Proceedings of the First International Workshop on Practical Aspects of Declarative Languages, San Antonio, USA, Springer Verlag, LNCS 1551, pp. 275-289

Applying Logic Programming to Derive Novel Functional Information of Genomes

Arvind K. Bansal[1]★ † and Peer Bork[2]

[1]Department of Mathematics and Computer Science
Kent State University, Kent, OH 44242, USA
Phone: +1.330.672-4004 ext. 214 E-mail: arvind@mcs.kent.edu
[2]Computer Informatics Division,
European Molecular Biology Laboratory,
Meyerhofstr 1, Postfach 10 22 09, 69012 Heidelberg, Germany
Phone: +49.6221.387.534 E-mail: Bork@EMBL-Heidelberg.de

Abstract. This paper describes an application of the logic programming paradigm to large-scale comparison of complete microbial genomes each containing four-million amino acid characters and approximately two thousand genes. We present algorithms and a Sicstus Prolog based implementation to model genome comparisons as bipartite graph matching to identify orthologs — genes across different genomes with the same function — and groups of orthologous genes — orthologous genes in close proximity, and gene duplications. The application is modular, and integrates logic programming with Unix-based programming and a Prolog based text-processing library developed during this project. The scheme has been successfully applied to compare eukaryotes such as yeast. The data generated by the software is being used by microbiologists and computational biologists to understand the regulation mechanisms and the metabolic pathways in microbial genomes.

Key-words. declarative programming, gene groups, genome comparison, logic programming, metabolic pathway, microbes, Prolog application, operons, orthologs

1 Introduction

Microbes serve as model organisms for understanding basic metabolic functions of living beings. Microbes are also important targets in biotechnology, disease treatment, and ecology. A natural step in understanding microbial genomes [1, 9] is to map the functionality and the variations in the functionality of genes and families of genes [19]. Knowledge of the functionality of genes will enhance our understanding of metabolic pathways — control flows of action among enzymes involved in high level process control in microbial organisms. Understanding the variations of metabolic pathways in two bacteria and their regulation is the key to the

† Corresponding author
★this research was supported in part by Research Council grant, Kent State University, Kent, Ohio, USA and Deutsche Forschungsgemeinschaft — The German Federal Agency

development of specific drugs to restrict the effects of harmful bacteria, and to enhance the effects of useful bacteria.

It is anticipated that the complete human genome will be fully sequenced by the year 2003. The development of mapping and comparison techniques will provide a basis to develop tools to understand human genome components: the functionality of genes and an understanding of metabolic pathways.

The genes inside a metabolic pathway are often clustered together in gene-groups called *operons*. The constituting genes of an operon are usually copied together (co-transcribed), and may have a common control region which facilitates their co-transcription. The identification of operons facilitates the identification of metabolic pathways using similarity analysis**.**

Currently, the functionality of many of the genes is available experimentally from wet laboratories. However, the function of genes and gene-groups of newly sequenced genomes is unavailable. Due to the size of genomes — an average microbial genome has 0.6 million – 4.7 million amino-acids and 472 – 4200 genes — and the increasing rate of availability of complete genomes, a cost-effective and time-efficient way is to use automated pair-wise comparison of genome sequences to identify functionally equivalent genes in newly sequenced genomes.

One technique to understand the functionality of genes in newly sequenced genomes is to identify *orthologs* — genes which have same function and a common evolutionary ancestor. However, It is difficult to ascertain common evolutionary ancestors due to the divergence and the convergence of species during evolution [8]. One heuristic is to identify two genes in evolutionarily close genomes [13, 18], represented as amino-acid sequences, with the highest similarity using pair-wise genome comparison. These most similar genes are the best candidates for being orthologs. We refer to such pair of genes as *putative orthologs*. These putative orthologs can be further confirmed by automated matching of secondary structure of amino-acid sequences and automated matching of the biochemical properties of different parts of the genes, and finally by wet lab techniques if needed.

Orthologous genes in close proximity are strong candidates to be involved together in a common function. Two gene-groups are *putative orthologous* — candidates for being involved together in a common function — if the constituting genes in the gene-groups are putative orthologs. These putative orthologous gene-groups may include one or more operons, and are natural starting points to identify operons and metabolic pathways.

The first step is to identify *homologs* — similar sequences identified using approximate string matching. Homologs include both *putative orthologs* and *paralogs* — duplicated genes with different functions [8]. Paralogs are pruned out from homologs leaving putative orthologs.

In this paper, we use previously developed software to identify [3, 11] and align homologs [12, 16, 21], and develop graph matching techniques to identify putative orthologs, homologous gene-groups, duplicated gene-groups, and putative orthologous gene-groups. We have modeled each genome as an ordered set of genes, modeled the matching of a pair of complete genomes as a bipartite graph matching problem, developed algorithms [7] to identify homologous gene-groups, and developed a variant of the stable marriage algorithm [14] to identify pairs of genes

with the highest similarity scores. The bipartite graph has 2000 to 4000 nodes in each set and approximately 12000 edges.

The problem required the integration of text processing, approximate string matching and dynamic programming techniques for gene comparison (already available in C), as well as various techniques such as heuristic reasoning, high level modeling using graphs and trees, and a high level interface to system programming. The problem is suitable for logic programming because:

1. The software must be experimental and continuously evolving since the output data are continuously providing new insights into the genomic structure and function. Many pieces of information [7] previously unknown are being identified and investigated by the microbiology community [20].
2. Prolog facilitates easy integration of heuristic reasoning, text processing and manipulation of complex data structures such as graphs and associative tables.

