MGA-Toolbox Version 0.1
Tool for Aligning Multiple Graphs

Thomas Fober Marco Mernberger
{thomas, mernberger}@mathematik.uni-marburg.de

Philipps-University Marburg
Mathematics and Computer Science Department
Knowledge Engineering & Bioinformatics Group
35032 Marburg
Germany

2008 January, 24th

Contents

1 Introduction

1.1 Multiple Graph Alignment

Graphs have been widely used for modelling structured objects that are difficult to model by other means. Especially in the field of bioinformatics, graphs have been used for modeling chemical structures, proteins and protein substructures as well as ligands that bind to them. Usually one is

interested in finding similarities between such molecules, for example to detect chemical ligands that bind to the same proteins or proteins that share similar functions.
Furthermore, graphs have been used beyond that field as a general means of modeling structured objects, e.g. XML documents, text documents, handwritten words, social networks or as object descriptors in images. Here, too, a comparison of graphs is likewise interesting, for example one might be interested in comparing hand-written texts, etc. Therefore a tool is needed for the comparision of such objects. Although several approaches exists, based on graph isomorphism, subgraph isomorphism and pattern recognition, most of them are limited to a pairwise comparision of graphs.
Our tool enables us to compare multiple graphs at once by calculating a multiple graph alignment. Therefore we are not limited to a pairwise comparison. A graph alignment in this respect is defined as a one-to-one mapping of vertices of different graphs onto each other, in a way that the correspondence of vertex labels as well as the mapping of edges with similiar distances onto each other is maximized. Therefore we expect the graphs to be vertex-labeled and edge weighted, where edge labels are interpreted as distance values.
Compared to sequence alignments, graph alignments present a much greater algorithmic challenge, as the vertices in a graph do not posess a natural order, as is the case with sequences with a certain sequence order. Moreover, a graph alignment tries to map the graphs in such a way that a maximum of edge identity as well as label identity is achieved. Since these alignments do not rely on exact matching techniques, inexact assignments of vertices to each other as well as edges to each other are allowed, albeit at a cost.
We measure the quality of a given alignment by means of a scoring function, in which we penalize edge mismatches and vertex mismatches separately and reward edge and vertex matches by scoring constants. By adjusting these scoring parameters, one is able to increase the influence of vertex labels and decrease the influence of edges on the calculated alignment and vice versa. Since edges should be
Since the number of possible mappings increases exponentially with the number of vertices in a graph, the task of finding the best possible alignment is very complex and the MGA-Problem is NP-hard.

1.2 Algorithms

We have implemented two algorithms solving the MGA-Problem; an evolutionary algorithm (EA) and a greedy heuristic. As showed in [3] the EA is able to produce alignments of much higher quality, albeit at the cost of a considerable increase of runtime. Therefore we recommend using the EA only on small graphs with up to 100 vertices. As a compromise between quality and runtime it is also possible to use a combination of greedy heuristic and

EA, called star-EA. This algorithm uses the EA to calculate pairwise alignments and aggregates them to a multiple one using the star-align heuristic. Obviously this method is only usable on MGA problems with more than two graphs.
The implementation of our algorithms is described in the appendix. Here we want to give a brief introduction in the three mentioned algorithms:

EA: The representation of the EA is able to encode a multiple alignment for m graphs. Unlike the algorithms described below the optimization procedure can consider the whole problem at once. Ingenious genetic operators, especially developed for the MGA problem are able to construct an optimal alignment reliably. However MGA is a very complex problem and the search-space grows exponentially with the number of vertices and graphs. In addition, the underlying fitness-function is based on a sum-of-pairs measure and its runtime grows also with number of vertices and number of graphs. With some clever tricks we could reduce the runtime for evaluation the fitness-function of a mutiple alignment dramatically but with very large number of vertices and graphs the runtime for optimization does still explode. Then EA in combination with star alignment can be used.

EA + Star alignment: This variation uses the EA, too, with the difference that only pairwise alignments are calculated. The MGA problem with m graphs is decomposed into $\frac{1}{2}m \cdot (m+1)$ pairwise alignments that are solved by EA. A heuristic called star alignment merges these $\mathcal{O}(m^2)$ pairwise alignments to a multiple one. We have shown [3] that this procedure leads to a significant speed-up of the optimization process at the cost of a slight decline in fitness and quality of the alignment. As mentioned above the runtime still grows exponentially with the number of vertices so that there are cases in which we have to replace the EA by a simple greedy heuristic.

