gene pair in C with labels l_1 and l_2 is considered to be a breakpoint if l_1 and l_2 are not consecutive, that is, $|l_1 - l_2| \neq 1$ (Kececioglu and Sankoff, 1995). We define the percentage of conserved neighboring gene pairs to be the total number of neighboring gene pairs that are not breakpoints divided by $k - 1$ (which is the total number of neighboring gene pairs), and use it to evaluate the degree of conservation of gene order. Among the 12231 clusters, 83.0% of them had perfectly conserved neighboring gene pairs (Figure 26), which means that the gene order within *E. coli* K12 and the gene order within each cluster are the same either in the forward or in the reverse direction.

When more than one pair of operon and predicted cluster are considered together, we define the overall percentage of conserved neighboring gene pairs to be the total number of neighboring gene pairs that are not breakpoints over all pairs of operon and predicted cluster divided by the total number of neighboring gene pairs over all pairs of operon and predicted cluster. Table V(c) shows that the conservation of neighboring gene pairs was in general very high (see Supplementary Table S8 for detailed results). Among all the neighboring gene pairs, 93.1% of them were conserved. Only four groups had overall percentage over 93.1%, including Gammaproteobacteria (96.6%), Spirochaetes (96.5%), Chlamydiae/Verrucomicrobia (96.2%) and Betaproteobacteria (93.2%). Among the bacterial genomes, *Sinorhizobium meliloti* 1021 had the lowest overall percentage (78.2%), with 13 clusters having non-conserved neighboring gene pairs among a total of 28 clusters.

Although gene order within operons can be unstable (Itoh *et al.*, 1999), our results on gene orientation and gene order indicate that predicted clusters tend to contain only one gene orientation and the gene order tends to be conserved.

6. Rearrangements across clusters

While most previous approaches in analyzing genome rearrangements consider each gene as a basic unit (Kececioglu and Sankoff, 1995), we study genome rearrangements also at the level of gene clusters. For each bacterial genome, we collect at most one significant cluster with the lowest e-value for each *E. coli* K12 operon. For each chromosome c, we assign a label from 1 to k for each operon that has a significant cluster on c according to the order of operons in *E. coli* K12. Then assign the corresponding label to each significant cluster on c according to the starting position of the window that the cluster occupies. If the starting window positions of two clusters are the same, then the clusters are ordered according to the ending window positions. Since our results indicated that none of the predicted clusters from different operons occupy exactly the same window, this always breaks a tie. The sequence of k clusters on c then corresponds to an unsigned permutation in which each neighboring cluster pair on c with labels l_1 and l_2 is considered to be a breakpoint if l_1 and l_2 are not consecutive, that is, $|l_1 - l_2| \neq 1$ (Kececioglu and Sankoff, 1995). We define the percentage of conserved neighboring cluster pairs to be the total number of neighboring cluster pairs that are not breakpoints divided by k (which is the total number of neighboring cluster pairs on a circular chromosome).

When more than one chromosome are considered together, we define the overall percentage of conserved neighboring cluster pairs to be the total number of neighboring cluster pairs that are not breakpoints over all chromosomes divided by the total number of neighboring cluster pairs over all chromosomes. Among all the neighboring cluster pairs, only 31.0% of them were conserved (Table V(d); see Supplementary Table S9 for detailed results). Among all groups, Gammaproteobacteria, to which *E. coli* K12 belongs, had the highest overall percentage (43.0%). The percentage was the highest in *E. coli* O157:H7 EDL933 (94.8%), which belongs to Gammaproteobacteria, with six non-conserved neighboring cluster pairs among a total of 116 neighboring cluster pairs. Interestingly, *Psychrobacter cryohalolentis* K5, which had the lowest percentage (6.1%), is also in this group.

When compared to rearrangements within clusters, our results indicate that large-scale rearrangements at the level of clusters are much more pronounced, although the degree of conservation can still be very high among closely related bacterial genomes.