Large scale iteration has been simulated using tail recursive programming. The tool was developed using Sicstus Prolog version 3.5 which has a library to interface with Unix-based systems and a library to manipulate graphs and associations. Currently the software takes around 180 minutes to compare and derive information for a pair of microbial genomes — each having approximately 4000 genes and around four million amino-acid characters — on a four processor SGI with 195 Mhz processors. The majority of time is taken by BLAST comparisons [3, 11] and the Smith-Waterman gene alignment [12, 16, 21]) to align the homolog-pairs. The detection of functional information about the genome (described in this paper) takes less than 100 seconds of execution for a bipartite graph consisting of approximately 4200 nodes in each set and approximately 12000 edges.

The claim of this paper is that this state-of-the-art Prolog software has automated the innovative analysis of genome-data to identify the functionality of micro-organisms [7]. The software identifies gene-groups, orthologs, gene-fusions — two genes fusing together to form a new gene, gene-duplications — single genes having multiple occurrences in another genome, and gene-group duplications — groups of genes in one genome having multiple occurrences in another genome. Such detailed information was not previously available to the biological community. The data generated by an extended version of this software[1] is being used by researchers to study gene regulation [20], and derive metabolic pathways [6]. The software executes in a realistic time for a computationally intensive problem and can be modified easily as new information about genome organization is discovered.

The paper is organized as follows: Section 2 describes the concepts and related software; Section 3 describes the overall scheme; Section 4 describes the integration of Unix-based programming and text processing to extract gene-related information from Genbank at NCBI, invocation of Unix shells for BLAST comparisons, and invocation of Unix shells for gene-pair alignment; Section 5 briefly describes the algorithm [7] for identifying homologous gene-groups and the related Prolog implementation; Section 6 describes an algorithm for identifying orthologs and the related Prolog implementation; and Section 7 briefly describes the related works, and the last section concludes the paper.

[1] The orthologs were confirmed by a variation of the Hungarian method for bipartite graph matching developed by Peter Stuckey [7].

2 Background

The author assumes a basic familiarity with Prolog [2, 17]. In this section, we describe some genome related concepts [1] needed to understand the paper, and a brief description of bipartite graphs needed to model genome comparison.

2.1 Genome Related Background

The genome of an organism is encoded within molecules of DNA [1, 16]. A molecule of DNA is a sequence composed of four nucleotides represented as 'A', 'C', 'G', 'T'. A protein is a sequence of twenty amino acids represented as alphabets of English language.

A *microbial genome* is modeled as an ordered set of pairs of the form $<(c_1, \gamma_1), \ldots, (c_N, \gamma_N)>$. Each γ_I is a sequence of the form $<s_1, ..., s_N>$ ($1 \leq I \leq N$) where s_I is a nucleotide for DNA and an amino-acid for a protein. Each c_I is a control region preceding γ_i. In this paper, we are interested in the comparison of protein sequences of the form $<\gamma_{11}, \ldots, \gamma_{1N}>$ and $<\gamma_{21}, \ldots, \gamma_{2M}>$. In this paper, we will denote a complete genome by the subscripted Greek letter Γ_I and individual gene by $\gamma_{I.}$ and a gene-pair by $(\gamma_{1I}, \gamma_{2J})$ where the first subscript denotes the genome number, and the second subscript denotes the sequential order of a gene within a genome.

Two gene-sequences are similar, if there is a significant match between them after limited shifting of characters. *Sequence alignment* arranges similar sequences in a way that asserts a correspondence between characters that are thought to derive from a common ancestral sequence. Aligning a set of sequences requires the introduction of spacing characters referred to as *indels*. If the aligned sequences did indeed derive from a common ancestral sequence, then indels represent possible evolutionary events. While aligning two sequences, a two dimensional matrix (PAM matrix) describing similarity between amino-acids is used [16, 21]: amino-acids with similar biochemical properties are better suited for identifying the similarity of a protein.

Example 1.1 Let us consider two amino-acid sequences 'AGPIAL' and 'APVV'. The amino-acids 'I,' 'L', and 'V' are very similar and belong to a common category: hydrophobic amino-acids. Thus matching the amino-acid 'I' with 'V' or matching the amino-acid 'I' with 'V' is preferred. A possible alignment is given below:

A G P I A L
A _ P V _ V

2.1 Sequence Comparison Techniques

The BLAST software [3, 11] — a popular technique to search for homologs — selects a small subsequence, using string matching to identify locations of matches, and expands the size of matched segments at the identified locations using finite state automata and approximate character matching using a PAM matrix [9].

The Smith-Waterman algorithm [12, 16, 21] is a matrix-based dynamic programming technique to align two sequences. The Smith-Waterman alignment is based upon iterative addition of the next position to a pair of best matching subsequences. There are three possibilities: the next position in the first sequence

matches an indel, the next position in the second sequence matches an indel, or two non-indel characters match. The algorithm uses the function:

similarity-score(A[1..I], B[1..J]) = max(
similarity_score(A[1..I-1], B[1..J]) + penalty(a_I, '_')),
similarity_score(A[1..I], B[1 .. J - 1]) + penalty('_', b_J)),
similarity_score(A[1..I - 1], B[1..J -1]) + match(a_I, b_J)))

where A[1..K] or B[1..K] is a subsequence of the length K starting from the beginning position, and a_I or b_J is an element in the sequence. The Smith-Waterman algorithm is more precise than BLAST. However, BLAST is more efficient than the Smith-Waterman algorithm.