Greedy + Star alignment: This procedure is similar to the EA + Star alignment method and simply replaces the EA by a greedy heuristic. This heuristic quickly solves the subgraph-isomorphism problem in a first step producing a seed-solution of the pairwise graph alignment problem. In a second step, this seed-solution is greedily extended to an alignment by adding the remaining vertices to the current solution. Once again we have to decompose the MGA problem as described above and use the star alignment heuristic. In comparison to all other described approaches, this method is the quickest one at the expense of a reduced quality of the results regarding fitness and quality.

2 Requirements and Installation

Our program is implemented in Java 1.6. Since Java is plattform independent, the MGA-Tool is runable on each operating system. You just have to install the Java interpreter and to copy the MGA-Tool file mga.jar into any folder. The latest version of the MGA-Tool is available at:

`www.uni-marburg.de/fb12/kebi/research/software/` .

3 Usage

The MGA-Tool can be started by typing `java -jar MGA.jar` plus additional arguments in a command window with correct path.
`java -jar MGA.jar -?` returns a list with all possible settings that are described in the following section in detail.

```
usage: MGA [options] [method] inputdirectory {inputdirectory2}
                              outputdirectory {parameter}

OPTIONS:
'-?':           Help, displays this text.
'-p':           calculate pairwise alignments. 2 input
                directories needed.
                All files in first directory are aligned
                with all files in second directory
'-m':           calculate multiple alignment. 1 input directory needed.
                All files in directory are aligned with each other

METHOD:
'EA':           Use Evolutionary Algorithm for calculation.
'EA+S':         Use Evolutionary Algorithm + star alignment for calculation.
'GS+S':         Use Greedy Strategy + star alignment for calculation.

PARAMETER:
'stallGen:x'    : stop EA after x stall generations. Default: x = infinity.
'stallTime:x'   : stop EA after x stall seconds. Default: x = infinity.
'time:x'        : stop EA after x seconds. Default: x = infinity.
'gen:x'         : stop EA after x generations. Default: x = infinity.
'fitness:x'     : stop EA at a fitness of x. Default: x = infinity.
'mu:x'          : population size. Default: x = 4.
'nu:x'          : selective pressure. Default: x = 20.
'delta:x'       : cutoff value for edge distances. Default: x = 11.
'nmm:x'         : penalty for vertex mismatches. Default: x = -5.
'nm:x'          : bonus for vertex matches. Default: x = 1.
```

```
'd:x'           : penalty for dummy matches. Default: x = -2.5.
'emm:x'         : penalty for edge mismatches. Default: x = -0.2.
'em:x'          : bonus for edge matches. Default: x = 0.1.
'eps:x'         : tolerance threshold for edge length differences.
                  Default: x = 0.2.
```

3.1 Program settings

First the user has to specify if a pairwise alignment or a multiple alignment should be calculated. This is done with the parameter `-p` or `-m` respectively. The former requires two input directories. All graphs in the first directory are (pairwise) aligned with the graphs in the second directory. The results are stored in the specified output directory. The latter requires one input directory. All graphs in this directory are aligned and the resulting multiple alignment is stored in the output directory.
Before specifying the input- and output directories the calculation method must be choosen. There are three methods available in our toolbox:

-EA EA for calculating the multiple graph alignment

-EA+S EA for calculation of $\mathcal{O}(m^2)$ pairwise alignments and the susequent merging of these by the star-align procedure.

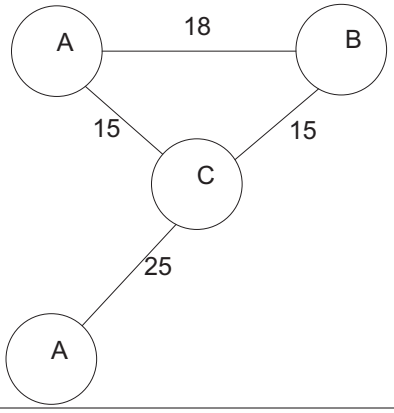
-GS+S Greedy heuristic for calculation of $\mathcal{O}(m^2)$ pairwise alignments and the susequent merging of these by the star-align procedure.

