Project: Mapping phase of MAGIC

Submitted by: Rani Zand.
MAGIC's site: http://magicmapping.sourceforge.net

Background:

Genomes undergo changes that can dramatically affect the properties of the organisms. These changes, referred to as mutations, can result in evolutionary more fit organisms, or cause fatal abnormalities. Mutations can be coarsely divided into two groups:

- Point mutations, affecting single nucleotides.
- Large-scale mutations, affecting long genomic segments.

Whereas point mutations have been well studied and characterized (both on the biomechanical level and the statistical-algorithmic level), investigating large-scale mutations is more complicated as it requires sequencing and comparing whole genomes. Yet, this comparison of the whole genomes, referred to as comparative mapping, is important for evolutionary, genetic, and genome rearrangement studies. Here, one is usually interested in investigating the relation between the genomic segments to establish their evolutionary origin:
Are the segments orthologous, and hence were inherited from the organisms' most recent common ancestor? Are they paralogs, and hence were duplicated from an ancestral segment? Did the segments undergo reordering? Were the segments deleted or inserted and — if so — how (insertion sequence, prophage, or horizontal gene transfer)?

Most of the pioneering studies considered the problem of comparative mapping over sets of genes instead of arbitrary genomic segments. These methods start, usually, by calculating an all-against-all alignment of common sets of genes (a preprocessing phase), and then, in a second phase, use clustering techniques to predict operons or collinear blocks. Recently, comparative mapping methods over arbitrary genomic segments were developed. The preprocessing phase in these methods consists of searching for similar genomic segments — referred to as hits, markers, or anchors (usually performed by a fast local alignment procedure). Then, in the mapping phase, a clustering procedure is applied to the output of the first phase. Examples for fast seed-based preprocessing phases include BLASTZ and CHAOS, in which the seeds are allowed to contain degeneracy, as well as that of Mauve, which searches for exact and unique matches. Examples for mapping phases include CHAIN-NET, FISH, GRIMM-Synteny, Mauve (as well as GRIL — its predecessor), and SLAGAN. CHAIN-NET and GRIMM-Synteny use the distance between anchors as a criterion for clustering (and hence referred to as distance-based mapping methods). Mauve and SLAGAN rely on solving (different) optimization problems to prune anchors and to define the mapping. SLAGAN prunes anchors by introducing various gap penalties for discriminating between different subsets. Finally, FISH is based on a statistical model for anchor clustering.

In this project, MAGIC (a rearrangement of Integrative and Accurate Comparative Genome Mapping), which is a new approach for comparative mapping (Swidan et. al., PLoS CB, 2006), is implemented in C++. Along with the C/C++ implementation of the pre-processing phase, and the Java graphical user interface (GUI), this tool provides an accurate, fast, and user friendly program for comparing genomes.

Project Relevancy

MAGIC's clustering approach is based on a new definition of "consecutive homologous segments" which relies on a biologically intuitive ordering of similar segments. It enables a better handling of duplications in general and allows to adequately address the problem of "nuisance cross overlaps", i.e., misleading similarities between duplications that occurred before the most recent common ancestor, in particular. Nuisance cross overlaps can introduce significant artifacts in the mapping and, to the best of our knowledge, were not taken explicitly into account before. MAGIC is also robust with respect to both its parameters' values and the initial set of anchors. It is capable of modifying and refining the mapping induced from the anchors, and even recognizing and reassigning false orthologs in the initial anchor set itself. Furthermore, MAGIC is scalable and can be applied to distantly related pairs as well as to large genomes. Finally, MAGIC is explicitly designed to handle circular genomes (by considering the last and first nucleotides to be successive). MAGIC's output consists of detailed coverage statistics of the genomes and of several tables including a one-to-one table describing the reorder-free segments.

By applying MAGIC to bacterial genomes and analyzing the results, Swidan et. al. (2006) showed that lateral transfer and large deletions affect bacteria significantly more than duplications. In addition, by applying statistical tests to these results, they surprisingly showed that the breakpoint distribution fits well to the uniform distribution in most of the compared pairs.