2.2 Functional Genomics Related Concepts

Homologs are similar genes derived from similarity search techniques such as BLAST [3, 11]. *Paralogs* are homologous genes resulting from gene duplication, and have variations in functionality. A *putative ortholog* is a homolog with the highest similarity in pair-wise comparison of genomes.

A *gene-group* is a cluster of neighboring genes $<\gamma_I\ \gamma_J\ \gamma_K \ldots>$ ($I < J < I + r$, $J < K < J + r$, where $r > 0$). A gene-group may have insertions, permutations, or deletions of genes with reference to a corresponding gene-group in another genome. A *homologous gene-group* $<\gamma_{1I}\ \gamma_{1J}\ \gamma_{1K} \ldots>$ ($I < J < K$) in the genome Γ_1 matches with the corresponding gene-group $<\gamma_{2M}\gamma_{2N}\gamma_{2P} \ldots>$ ($M < N < P$) in Γ_2 such that γ_{1I} and γ_{1J} and γ_{1K} etc. are similar to one of the genes in the sequence $<\gamma_{2M}\gamma_{2N}\gamma_{2P} \ldots>$. A *shuffled gene* is an ortholog of a gene in a gene-group such that the ortholog lies outside the putative orthologous gene-group. *Gene-gaps* are genes in one genome without any orthologous correspondence in another genome. A *fused gene* in a genome has two orthologs which fuse together to form a single gene.

A genome is modeled as an *ordered set of genes*. Pair-wise genome comparison is modeled as a weighted bipartite graph such that genes are modeled as vertices, two homologous genes in different sets have edges between them, and the similarity score between two genes is the weight of the corresponding edge.

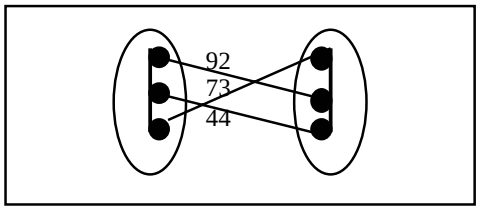

Fig. 1. Genome comparison as bipartite graph matching problem

3 A Scheme to Identify Putative Orthologs and Gene-groups

In this section, we describe an overall scheme for identifying functionally equivalent genes and gene groups. The identification involves six stages: a protein preparation stage, a homolog detection stage, an alignment stage, a gene-group identification stage, and an ortholog detection stage. The output after each stage is in the form of a text file. The advantages of representing the information in readable textual form are:

1. The development of each stage is modular and independent of others.
2. The output after each stage is in human comprehensible form for further analysis. The software development is incremental, and is dependent upon the feedback of microbiologists and computational biologists after each stage. The textual form of representation facilitates this development.

Some major features of the text-processing software are: skipping white characters, removing white characters, skipping text until a marker text is identified, splitting a line into various words, splitting an annotated word into smaller information units, splitting a string into two substrings after identifying a pattern, and converting strings into numbers or atoms and vica-versa.

The first stage prepares the data for comparison of two genomes. This stage extracts protein sequences from annotated text files given in Genbank (*ftp://ncbi.nlm.nih.gov/genbank/genomes/bacteria/*) and requires text-processing capability (coded in Sicstus Prolog). The software extracts the information about the *name of a gene, starting point and ending point of the protein-coding region*, *length of a gene*, and *the DNA-strand* used to extract the amino-acid sequence of the gene.

The second stage uses BLAST [11] to identify a set of homolog-pairs. The output of this stage is of the form (γ_{1I}, γ_{2J}, *similarity-score*) : gene γ_{1I} occurs in the first genome, the gene γ_{2J} is a homolog of γ_{1I} in the second genome, and *similarity-score* describes the amount of similarity between two genes. This method improves the execution efficiency by pruning edges with low similarity score.

The third stage aligns two homologous genes using the Smith-Waterman alignment software [12]. The output of this stage is of the form (γ_{1I}, γ_{2J}, *alignment-score, identity-score*, *beginning shift*, *end-shift*, *size of largest segment of continuous indels*).

The fourth stage models the pair of genomes as a weighted bipartite graph with each genome forming an ordered set. Putative groups of homologous genes are identified by traversing all the genes in the genomes and identifying the clusters of edges whose adjacent vertices in both the sets in the bipartite graph are within a predetermined range. As the new vertices are selected the range keeps expanding since it is centered around the current vertex. The gene-group breaks when there is at least one source-vertex which has all the edges ending in sink vertices out of the proximity of the previously matching vertices.

The fifth stage uses the bipartite graph to identify the best possible edge connecting two nodes. These nodes with best matches are marked as orthologs. Empirical data and the study of enzyme nomenclatures, other secondary structure prediction methods [20], and other bipartite graph matching techniques [7] have suggested that the scheme is a good heuristic for identifying the putative orthologs.

The sixth stage identifies the orthologous gene-groups by combining the knowledge of homologous gene groups and orthologs. Many of these groups may be

operons or a group of operons involved in metabolic pathways. This stage also identifies shuffled genes, gene-gaps, and fused genes.