Typically the upper needs the most time for calculation but is the exactest method. The last procedure is the fastest one but only a heuristic which cannot guarantee the optimal solution.
If *EA* oder *EA+S* is used for solving the MGA problem, EA-typical parameters can be specified. The population size μ can be set by `mu:`μ a selective pressure of ν analogous by `nu:`ν. Other EA typical parameter are not settable since we have performed some parameter tuning [1] that indicated that these are the only parameters that have influence on runtime and result. The parameter tuning indicates also recommended values. If `mu` and `nu` are not set by user than this recommended values are used during optimization.
The EA further needs one or several termination criteria. *Stall generations* are the generations in which no improvement occurs. This generations are a good indicator, for the progress of the optimization. Analogous it is possible to specify *stall time* as termination criteria. If the optimal *fitness* is known we can terminate, if this fitness is reached. Termination Criteria not based on progress are *time* and *generations*. It the maximum time or generations respectively are reached the optimization procedure is leaved. All these parameters can be set by parameter:value syntax as described above.
If *GS+S* is used as solver the described parameters are not required because greedy and star-alignment heuristic are parameterless.

Our fitness function is also parameterisable: As described in section 1.1 we can specify constants for match `nm` or mismatch `nmm` of assigned vertices as well as for edges (`em` and `emm` respectively). If the graphs differ in respect to their size (or if it is better to assign a vertex of one graph to no vertex of another graph) we match some vertices onto dummy vertices which is penalized by a constant `d`. Note that we are solving a maximization problem. So match constants should be larger than mismatch and dummy constants. Especially for biological applications we obtain a complete graph if we encode distances between vertices as edges with associated label. Sometimes these graphs contain to much information so that it has advantages to remove all edges with higher label than δ. This is done by setting the constant δ to a specific distance threshold. As edge labels are interpreted as distances, one can specify a tolerance threshold `eps` that sets the maximum for the difference in edge lengths, up to which the edges are still counted as a match. All here described settings are realized by parameter:value syntax.

3.2 Parser for graphs

The MGA Toolbox can distinguish between *mol2*, *pseudoc* and *rlbcoor* format[4]. Files ending with *mol2*, *pseudoc.txt* or *rlbcoor* are automatically parsed by their belonging parser. Additionally we have implemented a general parser reading graphs in a simple format based on matlab syntax. This format stores a graph in a simple text file ending with *.graph* (not with txt!) and containing exact two lines. The first line encodes the vertex-labels




```
A,B,C,A,
-,18,15,-;18,-,15,-;15,15,-,25;-,-,25,-
```

Figure 1: Example for illustrating the .graph-format

separated by comma; in the second line each row of the adjacency matrix is separated by semicolon, entries in the row by comma. Figure 1 shows a

simple vertex-labeled and edge-weighted graph with corresponding encoding in *graph*-format ("-" indicate no edge).

3.3 Output

The calculated alignments are stored in a *mga* file. This file is human readable and can be opened by a text editor. The assigned name of this file is based on the name of the graphs that are aligned and is a composition of the first and the second graph separated with "___". In the multiple case, the result file is named by the first input graph plus the keyword multiple. The structure is as follows:

1. name of the aligned graphs
2. number of graphs
3. calculation method
4. time required for calculation
5. score of alignment
6. required generations (if EA was used for solving the MGA problem)
7. reason for stopping (interesting if more than one criteria was used)
8. used scoring parameters
9. alignment matrix: each column codes an assignment of single vertices indices out of each graph, if a -1 exists in cell (i, j) we use a dummy vertex in graph i and assignment-column j instead of a vertex out of graph i.
10. for each graph based on alignment ordering:
    (a) labels: vertex labels
    (b) distance: adjacency matrix

3.4 Example

We have two classes of graphs in two orders *F1* and *F2* and want to align each graph out of each order pairwise and store the result in the folder res. As method we want to use an EA to achieve best results. The stopping criterion is 500 stall generations and we want to use standard parameters. The command is like follows:

```
java -jar MGA.jar -p EA /home/thomas/F1 /home/thomas/F2 /home/thomas/res stallGen:500
```

A Problems and Solutions

Problem	Reason / Solution
The program does not stop.	Check the parameters you have given. If you specified stop criteria that cannot possibly be met (such as a certain fitness value) the program would not stop.
The program does not use the right files for the alignment calculation.	Make sure you have specified the input and output folders in the right order.
The program breaks with a "no such directory error".	Make sure your directories exist and be sure to specify two input directories for the pairwise case. These can of course be the same but you have to specify it two times.
My parameters are ok, but the program still does not seem to stop.	MGA is a very complex problem that needs time to be solved. If you use the EA change the stopping criteria or switch to the greedy heuristic to save time.
The calulated alignment is obviously not optimal.	If you used any approach other than EA, the program cannot guarantee to reach a global optimum. If you used the EA and it is still not optimal, try again and allow a longer runtime by changing the stop criteria.
The program did calculate an alignment, but it is not what we expected.	Be aware, that the scoring parameters have a strong impact on the definition of a "good" alignment. By varying these, one can emphasize the importance of vertex labels or edge weights respectively. E.g. if the alignments show perfect vertex label assignments but completely neglect structural similarities, try to increase match boni for edges or lower match boni for vertices.
How can the program be stopped?	Use your operating system to kill the process (i.e. 'kill' on UNIX-System or `Strg-c` on Windows-Systems).
I can not parse my graphs.	The files storing the graphs does not end with '.mol2', '.pseudoc.txt', '.rlb-coor' or '.graph'.