D. Discussion

We have demonstrated that our querying strategy is well suited for analyzing gene clusters across a large number of genomes. Due to the speed of the algorithm, we were able to obtain all the results on 400 bacterial genomes in less than one day. Since the algorithm does not make any assumptions on the orientation or the density of genes within clusters and only requires that genes in a significant cluster are closer together than expected by chance, it can also be used for analyzing gene clusters on higher organisms such as yeast. Other than using BLAST to establish relations between genes or proteins, it is also possible to use other methods such as COG (Tatusov *et al.*, 1997) or Inparanoid (Remm *et al.*, 2001) to identify related genes that are orthologs or paralogs.

CHAPTER VI

CONCLUSION AND FUTURE WORK

In this dissertation, we improve and develop algorithms for five biological pattern finding problems. For the multiple sequence alignment problem, we propose an alternative formulation in which a final alignment is obtained by preserving pairwise alignments specified by edges of a given tree. In contrast with traditional NP-hard formulations, our preserving alignment formulation can be solved in polynomial time without using a heuristic while having very good accuracy. Such a formulation is very important as it makes it possible to know what the alignment means and also ensures that the optimal solution can be found. In order to align k sequences, a tree of k nodes is required to determine the set of pairwise alignments that are to be preserved in the final alignment. Sometimes only some of the sequence pairs are required to be preserved, or multiple trees or graphs that represent the relationships among sequences are given. For future work, it may be interesting to develop efficient algorithms to solve such more general preserving alignment problems.

For the path matching problem, we take advantage of the linearity of the query path to reduce the problem to finding a longest weighted path in a directed acyclic graph. By using known algorithms for the k-shortest path problem, we can find k paths with top scores in a network from the query path in polynomial time. By allowing a vertex to appear multiple times in the results, our formulation is more consistent to biological facts and is also biologically useful in identifying multiple roles of a vertex. In our method, the length of the query path will not be a limiting factor and this is in contrast with most previous approaches that used exponential time algorithms to find simple paths which are practical only when the paths are short. As many biological pathways are not linear and may consist of multiple interacting components, our graph matching approach allows a non-linear graph query to be given. By leveraging the observation that the query graph usually corresponds to a small functional module, our graph matching formulation overcomes the common weakness of previous approaches that there is no guarantee on the quality of the results. We notice that considerable research efforts have recently been devoted to exploring conserved pathways and functional modules in biological networks. However, these pathways or modules are often found based on sequence similarity between corresponding biological objects. The sequence similarity-oriented models emphasize the biological objects by labeling the vertices and using the similarity between them, but ignore the importance of interactions between objects by treating all edges equally. Since the models miss important components implied in biological interactions, what are found are not functionally conserved paths. For future work, it is of interest to investigate an approach that is able to find functionally similar paths or subgraphs for a given query path or query graph. One noticeable advantage is that the new approach will be able to find functionally similar pathways or modules from both the parallel evolution and the convergent evolution point of view, which are ignored by current approaches.

For the gene cluster finding problem, we investigate a formulation based on constraining the overall size of a cluster and develop statistical significance estimates that allow direct comparisons of clusters of different sizes. We prove that the problem is *NP*-hard when multiple genomes are given. We explore both a restricted version which requires that orthologous genes are strictly ordered within each cluster, and the unrestricted problem that allows paralogous genes within a genome and clusters that may not appear in every genome. We solve the first problem in polynomial time and develop practical exact algorithms for the second one. In our model, a gene is allowed to appear in more than one orthologous group and this is consistent with the biological fact that many known genes have multiple functions due to different domains. Our approach is also able to provide statistical significance estimates while allowing clusters that may not appear in every genome.

In the gene cluster querying problem, based on a querying strategy, we propose an efficient approach for investigating clustering of related genes across multiple genomes for a given gene cluster. By analyzing gene clustering in 400 bacterial genomes, we show that our algorithm is efficient enough to study gene clusters across hundreds of genomes.