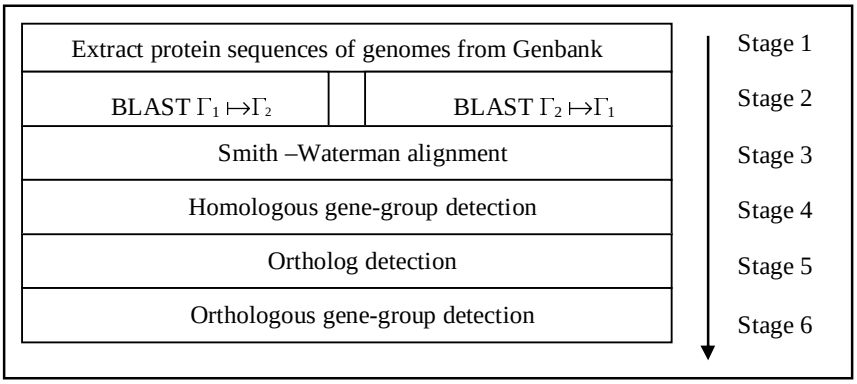

Fig 2. A complete schematics to identify orthologs and orthologous gene-groups

4 Implementing Stages I, II, and III

In this section, we briefly describe the implementations of Stage I — extracting genome information from Genbank, Stage II — identifying homologous genes using BLAST [11], and Stage III — alignment of homologous gene-pairs using the Smith-Waterman software [16].

4.1 Stage 1: Extracting Genome Information from Genbank

The amino acid sequences given in the Genbank (*ftp://ncbi.nlm.nih.gov/genbank/ genomes/ bacteria/*) in the GBK format are extracted and processed using the text-processing library developed in Prolog. The software generates two text files: a file containing an ordered set of protein coding regions of genes in FASTA format, and a file containing the ordered set of 5-tuples (*gene-name, index of the gene, location of first nucleotide of a gene, location of the last nucleotide of a gene, the direction of the gene*). The information in the second file is later used to identify and analyze the control regions of orthologs.

4.2 Stage II: Identifying Homologs by BLAST Comparisons

In the case of the BLAST search, one of the genomes is converted as a database of genes, and every gene in the second genome is matched (using a Bourne shell) against the database for similarity. The code to invoke BLAST comparison is illustrated in Figure 3.

The predicate *homologs/4* takes one of the genomes as an input stream *GeneStrm*, and the other genome as a database *Database* created by a special blast command.

The predicate *homologs/4* tail-recursively extracts the next gene sequence in a temporary file *Gene* using the predicate *next_gene/2*, invokes the predicate *blastp/5* to identify homolog-pairs, and deletes the temporary file *Gene* after collecting the homologs for the gene.

```
homologs(GeneStream, Database, Threshold, OutStream):-
  (next_gene(GeneStream, Gene) ->
     blastp(Gene, Database, Threshold, OutStream),
     homologs(GeneStream, Database, Threshold, OutStream)
     delete_file(Gene)
   ;otherwise -> true).

blastp(Gene, Database, Threshold, OutStream):-
  temp_file(Result),
  shell_cmnd([blastp, Gene, DataBase, '>', Result]),
  filter_high_pairs(Result, Threshold, OutStream),
  delete_file(Result).
```

Fig. 2. Identifying homologs by invoking BLAST using Bourne shell

The predicate *blastp/*5 creates a temporary file *Result*, invokes a Bourne shell using the predicate *shell_cmnd/1* which writes the matching gene-pairs with similarity scores in the temporary file *Result*, filters out the gene-pairs above a user-defined threshold value *Threshold* using the procedure *filter_high_pairs/3*, and finally deletes the file *Result*.

The predicate *shell_cmnd/1* fuses the elements of the list into one Unix command, and uses the built-in predicate *shell/1* to invoke the command '*blastp GeneFile DataBase > Result*'. The predicate *temp_file/1* creates a random file previously absent in the current subdirectory.

4.3 Stage III: Aligning Homolog-pairs

For the gene-alignment stage, every homolog-pair is aligned using separate Bourne shells. The shell invocation is given in Figure 4.

The predicate *align_genome/4* takes as inputs *HomologStrm* — the stream of homolog-pairs produced by the BLAST comparisons, *HomologAssoc* — an association table of attributes of homolog-pairs, *Parameter* — a set of parameters needed by the alignment command *mlocalS* [16], and outputs a similarity score and other alignment related attributes. The predicate *next_homolog_pair/2* extracts the next homolog-pair from the stream *HomologStrm*. The predicate *get_assoc/3* extracts the homolog-related attributes *HomologAttrs* from the association table *HomologAssoc*. The predicate *alignment/3* produces alignment related attributes *AlignAttrs* of a homolog-pair. The predicate *append/2* appends the gene names and homolog related attributes and alignment related attributes. The predicate *write_score/2* writes the information in the output stream *OutStrm*.

The predicate *alignment/3* accepts a homolog-pair (*Gene1, Gene2)* and a set of parameters *Parameter*, and outputs a similarity score in the file *ScoreFile* after the alignment of the homolog-pair. The predicate *temp_file/1* creates a temporary file *ScoreFile* where the similarity score is written by the Smith-Waterman software in a textual form. The predicate *similarity_score/4* extracts the similarity score and other

alignment-related attributes from the file *ScoreFile* using text processing. The predicate *delete_file/1* deletes the file *ScoreFile*.