For more information we refer the reader to (Swidan et. al., PLoS CB, 2006).

Description of the Algorithm:

The mapping phase's input constitutes of a comprehensive set of maximal similar segments between two given genomes. Practically, this set is represented as a table (see Table 1 in the end of this document). This table is generated in the preprocessing phase. A thorough description of its generation is given in the preprocessing phase documentations.

To describe MAGIC's methodology for comparing two given genomes, we present first the following notation (an overview of the flow of the algorithm is given in Fig. 1).

Given a comprehensive table, T, having the same structure as Table 1, let T[I; S1, E1] denote the start and end coordinates of entry I in the first organism. Similarly, let T[I; S2, E2] denote the start and end coordinates of entry I in the second organism. Let T_org1 denote the table resulting from sorting T in an increasing lexicographical order according to S1, E1 (i.e. according to the start and end coordinates of the first organism). Similarly, let T_org2 denote the table resulting from sorting T in an increasing lexicographical order according to S2, E2 (i.e., according to the start and end coordinates of the second organism).

Based on the above notation, we can describe the methodology used in the mapping phase (see also Fig. 1):

Step 1: Keeping single copy of identical entries

The initial comprehensive table may contain multiple identical entries (i.e. entries having the same S1, E1, S2, E2, and sign values). To keep only one copy out of these, we consider first T_org1 (i.e. the table sorted according to S1, E1) We scan this table for identical entries. In case such redundant entries are found, we keep a single entry and remove the rest. The time complexity of this step is O(|T|log(|T|)).

Step 2: Removing short entries

The table resulting from Step 1 may often contain short entries that one might want to filter out. Entries whose length in either of the two organisms is less than a given threshold (parameter cleanMinLen – see Table 4) are removed. The length of the entry in the first organism is calculated according to S1, E1, while taking into account the genome's circularity, if applicable. The length of the entry in the second organism is calculated similarly. The time complexity of this step is linear in the size of the table (O(|T|)).

Step 3 (optional): Removing Transposable Elements

Transposable elements (TE) are another type of noise that complicates the mapping process. MAGIC can filter out entries associated with TEs, given that the user supplies the annotations of these elements in each genome. The format of the files containing the information on the TEs is described in Table 2.
Given that the user supplied the annotation of the TEs, entries in the table that intersect significantly (more than cleanISPerc) with a TE are discarded. This filtering step can be done efficiently (in O(nlog(n)+n(m+k)) time, where n is the size of the comprehensive table, m the size of table containing the TE annotation in the first organism, and k the size of the table containing the TE annotation in the second organism). Our strategy here is similar to merge sort: First, we consider the TE table of the first organism and the T_org1 table. These two tables are simultaneously scanned as in merge sort, while updating the

amount of intersection between table entries and TEs. If the genomes are circular, we do the following correction: For each entry in the TE table we check if it intersects with any of the "circular" entries - i.e. entries whose end coordinate is less than their start coordinate (T[I, S1] > T[I,E1)) - in the comprehensive table. A similar process is done on the TE table of the second organism and the T_org2 table.

Finally, all entries in the table whose intersection percentage with TE segments is larger than a given threshold (parameter cleanISPerc) are removed.

Step 4 (optional): Removing Viruses

Similar to Step 3, but instead of the TE annotation, a virus (or a prophage for prokaryotes) annotation is used (see the structure of a virus file in Table 3). The threshold for this step is given by parameter cleanProPerc and its time complexity is O(nlog(n)+n(m+k)), where n is the size of the comprehensive table, m the size of table containing the virus annotation in the first organism, and k the size of the table containing the virus annotation in the second organism.

Step 5: Removing overlaps

The table may contain entries that overlap in both organisms. These entries should be joined into one entry.
We do that by scanning the T_org1 table: For each entry I, let J be its successive (in T_org1). If J intersects with I in both organisms, we join them into a single entry and increase J (J = J + 1). If J intersects with I only in the first organism, we move to increase J (J = J + 1, without joining it to I). If J does not intersect with I, we increase I (I = I+1), and update J accordingly (J = I + 1). Finally, if entry I is circular (T[I, S1] > T[I, E1]), we perform in addition a similar check with all the entries in the beginning of the table T_org1. The time complexity of a single iteration of this step is Ω(|T|log(|T|)) and O(|T|^2).