As more and more genomes have been sequenced, it is desirable to have an approach that can solve the gene cluster finding problem efficiently and identify clustering of sets of homologous genes among multiple genomes. For future work, it is possible to combine our ideas for finding and querying problems of gene clusters and develop an efficient two-phase algorithm for the gene cluster finding problem as follows. In phase one, a small set of representative genomes are chosen and an existing algorithm is used to identify all common gene clusters among these chromosomes. In phase two, our gene cluster querying algorithm is used to query the clusters found above against the rest of the genomes. Since all common clusters should be found in phase one, this two-phase approach will not miss any significant results.

REFERENCES

Akutsu, T., Kuhara, S., Maruyama, O., and Miyano, S. 1998. Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In *Proc. 9th Ann. ACM-SIAM Sym. Discrete Alg.* (SODA'1998), 695–702.

Alloing, G., Trombe, M.C., and Claverys, J.P. 1990. The *ami* locus of the gram-positive bacterium *Streptococcus pneumoniae* is similar to binding protein–dependent transport operons of gram-negative bacteria. *Mol. Microbiol.* 4, 633–644.

Alon, N., Yuster, R., and Zwick, U. 1995. Color-coding. *J. ACM* 42, 844–856.

Altschul, S.F., and Erickson, B.W. 1986. Optimal sequence alignment using affine gap costs. *Bull. Math. Biol.* 48, 603–616.

Altschul, S.F., and Lipman, D.J. 1989. Trees, stars, and multiple biological sequence alignment. *SIAM J. Appl. Math.* 49, 197–209.

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. 1990. Basic local alignment search tool. *J. Mol. Biol.* 215, 403–410.

Archer, C.D., and Elliott, T. 1995. Transcriptional control of the *nuo* operon which encodes the energy-conserving NADH dehydrogenase of *Salmonella typhimurium*. *J. Bacteriol.* 177, 2335–2342.

Barker, W.C., Garavelli, J.S., Huang, H., McGarvey, P.B., Orcutt, B.C., et al. 2000. The protein information resource (PIR). *Nucleic Acids Res.* 28, 41–44.

Bergeron, A., Corteel, S., and Raffinot, M. 2002. The algorithmic of gene teams. *Lect. Notes Comp. Sci.* 2452, 464–476.

Blasco, F., Iobbi, C., Ratouchniak, J., Bonnefoy, V., and Chippaux, M. 1990. Nitrate reductases of *Escherichia coli*: sequence of the second nitrate reductase and comparison with that encoded by the *narGHJI* operon. *Mol. General Genet.* 222, 104–111.

Blattner, F.R., Plunkett, G. III, Bloch, C.A., Perna, N.T., Burland, V., et al. 1997. The complete genome sequence of *Escherichia coli* K-12. *Science* 277, 1453–1462.

Calabrese, P.P., Chakravarty, S., and Vision, T.J. 2003. Fast identification and statistical evaluation of segmental homologies in comparative maps. *Bioinformatics* 19, SI74–80.

Caldwell, R., Sapolsky, R., Weyler, W., Maile, R.R., Causey, S.C., et al. 2001. Correlation between *Bacillus subtilis scoC* phenotype and gene expression determined using microarrays for transcriptome analysis. *J. Bacteriol.* 183, 7329–7340.

Campbell, J.W., Morgan-Kiss, R.M., and Cronan, J.E. Jr. 2003. A new *Escherichia coli* metabolic competency: growth on fatty acids by a novel anaerobic β-oxidation pathway. *Mol. Microbiol.* 47, 793–805.

Carillo, H., and Lipman, D. 1988. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.* 48, 1073–1082.

Chang, C., and Stewart, R.C. 1998. The two-component system. Regulation of diverse signaling pathways in prokaryotes and eukaryotes. *Plant Physiol.* 117, 723–731.

Che, D., Li, G., Mao, F., Wu, H., and Xu, Y. 2006. Detecting uber-operons in prokaryotic genomes. *Nucleic Acids Res.* 34, 2418–2427.

Cherry, J.M., Adler, C., Ball, C., Chervitz, S.A., Dwight, S.S., et al. 1998. SGD: *Saccharomyces* Genome Database. *Nucleic Acids Res.* 26, 73–79.