The Smith-Waterman software [12] invokes a command '*mlocalS GeneFile1 GeneFile2 Matrix GapPenalty MultipleGapPenalty > Scorefile'*. The gene sequences are accepted in files *GeneFile1* and *GeneFile2*, and similarity score and other alignment related attributes are written in the file *ScoreFile*. The names of the files created for each gene is the same as gene-names. The predicate *absolute_file/2* takes a gene name, and returns an absolute file.

```
align_genome(HomologStrm, HomologAssoc, Parameter, OutStrm):-
  (next_homolog_pair(HomologStrm, Source-Sink) ->
   get_assoc(Source, HomologAssoc, HomologAttrs),
   alignment((Source, Sink), Parameter, AlignAttrs),
   append([[Source, Sink], HomologAttrs, AlignAttrs], Line),
   write_scores(OutStrm, Line),
   align_genome(HomologStrm, HomologAssoc, Parameter, OutStrm)
  ;otherwise -> true).

alignment(HomologPair, Parameter, AlignAttrs):-
  HomologPair = (Gene1, Gene2),
  temp_file(ScoreFile),
  align_gene_pair(Gene1, Gene2, Parameter, ScoreFile),
  similarity_score(ScoreFile, AlignAttrs),
  delete_file(ScoreFile).

align_gene_pair(GeneName1, GeneName2, Parameter, OutFile):-
  Parameter = (Matrix, Offset, GapPenalty, MultiGapPenalty),
  absolute_file(GeneName1, GeneFile1),
  absolute_file(GeneName2, GeneFile2),
  shell_cmnd([mlocalS, GeneFile1, GeneFile2, Matrix, Offset,
           GapPenalty, MultiGapPenalty, '>', OutFile]).
```

Fig. 3. Code to invoke a shell for the Smith-Waterman gene-pair alignment

5 Identification of Homologous Gene-groups

In this section, we describe an algorithm to identify homologous gene-groups in two genomes, and a Prolog implementation for the corresponding algorithm. This algorithm is based upon identifying islands of edges such that sets of sources and sinks in the bipartite graph are in proximity in the corresponding ordered sets. The implementation models the bipartite graph, and then identifies the islands of edges in proximity.

5.1 Modeling Genome Comparison as a Bipartite Graph Matching Problem

A weighted bipartite graph is modeled as an associative table of pairs of the form (source-node, value). The use of an associative table facilitates random access of the source genes and the corresponding sinks and the related attributes during the bipartite graph matching. The value is a list of 5-tuples of the form (*sink node, weight of the edge given by the similarity score, gene-strand information, length of the gene, length of the aligned portion of the gene).*

The code for modeling the pair-wise genome comparison as bipartite graph is given in Figure 5. The predicate *edges/2* collects all the gene-pairs from the output of the gene alignment stage. The predicate *source_vertices/2* collects all the source nodes from the gene-pairs. The predicate *group_edges_by_source/3* groups the edges as a list of pairs (*source-node, the list of edges and related attributes*), and forms an association out of the list to facilitate efficient access of the edges and their attributes.

```
bipartite_graph(Stream, BipartiteAssoc) :-
  edges(Stream, Edges), source_vertices(Edges, Nodes),
  group_edges_by_source(Nodes, Edges, Graph),
  list_to_assoc(Graph, BipartiteAssoc),
  group_edges_by_source([ ], _, [ ]).

group_edges_by_source([Src|Ss], Edges, Graph) :-
  group_edges_for_node(Edges, Src, RemainingEdges, SubGraph),
  Graph = [Src-SubGraph|Gs],
  group_edges_by_source(Ss, RemainingEdges, Gs).

group_edges_for_node([ ], _, [ ], [ ]).
group_edges_for_node([Source-SinkAttributes|Es], Node,
                     RemainingEdges, SubGraph):-
  (Source == Node ->
    SubGraph = [SinkAttributes|Ss],
    group_edges_for_node(Es, Node, RemainingEdges, Ss)
  ;otherwise ->
    RemainingEdges = [Source-SinkAttributes|Es],
    SubGraph = [ ]).

edges(Stream, ListofEdges) :-
  skip_white_spaces(Stream, NextChar),
  is_end_of_file(NextChar) -> ListofEdges = [ ]
  ;otherwise -> next_line(Stream, Line),
    weighted_edge(Line, WeightedEdge),
    ListofEdges = [WeightedEdge|Es],
    edges(Stream, Es)).
```

Fig. 4. Modeling genome comparison as bipartite graph matching

5.2 Identifying Islands of Clustered Edges for Homologous Gene-groups

In the following algorithm, we denote a set of elements in the I_{th} field in a set of m-tuples S by $\Pi_I(S)$. Given a set of gene-pairs S, the set of source-vertices will be denoted as $\Pi_1(S)$, and the set of sink vertices will be denoted as $\Pi_2(S)$.