This step is repeated until no more overlapping entries in both organisms are found.

Step 6: Clustering

One of the main goals of the mapping phase is to identify the reorder-free segments in the given genomes. This identification is complicated by point mutations, indels, and (in- or out-) paralogs that might have affected these segments. Overcoming the point mutation hurdle is done in the preprocessing phase. However, dealing with the other hurdles needs to be done in the mapping phase. For that, we seek to identify segments having the same orientation in both genomes and potentially either duplicated or spanned with indels to cluster them together.

To this end, MAGIC uses a novel definition of "consecutive" entries to identify reorder-free segments. Intuitively, two entries are consecutive if they (or more generally their duplicates) are successive in both organisms; see (Swidan et. al., PLoS CB, 2006) for more details.
The calculation of consecutive entries is done as follows: First, given an entry I, we identify all its "significant overlaps" in the first organism (according to T1_org1). Two entries are said to significantly overlap if their intersection percentage in the given organism is greater than the value of the parameter dupPerc. Similarly, we identify all I's significant overlaps in the second organism. Finally, we calculate the intersection of these two sets with the help of the C++ STL Set class (i.e., significant overlaps in first and second organisms). If the intersection contains a single entry, say J, then I and J are defined as consecutive and are joined together.
The time complexity of this step is Ω(|T|log(|T|)) and O(|T|^2).

Step7: Removing Nuisances

Nuisances (or nuisance cross overlaps) are a computational hurdle, resulting usually from the outparalogs biological phenomenon. They can confuse the clustering algorithm and cause a wrong estimation of inparalogs.
Nuisances are identified and removed as follows: First, given an entry I, we calculate its significant overlaps according to the first organism (see Step 6). We seek significant overlaps that are much longer than I, i.e. I's length is less than a given percentage (parameter orthPerc) of their length. If such an entry is found in the first organism, we seek a similar one in the **second** organism. If two such entries are found, I is considered a nuisance and is discarded.
The time complexity of this step is Ω(|T|log(|T|)) and O(|T|^2).

Step8: Identifying Inparalogs

Identifying inparalogs is required to quantify the amount of duplications that occurred since the divergence of the organisms from their cenancestor. It prevents also the inparalogs from confusing the clustering algorithm.
Inparalogs are identified similar to nuisances with the exception that we require them to be much shorter than one of their significant overlaps only in a single organism (and not in both).
The time complexity of this step is Ω(|T|log(|T|)) and O(|T|^2).

Steps 6, 7 and 8 are repeated until convergence of the table.

Step 9: Removing Final Conflicts

At this stage the table might still contain significant overlapping entries for which paralogs and orthologs cannot be determined (because the overlapping entries have similar lengths). Such significant overlaps are referred to as conflicts. Conflicts might result, e.g., from unidentified TEs that have selfishly duplicated in the genome. Hence, one needs to remove these conflicts.
Removing these entries is done as follows: For each entry I, if it has significant overlaps in the first organism (calculated over T_org1), then the entry I, and the significant overlaps are removed from the table. A similar process is performed in the second organism (over T_org2).
The time complexity of this step is Ω(|T|log(|T|)) and O(|T|^2).

Step 10: Removing Final (non-Conflict) Overlaps

At this stage, the table contains no significant overlaps, however, it might still contain some (non-significant) overlaps. To calculate coverage statistics of the genomes correctly, these overlaps need to be removed. This is done by scanning the tables T_org1 and T_org2 and resizing entries that are found to overlap in either of the two organisms.
The time complexity of this step is Ω(|T|log(|T|)) and O(|T|^2).

Software description:

The mapping application has two interfaces: 1) a command line and 2) a Java GUI.