Coenye, T., and Vandamme, P. 2005. Organization of the *S10*, *spc* and *alpha* ribosomal protein gene clusters in prokaryotic genomes. *FEMS Microbiol. Lett.* 242, 117–126.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. 2001. *Introduction to Algorithms*. The MIT Press, Cambridge, MA.

Cossart, P., Katinka, M., and Yaniv, M. 1981. Nucleotide sequence of the *thrB* gene of *E. coli*, and its two adjacent regions; the *thrAB* and *thrBC* junctions. *Nucleic Acids Res.* 9, 339–347.

Cotter, P.A., Chepuri, V., Gennis, R.B., and Gunsalus, R.P. 1990. Cytochrome *o* (*cyoABCDE*) and *d* (*cydAB*) oxidase gene expression in *Escherichia coli* is regulated by oxygen, pH, and the *fnr* gene product. *J. Bacteriol.* 172, 6333–6338.

Dandekar, T., Schuster, S., Snel, B., Huynen, M., and Bork, P. 1999. Pathway alignment: application to the comparative analysis of glycolytic enzymes. *Biochem. J.* 343, 115–124.

Das, A., and Ljungdahl, L.G. 2003. *Clostridium pasteurianum* F_1F_o ATP synthase: operon, composition, and some properties. *J. Bacteriol.* 185, 5527–5535.

Deane, C.M., Salwinski, L., Xenarios, I., and Eisenberg, D. 2002. Protein interactions: two methods for assessment of the reliability of high throughput observations. *Mol. Cell. Proteomics* 1, 349–356.

Deng, M., Mehta, S., Sun, F., and Chen, T. 2002. Inferring domain-domain interactions from protein-protein interactions. *Genome Res.* 12, 1540–1548.

Didier, G. 2003. Common intervals of two sequences. *Lect. Notes Comp. Sci.* 2812, 17–24.

Do, C.B., Mahabhashyam, M.S., Brudno, M., and Batzoglou, S. 2005. ProbCons: probabilistic consistency-based multiple sequence alignment. *Genome Res.* 15, 330–340.

Durand, D., and Sankoff, D. 2003. Tests for gene clustering. *J. Comp. Biol.* 10, 453–482.

Edgar, R.C. 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.* 32, 1792–1797.

Eppstein, D. 1998. Finding the k shortest paths. *SIAM J. Computing* 28, 652–673.

Eres, R., Landau, G.M., and Parida, L. 2004. Permutation pattern discovery in biosequences. *J. Comp. Biol.* 11, 1050–1060.

Feng, D., and Doolittle, R. 1987. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.* 25, 351–360.

The Flybase Consortium. 1996. FlyBase: the Drosophila database. *Nucleic Acids Res.* 24, 53–56.

Fox, B.L. 1975. k-th shortest paths and applications to the probabilistic networks. *ORSA/TIMS Joint National Meeting* 23, B263.

Fujibuchi, W., Ogata, H., Matsuda, H., and Kanehisa, M. 2000. Automatic detection of conserved gene clusters in multiple genomes by graph comparison and P-quasi grouping. *Nucleic Acids Res.* 28, 4029–4036.

Garey, M.R., and Johnson, D.S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York, NY.

Gaurivaud, P., Laigret, F., Garnier, M., and Bove, J.M. 2000. Fructose utilization and pathogenicity of *Spiroplasma citri*: characterization of the fructose operon. *Gene* 252, 61–69.

Gerike, U., Hough, D.W., Russell, N.J., Dyall-Smith, M.L., and Danson, M.J. 1998. Citrate synthase and 2-methylcitrate synthase: structural, functional and evolutionary relationships. *Microbiol.* 144, 929–935.

Gotoh, O. 1990. Consistency of optimal sequence alignments. *Bull. Math. Biol.* 52, 509–525.

Gotoh, O. 1996. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *J. Mol. Biol.* 264, 823–838.

Gusfield, D. 1993. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull. Math. Biol.* 55, 141–154.

Gustin, M.C., Albertyn, J., Alexander, M., and Davenport, K. 1998. MAP kinase pathways in the yeast *Saccharomyces cerevisiae*. *Microbiol. Mol. Biol. Rev.* 62, 1264–1300.