The technique uses a sliding window of size b ($b \geq 1$) to identify the cluster of edges in close proximity. For any edge (γ_{1I} , γ_{2J}), the process is repeated if another matching ($\gamma_{1(I+r)}$, $\gamma_{2(J \pm s)}$) ($0 < r, s < b$) is found. The next search is performed in the range ($J - b$, $J + s + b$) for the matching ($\gamma_{1(I+r)}$, $\gamma_{2(J+s)}$), in the range ($J - s - b$, $J + b$) for the matching ($\gamma_{1(I+r)}$, $\gamma_{2(J-s)}$). Since the window keeps sliding, variably-sized gene-groups are identified.

Let the set of nodes in Γ_1 (or Γ_2) be S_1, and the set of unprocessed weighted edges be S_2. Every time an edge (γ_{1I}, γ_{2J}) $\in$ set of remaining edges is picked such that γ_{1I} has the least value of I. S_3 — the neighborhood set of γ_{1I} — is $\{\gamma_{1(I-b)}, \ldots, \gamma_{1(I+b)}\}$ such that $0 \leq I < b$ and $I + b \leq$ *number of genes in* Γ_1. Let the set of all the nodes in Γ_2

which match with γ_{1I} be S_4. Let the set of nodes in S_4 which share edges with the nodes in the neighborhood set S_3 be S_5. The set S_5 is identified by selecting the nodes in S_4, one at a time, and taking the intersection of the set of nodes in Γ_1 connected to the node and the set S_3. If the intersection is non-empty then that node in S_4 is included in S_5. If the set S_5 is non-empty then there is at least one possibly matching gene-group. After detecting the presence of a homologous gene-group, the nodes in Γ_1 are traversed from the node γ_{1I}, and the putative gene-groups are collected.

To facilitate the dynamic alteration of neighboring nodes, a copy of the sets S_3 and S_5 is made in the sets S_7 and S_9 respectively. A set S_8 is used to verify matching edges incident upon the nodes in S_3 during the collection of variably sized gene-groups. If there is a matching node within the range $\gamma_{1(K+r)\,(0 \le r < b,\ K < I)} \in S_7$ which matches one of the nodes $\gamma_{2L} \in S_9$, then there is a group. The set of edges in a neighborhood is collected in S_6; S_7 — the neighborhood set of $\gamma_{1K\ (K \ge I)} \in S_7$ — is extended dynamically to include the neighbors of $\gamma_{1(K+r)}$; S_9 — the neighborhood set of matching nodes in Γ_2 — is extended to include the neighbors of γ_{2L}; and all the edges incident on $\gamma_{1(K+r)\,(0 \le r < b \text{ and } K \ge I)}$ are included dynamically in the set S_8. Those edges which are traversed once are deleted from S_8, and those nodes in Γ_2 which have been traversed once are deleted from the sets S_5 and S_9. After a matching $(\gamma_{1(K+r)} \in S_7) \mapsto (\gamma_{2L} \in S_9)$ is not found in the neighborhood, S_6 — the current collected set of edges — is closed; $\Pi_1(S_6)$ — the set of source-nodes in S_6 — gives a putative corresponding gene-group in Γ_1, and $\Pi_2(S_6)$ — the set of sink-nodes in S_6 — gives the putative homologous gene-group in Γ_2. The set of nodes in Γ_2 which have been traversed during the identification of the last group is deleted from the set S_9, and the set of traversed edges is deleted from the set S_8.

The process is repeated to identify the next homologous gene-group which starts with γ_{1I}. This is possible since there are duplicates of gene-groups. After finding out all the homologous gene-groups involving γ_{1I}, the set of edges incident upon γ_{1I} is deleted to avoid reconsideration. The process is repeated by picking up the next edge in S_2 which has the minimum index, until S_2 is empty.

5.3 Implementing the Algorithm

Figure 6 illustrates the top level code for the implementation of the algorithm. The predicate *gene_groups/3* picks up all the edges starting from a gene in the variable *EdgesfromSource*, extracts the corresponding sink vertices using the predicate *sink_vertices/2*, and identifies the list of genes in the proximity which have neighbors using the predicate *have_neighbors/4*, and then picks up all the groups associated with the neighboring genes *HavNbrs* using the predicate *islands/6*.

The predicate *island/6* picks up the groups *Groups* corresponding to one source node *Node* using the predicate *one_group/5*, deletes the corresponding edges incident upon the source node *Node* from the graph *Graph* using the predicate *delete_edges/3*, and repeats the process with the remaining nodes *Gs* and the remaining subgraph *Rest*.

```
gene_groups([ ], _, [ ]).
gene_groups(Graph, Boundary, GeneGroups):-
  Graph = [EdgesfromSource|Gs],
  EdgesfromSource = Source-SinksandAttributes,
```

```
   sink_vertices(SinksandAttributes, Sinks),
   (have_neighbors(Source-Sinks, Boundary, Graph, HavNbrs)->
      Prox is Boundary - 1,
      islands(HavNbrs, Boundary, Prox, Graph, Group, Rest),
      GeneGroups = [Group|Rs],
      gene_groups(Rest, Boundary, Rs)
   ;otherwise -> gene_groups(Gs, Boundary, GeneGroups)).

islands([ ], _, _, Graph, [ ], Graph).
islands([Node|Gs], Boundary, Prox, Graph, Groups, New):-
   one_group(Node, Boundary, Prox, Graph, Group),
   delete_edges(Group, Graph, Rest),
   (has_more_than_two_edges(Group) ->
      Groups = [Group|Ns],
      islands(Gs, Boundary, Prox, Rest, Ns, New)
   ;otherwise ->
      islands(Gs, Boundary, Prox, Rest, Groups, New)).
```

Fig. 5. Top level code to identify homologous gene-groups

6 Identification of Putative Orthologs

The technique sorts edges in the descending order of weight, and marks the node-pairs with the highest weights as putative orthologs, and all the edges incident upon the identified putative orthologs are pruned. The process is repeated until gene-pairs are consumed. The genes inside homologous gene-groups are positively biased since genes inside a gene-group are more likely to retain the function of an operon. In case there are two edges from a node with a very high score (above a threshold value), both are treated as putative orthologs. If there are two edges whose similarity score is below the threshold and both the scores are close, then both homologs are marked as conflicting orthologs. The conflicting orthologs are resolved by structure analysis or metabolic path analysis, and are outside the scope of this paper.