1) We describe the parameters used in the command line version:

mappingPhase {tableFile} {numOfRowsInTable} {outputFile} {outputTracePath} {cleanISPerc} {cleanProPerc} {cleanMinLen} {dupPerc} {orthPerc} {org1DataFile} {org2DataFile} {deletedParalogOrg1File} {deletedParalogOrg2File} [TEFile1] [TEFile2] [ProphagesFile1] [ProphagesFile2]

- tableFile – a string (without white spaces) – The path to the input comprehensive table file.
- numOfRowsInTable – an unsigned int – The number of lines of the comprehensive table file (not including the header line).
- outputFile– a string (without white spaces) – The path to the output file containing the reorder-free segments.
- outputTracePath – a string (without white spaces) – The path to the trace file.
- cleanISPerc – a double – The percentage threshold for filtering out TEs (see Step 3 above).
- cleanProPerc – a double – The percentage threshold for filtering out viruses (see Step 4 above).
- cleanMinLen – an unsigned long – The length threshold for filtering out short entries (see Step 2 above).
- dupPerc – a double – The percentage threshold for determining significant overlaps (see Step 6 above).
- orthPerc – a double – The percentage threshold for distinguishing between paralogs and orthologs (see Steps 7 and 8).
- Org1DataFile – a string (without white spaces) – The path to the input properties file of the first organism.
- Org2DataFile – a string (without white spaces) – The path to the input properties file of the second organism.
- deletedParalogOrg1File – a string (without white spaces) – The path to the output file to print the deleted entries from the ‘Remove Paralogs’ step in organism 1.
- deletedParalogOrg2File – a string (without white spaces) – The path to the output file to print the deleted entries from the ‘Remove Paralogs’ step in organism 1.
- TEFile1 (optional) – a string (without white spaces) – The path to the input file that contains the TE description in the first organism.
- TEFile2 (optional) – a string (without white spaces) – The path to the input file that contains the TE description in the second organism.
- ProphagesFile1 (optional) – a string (without white spaces) – The path to the file that contains the viruses (or prophages) description in the first organism.
- ProphagesFile2 (optional) – a string (without white spaces) – The path to the file that contains the viruses (or prophages) description in the second organism.

2) When the application is initiated through the Java GUI, these parameters can be set through GUI (refer to the GUI's documentations for more details).

Software design:

Diagram 1 describes the relationships between the main classes used in the code.
Here we describe these classes in more detail:

Genome:
This class holds the information and the properties of a genome.
Currently, the following properties are associated to each genome:

- The genome's length in nucleotides.
- The genome's name.
- The genome's circularity.

Entry:
This class contains the fields required to describe an entry in the table T (see Section Description of the Algorithm and Table 1).
Each entry has the following information:

- An indicator as whether the entry is in use or not.
- The (unique) label of the entry.
- The start and end coordinates of the segments in the two organisms (previously referred to as S1, E1, S2, E2).
- The entry's sign: '+' denotes that the two segments have an identical orientation, and '-' denotes that the two segments have a reversed orientation.

Table:
This class contains the fields required to hold all the table's information:

- The entries of the table (of type Entry[]).
- The number of entries in the table (table's length).
- The properties of the first genome (of type Genome).
- The properties of the second genome (of type Genome).

In addition, the Table class provides all the functionality required in the mapping phase.

compare_entries_1:
This class has two purposes.
The first is to enable sorting of the table according to the first organism, and is required by the C++ STL (Standard template library) sort function.
The second is to ensure that only a single copy of an entry is inserted into an instance of the STL set class.

compare_entries_2:
The purpose of this class is to enable sorting the table according to the second organism and is required by the C++ STL (Standard template library) sort function.