Hattori, M., Okuno, Y., Goto, S., and Kanehisa, M. 2003. Heuristics for chemical compound matching. *Genome Inform.* 14, 144–153.

He, X., and Goldwasser, M.H. 2005. Identifying conserved gene clusters in the presence of homology families. *J. Comp. Biol.* 12, 638–656.

Heber, S., and Stoye, J. 2001a. Algorithms for finding gene clusters. *Lect. Notes Comp. Sci.* 2149, 252–263.

Heber, S., and Stoye, J. 2001b. Finding all common intervals of k permutations. *Lect. Notes Comp. Sci.* 2089, 207–218.

Heger, A., Lappe, M., and Holm, L. 2003. Accurate detection of very sparse sequence motifs. In *Proc. 7th Ann. Int. Conf. Res. Comp. Mol. Biol.* (RECOMB), 139–147.

Hoberman, R., Sankoff, D., and Durand, D. 2004. The statistical significance of max-gap clusters. *Lect. Notes in Bioinf.* 3388, 55–71.

Huerta, A.M., Salgado, H., Thieffry, D., and Collado-Vides, J. 1998. RegulonDB: a database on transcriptional regulation in *Escherichia coli*. *Nucleic Acids Res.* 26, 55–59.

Itoh, T., Takemoto, K., Mori, H., and Gojobori, T. 1999. Evolutionary instability of operon structures disclosed by sequence comparisons of complete microbial genomes. *Mol. Biol. Evol.* 16, 332–346.

Iwanicki, A., Hinc, K., Seror, S., Węgrzyn, G., and Obuchowski, M. 2005. Transcription in the *prpC-yloQ* region in *Bacillus subtilis*. *Arch. Microbiol.* 183, 421–430.

Jiménez, V., and Marzal, A. 2003. A lazy version of Eppstein's K shortest paths algorithm. *Lect. Notes Comp. Sci.* 2647, 179–190.

Jonassen, I., Collins, J.F., and Higgins, D.G. 1995. Finding flexible patterns in unaligned protein sequences. *Protein Sci.* 4, 1587–1595.

Just, W. 2001. Computational complexity of multiple sequence alignment with SP-score. *J. Comp. Biol.* 8, 615–623.

Karlin, S., and Altschul, S.F. 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci.* 87, 2264–2268.

Karp, P.D., Riley, M., Saier, M., Paulsen, I.T., Collado-Vides, J., et al. 2002. The EcoCyc Database. *Nucleic Acids Res.* 30, 56–58.

Kasimoglu, E., Park, S.J., Malek, J., Tseng, C.P., and Gunsalus, R.P. 1996. Transcriptional regulation of the proton-translocating ATPase (*atpIBEFHAGDC*) operon of *Escherichia coli*: control by cell growth rate. *J. Bacteriol.* 178, 5563–5567.

Kececioglu, J. 1993. The maximum weight trace problem in multiple sequence alignment. *Lect. Notes Comp. Sci.* 684, 106–119.

Kececioglu, J., and Sankoff, D. 1995. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica* 13, 180–210.

Kelley, B.P., Sharan, R., Karp, R.M., Sittler, T., Root, D.E., et al. 2003. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proc. Natl. Acad. Sci. USA* 100, 11394–11399.

Kellis, M., Patterson, N., Endrizzi, M., Birren, B., and Lander, E.S. 2003. Sequencing and comparison of yeast species to identify genes and regulatory elements.

Nature 423, 241–254.

Kim, S., Choi, J.H., and Yang, J. 2005. Gene teams with relaxed proximity constraint. In *Proc. IEEE Comp. Sys. Bioinformatics Conf.* (CSB'2005), 44–55.

Knuth, D.E. 1997. *The Art of Computer Programming, Volume 1: Fundamental Algorithms.* Addison-Wesley, New York, NY.

Kobashi, N., Nishiyama, M., and Tanokura, M. 1999. Aspartate kinase-independent lysine synthesis in an extremely thermophilic bacterium, *Thermus thermophilus*: lysine is synthesized via α-aminoadipic acid not via diaminopimelic acid. *J. Bacteriol.* 181, 1713–1718.