6.1 Implementing the Ortholog Detection

The top-level code for ortholog detection is given in Figure 7. The predicate *ortholog/3* accepts a list of gene groups in the variable *GeneGroups*, the bipartite graph association in the variable *BipartiteAssoc*, and derives the orthologs in the variable *Orthologs*. The homologous genes inside the gene-groups are positively biased by a bias factor *Factor* using the predicate *bias_grouped_edges/3*. The predicate *biased_edges/3* picks up non-grouped weighted edges from the association *BipartiteAssoc*, and picks up the weighted biased edges from the list *BiasedEdges*. The joint list of the weighted edges *WeightedEdges* is sorted in the descending order by weight using the predicate *descending_sort/2*. The resulting list of edges *SortedEdges* is processed by the predicate *stable_marriage/2* to derive the homolog-pairs with the highest similarity.

The predicate *stable_marriage/2* invokes predicates *best_match/4* and *filter_well_separated/4*. The predicate *best_match/4* identifies edges with the highest weight, checks whether any previously processed edge already used any node in the current node-pair, and checks whether the difference in the weights of the previously

processed edge and the current edge is significant. The predicate *filter_well_separated/2* filters out the clean orthologs.

```
orthologs(GeneGroups, BipartiteAssoc, Orthologs):-
  flatten_gene_groups(GeneGroups, GroupedHomologs),
  '$bias_factor'(Factor),
  bias_grouped_edges(GroupedHomologs, Factor, BiasedEdges),
  biased_edges(BiasedEdges, BipartiteAssoc, WEdges),
  descending_sort(WEdges, SortedEdges),
  stable_marriage(SortedEdges, UnsortedOrthologs),
  sort_by_gene_index(UnsortedOrthologs, Orthologs).

stable_marriage(Edges, Orthologs):-
  list_to_assoc([ ], SrcAssoc),
  list_to_assoc([ ], SinkAssoc),
  best_match(Edges, SrcAssoc, SinkAssoc, MarkedPairs),
  filter_well_separated(MarkedPairs, Orthologs).

best_match([ ], SrcAssoc, _, SrcAssoc).
best_match([WEdge|Cs], SrcAssoc, SinkAssoc, Marker):-
  WEdge = Weight-Src-Sink-SrcSt-SrcEnd-SinkSt-SinkEnd,
  get_value(Src-SrcSt-SrcEnd, SrcAssoc, Weight1),
  get_value(Sink-SinkSt-SinkEnd, SinkAssoc, Weight2),
  ((are_separated(Weight, Weight1),
    are_separated(Weight, Weight2)) ->
    I1 = Sink-Weight-SrcSt-SrcEnd-SinkSt-SinkEnd-separated
    I2 = Src-Weight-SinkSt-SinkEnd-SrcSt-SrcEnd-separated
    insert_first(Src, I1, SrcAssoc, NewSrcAssoc),
    insert_first(Sink, I2, SinkAssoc, NewSinkAssoc)
  ;otherwise ->
    I1 = Sink-Weight-SrcSt-SrcEnd-SinkSt-SinkEnd-conflict,
    I2 = Src-Weight-SinkSt-SinkEnd-SrcSt-SrcEnd-conflict,
    insert_last(Src, I1, SrcAssoc, NewSrcAssoc),
    insert_last(Sink, I2, SinkAssoc, NewSinkAssoc)),
  best_match(Cs, NewSrcAssoc, NewSinkAssoc, Marker).
```

Fig. 6. Top level code to identify orthologs

Two association tables are kept in the predicate *best_match/4* to check the presence of a previously processed edge and to compare the weights of previously processed edge and the current edge. The first association *SourceAssoc* stores a double-ended queue as the value of the source-vertex in the bipartite graph. The second association table *SinkAssoc* stores a double ended queue as the value of the sink-vertex in the bipartite graph. If the difference in the weights of the previously processed edge and the current edge is significant then the current edge is discarded. First edge is inserted in front of the associations *SourceAssoc* and *SinkAssoc*. If the difference in the weight of previously processed edge and the current edge is insignificant, then both edges are marked *conflicting*. Conflicting edges are resolved by biological reasoning. The predicate *get_value/3* gets the weight from the association for comparison with the current weight. The predicate *are_separated/2* checks whether the difference between weights of the previously processed edge and the current edge is significant. The predicate *insert_last/4* inserts an edge and its attributes at the end of the deque. The predicate *insert_first/4* inserts an edge and its attributes in the front of the deque.

7 Related Works

There are two types of related works: ortholog detection [19] and application of logic programming to genome related problems.