Software dependencies:

- STL (Standard Template Library) library.

e.coli.start	e.coli.end	length	s.flexneri_2457t.start	s.flexneri_2457t.end	length.1	signs	identifier
16728	19802	3075	15378	18452	3075	+	KIS35-4
16760	17055	296	2039306	2039621	316	+	G683
19790	20553	764	2160650	2161413	764	+	G786
19790	20565	776	4389052	4389827	776	-	G1247
19793	20559	767	1053802	1054568	767	+	G1426
19794	20453	660	3168464	3169123	660	-	G956
19795	20533	739	3393040	3393778	739	+	G1016
19795	20549	755	4480585	4481305	721	+	G1216
19795	20550	756	2186313	2187068	756	-	G802
19795	20555	761	1774278	1775038	761	-	G619
19795	20559	765	275896	276660	765	+	BP713
19795	20559	765	1240535	1241299	765	+	G1463
19795	20559	765	1795335	1796099	765	+	BP480
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Table 1

Start	End
5635	5670
6360	6656
12102	12221
12752	12788
18663	19441
25952	25987
36705	37483
48032	48160
.	.
.	.

Table 2

Start	End
15401	15610
48627	49106
195557	199039
238088	238294
238511	238780
239101	239415
.	.
.	.

Table 3

Parameter	Used for	In	Default value
cleanMinLen	removing short entries	Step 2	200
cleanISPerc	removing transposable elements	Step 3	0.4
cleanProPerc	removing prophages	Step 4	0.4
orthPerc	identifying orthologs	Step 7 and 8	0.5
dupPerc	fnding duplicates	Setps 5-9	0.5

Table 4

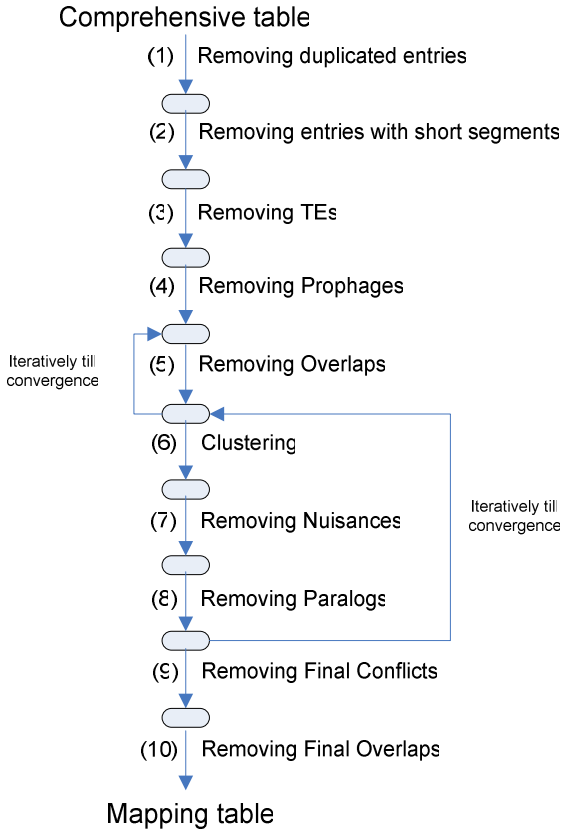

Figure 1

Class Diagram

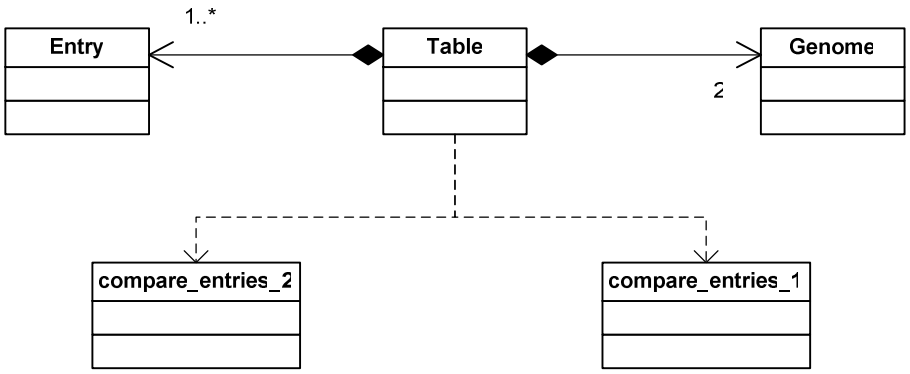

Diagram 1