Koebmann, B.J., Nilsson, D., Kuipers, O.P., and Jensen, P.R. 2000. The membrane-bound H^+-ATPase complex is essential for growth of *Lactococcus lactis*. *J. Bacteriol.* 182, 4738–4743.

Koide, A., and Hoch, J.A. 1994. Identification of a second oligopeptide transport system in *Bacillus subtilis* and determination of its role in sporulation. *Mol. Microbiol.* 13, 417–426.

Koyutürk, M., Grama, A., and Szpankowski, W. 2004. An efficient algorithm for detecting frequent subgraphs in biological networks. *Bioinformatics* 20, SI200–207.

Koyutürk, M., Kim, Y., Topkara, U., Subramaniam, S., Szpankowski, W., et al. 2006. Pairwise alignment of protein interaction networks. *J. Comp. Biol.* 13, 182–199.

Lane, M.C., Simms, A.N., and Mobley, H.L.T. 2007. Complex interplay between type 1 fimbrial expression and flagellum-mediated motility of uropathogenic

Escherichia coli. *J. Bacteriol.* 189, 5523–5533.

Lawler, E.L. 1972. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Sci.* 18, 401–405.

Lee, C., Grasso, C., and Sharlow, M.F. 2002. Multiple sequence alignment using partial order graphs. *Bioinformatics* 18, 452–464.

Lee, J.M., and Sonnhammer, E.L.L. 2003. Genomic gene clustering analysis of pathways in eukaryotes. *Genome Res.* 13, 875–882.

Luc, N., Risler, J.L., Bergeron, A., and Raffinot, M. 2003. Gene teams: a new formalization of gene clusters for comparative genomics. *Comp. Biol. Chem.* 27, 59–67.

Luettich, K., and Schmidt, C. 2003. TGFβ_1 activates c-Jun and Erk1 via $\alpha_{\mathrm{V}}\beta_6$ integrin. *Mol. Cancer* 2, 33.

Moreno-Hagelsieb, G., and Collado-Vides, J. 2002. A powerful non-homology method for the prediction of operons in prokaryotes. *Bioinformatics*, 18, S329–336.

Morgenstern, B., Dress, A., and Werner, T. 1996. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci. USA* 93, 12098–12103.

Morrison, D.A. 1996. Phylogenetic tree-building. *Int. J. Parasitology* 26, 589–617.

Needleman, S.B., and Wunsch, C.D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48, 443–453.

Nishida, H., Nishiyama, M., Kobashi, N., Kosuge, T., Hoshino, T., et al. 1999. A prokaryotic gene cluster involved in synthesis of lysine through the amino adipate pathway: a key to the evolution of amino acid biosynthesis. *Genome Res.* 9, 1175–1183.

Notredame, C., Higgins, D.G., and Heringa, J. 2000. T-Coffee: a novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.* 302, 205–217.

Notredame, C., Holm, L., and Higgins, D.G. 1998. COFFEE: an objective function for multiple sequence alignments. *Bioinformatics* 14, 407–422.

Ogata, H., Fujibuchi, W., Goto, S., and Kanehisa, M. 2000. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Res.* 28, 4021–4028.

O'Sullivan, O., Suhre, K., Abergel, C., Higgins, D.G., and Notredame, C. 2004. 3DCoffee: combining protein sequences and structures within multiple sequence alignments. *J. Mol. Biol.* 340, 385–395.

Overbeek, R., Fonstein, M., D'Souza, M., Pusch, G.D., and Maltsev, N. 1999. The use of gene clusters to infer functional coupling. *Proc. Natl. Acad. Sci. USA* 96, 2896–2901.

Parida, L. 2007. Gapped permutation pattern discovery for gene order comparisons. *J. Comp. Biol.* 14, 45–55.

Paris, S., Viemon, C., Curien, G., and Dumas, R. 2003. Mechanism of control of *Arabidopsis thaliana* aspartate kinase-homoserine dehydrogenase by threonine. *J. Biol. Chem.* 278, 5361–5366.