The work on ortholog detection [19], done independently and concurrently to our work, identifies clusters of orthologs using pair-wise comparison of all the proteins in a limited set of multiple genomes using *semi-automated* software and techniques. Our contribution over this work is that we have developed algorithms and implemented software for the completely automated identification of orthologs, gene-groups, shuffled genes. In addition, we are also able to identify gene fusions and duplicated gene-groups.

Logic programming has been applied to develop logical databases to retrieve information about metabolic pathways [10], to identify and model genome structure [4, 15], and to identify constrained portion of genes by integrating the information of phylogenetic tree and multiple sequence alignments [5]. All these works are significant for different aspects of genome modeling and analysis. However, this application performs large scale comparison of complete genomes to identify gene functions and other novel information significant for automated identification of operons and and function of genes in newly sequenced genomes.

8 Future Works and Conclusion

We have described a novel application of logic programming to derive new functional information about microbial genomes. Logic programming is suited since the software has to be continuously modified to incorporate new changes based upon the analysis of the previous output. Logic programming integrates string processing, text processing, system-based programming, and heuristic reasoning. Currently the software is being modified to understand the constituent domains within genes, regulation mechanisms within genes [20], and to derive the metabolic pathways automatically [6].

Acknowledgements

The first author acknowledges lively discussions with the colleagues at EMBL, especially with Chris Sander and his group during Fall 1995. Warren Gish provided the latest version of BLAST [11]. Paul Hardy provided the portable library of the Smith-Waterman alignment software [12]. Peter Stuckey confirmed the results using a variant of the Hungarian method [7].

References

[1] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J. D.: Molecular Biology of THE CELL, Garland Publishing Inc. 1983

[2] Almgren, J., Anderson J., Anderson S., et. al.: Sicstus 3 Prolog Manual. Swedish Institute of Computer Science, 1995
[3] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., Lipman, D. J: Basic Alignment Search Tools, J. Mol. Biol., vol. 215, (1991) 403 - 410
[4] Baby, O. and : Non-Deterministic, Constraint-Based Parsing of Human Gene, Ph. D. thesis, Brandeis University, USA, (http://www.cs.brandeis.edu/~obaby/ phdthesis.html)
[5] Bansal, A. K.: Establishing a Framework for Comparative Analysis of Genome Sequences. Proceedings of the International Symposium of Intelligence in Neural and Biological Systems, Herndon VA USA (1995) 84 – 91
[6] Bansal, A. K.: Automated Reconstruction of Metabolic Pathway of Newly Sequenced Microbial Genomes using Ortholog Analysis. to be submitted
[7] Bansal, A. K., Bork, P., and Stuckey P.: Automated Pair-wise Comparisons of Microbial Genomes. Mathematical Modeling and Scientific Computing, Vol. 9 Issue 1 (1998) 1 – 23
[8] Fitch, W. M.: Distinguishing Homologous from Analogous Proteins. Systematic Zoology, (1970) 99 - 113
[9] Fleischmann, R. D., Adams, M. D., White O., et. al.: Whole-Genome Random Sequencing and Assembly of Haemophilus influenzae Rd. Science 269 (1995) 496 - 512
[10] Gaasterland, T., Maltsev, N., and Overbeek, R.: The Role of Integrated Databases In Microbial Genome Sequence Analysis and Metabolic Reconstruction. In: Proceedings of the Second International Meeting on Integration of Molecular Biological Databases, Cambridge, England, July 1995.
[11] Gish, W.: WU-BLAST version 2.0. Washington University, St. Louis MO USA, (http://blast.wustl.edu)
[12] Hardy, P. and Waterman, M. S.: The Sequence Alignment Software Library. University of Southern California LA USA, (http://www-hto.usc.edu/software/seqaln/)
[13] Olsen, J., Woese, C. R., and Overbeek R.: The Winds of Evolutionary Change: Breathing New Life into Microbiology. Journal of Bacteriology, Vol. 176 issue 1 (1994) 1 - 6
[14] Papadimitrou, C. H., and Steiglitz, K.: Combinatorial Optimization: Algorithm and Complexity. Prentice Hall, (1982)
[15] Searls, D. B.: The Linguistics of DNA. American Scientist 80: 579 - 591
[16] Setubal, J. and Meidanis J.: Introduction to Computational Biology. PWS Publishing Company, (1997)
[17] Sterling, L. S.. and Shapiro, E. Y.: The Art of Prolog. MIT Press, (1994)
[18] Tatusov, R. L., Mushegian, M., Bork P. et. al.: Metabolism and Evolution of Haemophilius Influenzae Deduced From a Whole-Genome Comparison with Escherichia Coli. Current Biology, Vol. 6 issue 3 (1996) 279 - 291
[19] Tatusov, R. L., Koonin, E. V., Lipman, D. J.: A Genomic Perspective on Protein Families. Science, Vol. 278 (1997) 631 - 637
[20] Vitreschak, A., Bansal, A. K., Gelfand, M. S.: Conserved RNA structures regulation initiation of translation of Escerichia coli and Haemophilus influenzae ribosomal protein operons. First International Conference on Bioinformatics of Genome Regulation and Structure, Novosibirsk, Russia, (August 1998) 229
[21] Waterman, M. S.: Introduction to Computational Biology: Maps, Sequences, and Genomes. Chapman & Hall, (1995)