Pasek, S., Bergeron, A., Risler, J.L., Louis, A., Ollivier, E., et al. 2005. Identification of genomic features using microsyntenies of domains: domain teams. *Genome Res.* 15, 867–874.

Pevzner, P.A. 2000. *Computational Molecular Biology: an Algorithmic Approach.* The MIT Press, Cambridge, MA.

Price, M.N., Huang, K.H., Alm, E.J., and Arkin, A.P. 2005. A novel method for accurate operon predictions in all sequenced prokaryotes. *Nucleic Acids Res.* 33, 880–892.

Rahmann, S., and Klau, G.W. 2006. Integer linear programs for discovering approximate gene clusters. *Lect. Notes in Bioinf.* 4175, 298–309.

Remm, M., Storm, C.E., and Sonnhammer, E.L. 2001. Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *J. Mol. Biol.* 314, 1041–1052.

Salgado, H., Moreno-Hagelsieb, G., Smith, T.F., and Collado-Vides, J. 2000. Operons in *Escherichia coli*: genomic analyses and predictions. *Proc. Natl. Acad. Sci. USA* 97, 6652–6657.

Schmidt, T., and Stoye, J. 2004. Quadratic time algorithms for finding common intervals in two and more sequences. *Lect. Notes Comp. Sci.* 3109, 347–358.

Scott, J., Ideker, T., Karp, R.M., and Sharan, R. 2006. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Comp. Biol.* 13, 133–144.

Sharan, R., Suthram, S., Kelley, R.M., Kuhn, T., McCuine, S., et al. 2005. Conserved patterns of protein interaction in multiple species. *Proc. Natl. Acad.*

Sci. USA 102, 1974–1979.

Shazand, K., Tucker, J., Grunberg-Manago, M., Rabinowitz, J.C., and Leighton, T. 1993. Similar organization of the *nusA-infB* operon in *Bacillus subtilis* and *Escherichia coli*. *J. Bacteriol.* 175, 2880–2887.

Smith, T.F., and Waterman, M.S. 1981. Identification of common molecular subsequences. *J. Mol. Biol.* 147, 195–197.

Spirin, V., and Mirny, L.A. 2003. Protein complexes and functional modules in molecular networks. *Proc. Natl. Acad. Sci. USA* 100, 12123–12128.

Steffen, M., Petti, A., Aach, J., D'haeseleer, P., and Church, G. 2002. Automated modelling of signal transduction networks. *BMC Bioinfo.* 3, 34.

Stein, L., Sternberg, P., Durbin, R., Thierry-Mieg, J., and Spieth, J. 2001. WormBase: network access to the genome and biology of *Caenorhabditis elegans*. *Nucleic Acids Res.* 29, 82–86.

Steiner, K., and Malke, H. 2001. *relA*-independent amino acid starvation response network of *Streptococcus pyogenes*. *J. Bacteriol.* 183, 7354–7364.

Stoye, J. 1998. Multiple sequence alignment with the divide-and-conquer method. *Gene* 211, GC45–56.

Sze, S.-H., Lu, S., and Chen, J. 2004. Integrating sample-driven and pattern-driven approaches in motif finding. *Lect. Notes in Bioinf.* 3240, 438–449.

Sze, S.-H., Lu, Y., and Yang, Q. 2006. A polynomial time solvable formulation of multiple sequence alignment. *J. Comput. Biol.* 13, 309–319.

Tamames, J. 2001. Evolution of gene order conservation in prokaryotes. *Genome Biol.* 2, 0020.1–0020.11.

Tao, H., Hasona, A., Do, P.M., Ingram, L.O., and Shanmugam, K.T. 2005. Global gene expression analysis revealed an unsuspected *deo* operon under the control of molybdate sensor, ModE protein, in *Escherichia coli. Arch. Microbiol.* 184, 225–233.

Tatusov, R.L., Koonin, E.V., and Lipman, D.J. 1997. A genomic perspective on protein families. *Science* 278, 631–637.

Taylor, W.R. 1987. Multiple sequence alignment by a pairwise algorithm. *Comp. Appl. Biosci.* 3, 81–87.

Taylor, W.R. 1988. A flexible method to align large numbers of biological sequences. *J. Mol. Evol.* 28, 161–169.

Teichmann, S.A., Rison, S.C., Thornton, J.M., Riley, M., Gough, J., et al. 2001. Small-molecule metabolism: an enzyme mosaic. *Trends Biotechnol.* 19, 482–486.

Thompson, J.D., Higgins, D.G., and Gibson, T.J. 1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 22, 4673–4680.

Thompson, J.D., Plewniak, F., and Poch, O. 1999. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Res.* 27, 2682–2690.

Tohsato, Y., Matsuda, H., and Hashimoto, A. 2000. A multiple alignment algorithm for metabolic pathway analysis using enzyme hierarchy. In *Proc. 8th Int. Conf.*

Intelligent Systems Mol. Biol. (ISMB'2000), 376–383.

Uno, T., and Yagiura, M. 2000. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica* 26, 290–309.

Vandepoele, K., Saeys, Y., Simillion, C., Raes, J., and Van De Peer, Y. 2002. The automatic detection of homologous regions (ADHoRe) and its application to microcolinearity between *Arabidopsis* and rice. *Genome Res.* 12, 1792–1801.

Van Walle, I., Lasters, I., and Wyns, L. 2004. Align-m — a new algorithm for multiple alignment of highly divergent sequences. *Bioinformatics* 20, 1428–1435.

Vingron, M., and Argos, P. 1991. Motif recognition and alignment for many sequences by comparison of dot-matrices. *J. Mol. Biol.* 218, 33–43.

Voyich, J.M., Sturdevant, D.E., Braughton, K.R., Kobayashi, S.D., Lei, B., et al. 2003. Genome-wide protective response used by group A *Streptococcus* to evade destruction by human polymorphonuclear leukocytes. *Proc. Natl. Acad. Sci. USA* 100, 1996–2001.

Walt, A., and Kahn, M.L. 2002. The *fixA* and *fixB* genes are necessary for anaerobic carnitine reduction in *Escherichia coli*. *J. Bacteriol.* 184, 4044–4047.

Wang, N., Lu, S.E., Yang, Q., Sze, S.-H., and Gross, D.C. 2006. Identification of the syr-syp box in the promoter regions of genes dedicated to syringomycin and syringopeptin production by Pseudomonas syringae pv. syringae B301D. *J. Bacteriol.* 188, 160–168.

Watanabe, H., Mori, H., Itoh, T., and Gojobori, T. 1997. Genome plasticity as a paradigm of eubacteria evolution. *J. Mol. Evol.* 44, S57–64.

Wheeler, D.L., Chappey, C., Lash, A.E., Leipe, D.D., Madden, T.L., et al. 2000. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.* 28, 10–14.

Wiame, E., and Van Schaftingen, E. 2004. Fructoselysine 3-epimerase, an enzyme involved in the metabolism of the unusual Amadori compound psicoselysine in *Escherichia coli*. *Biochem. J.* 378, 1047–1052.

Wilcoxon, F. 1947. Probability tables for individual comparisons by ranking methods. *Biometrics* 3, 119–122.

Wolfe, K.H., and Shields, D.C. 1997. Molecular evidence for an ancient duplication of the entire yeast genome. *Nature* 387, 708–713.

Xenarios, I., Rice, D.W., Salwinski, L., Baron, M.K., Marcotte, E.M., et al. 2000. DIP: the database of interacting proteins. *Nucleic Acids Res.* 28, 289–291.

Yang, Q., and Sze, S.-H. 2007. Path matching and graph matching in biological networks. *J. Comput. Biol.* 14, 56–67.

VITA

Name:	Qingwu Yang
Address:	Department of Computer Science
	Texas A&M University
	College Station, TX 77843-3112
Email Address:	qyang@cs.tamu.edu
Education:	B.S. in Biochemistry and Molecular Biology,
	Peking University, China, 1996