Problem	Reason / Solution
My graph-file has the correct ending but parsing is not possible.	1. Your file does not use the standard syntax defined in [4]. Translate the file into the '.graph' format (cf. section 3.2) and parse it again. 2. Windows Systems hide well-known endings of a file, so that it is possible, that i.e. a file *test.mol2.txt* is displayed as *test.mol2*.
The program breaks with a *Null Pointer Exception*.	Options, Method, Directories, Parameters are given in the false order during calling the program out of the command window.
Is a graphical user interface available?	No!

If you have any problems or comments concerning the algorithm or the implementation feel free to contact us.

B Implementation – An overview

B.1 Greedy heuristic

The greedy heuristic uses the Bron-Kerbosh-Algorithm for the calculation of a seed-solution for a pairwise alignment. Here, vertices consist of tuples containing one vertex from graph 1 and one vertex from graph 2. Let $G_1 = (V_1, E_1)$ be graph one and $G_2 = (V_2, E_2)$ be graph two. For all $v_i \in V_1$ for all $v_j \in V_2$ the tuple (v_i, v_j) is added to the productgraph, if both vertices have the same label. An edge is drawn between two product vertices, if the corresponding vertices are connected in both graphs or not connected in both graphs.
Then a clique algorithm performs a search of the first 100 cliques in the graph and sets the largest clique as the seed-solution. The algorithm then extends the solution greedily, in a way that the vertex tuple is added to the seed-solution that has the largest number of neighbours inside the seed solution. This is done until all remaining vertices are added to the solution. In the multiple case, all possible pairwise alignments are calculated and subsequently merged via a star-alignment merging algorithm.

B.2 Star alignment

The star align merging algorithm merges a number of pairwise alignments to one multiple alignment. For a given set of graphs all pairwise alignments are calculated, either by the greedy algorithm (greedy approach) or by the EA (star-EA). We subsequently calculate a multiple alignment by using one graph as a pivot graph and align all other graphs as calculated in the pairwise alignments. For each vertex in the pivot graph, the corresponding matched vertex in the particular pairwise alignment is mapped onto it. If dummy vertices occur, they will be mapped onto dummies in the corresponding pairwise alignment. This is done for each pairwise alignment in which the pivot graph is aligned with another graph, starting with the biggest in respect to the number of vertex entries. If eventually an alignment is added that has a shorter length, the remaining positions are filled with dummy vertices. As it is unclear, which graph is suited best as a pivot graph, we try each graph as a pivot graph and pick the multiple alignment yielding the highest score.

B.3 Evolutionary algorithm

The EA encodes the MGA problem as $m \times n$ matrix, with m is number of graphs and n is a variable. We use ES typical operators [2] for mating-selection and selection where for the selection only plus-selection is used. As termination criteria we have implemented stall-generations, stall-time, time, generations and fitness which are realized as simple if-conditions.
The initialization is realized as follows: We determine the number of vertices in the largest graph $maxL$ and set the genome-length n to $maxL + 1$. Now a $m \times n$ matrix is created and for each row a random permutation of 0 to $maxL$ is generated. Obviously if any of the graphs does not contain enough vertices, the remaining cells (i, j) are set to "-1" which indicates a dummy-vertex.
For mutation we use a very simple mechanism that selects two cells in the same row and swaps them. We have also implemented much more complex operators (i.e. using a self-adaptation mechanism) – but, experimental results have shows that a simple mutation performs much better.
The recombination is realized for ρ Individuals. So it is possible to switch off the recombination by set ρ to one. Once again experimental results have shown that recombination should be turned on and ρ set to 2. The recombination operator takes two uniformly randomly choosen parental individuals, cuts them on a uniformly randomly choosen row and exchanges their in this way created blocks. Note, that alignment indices are not ordered in the alignment, so a simple merging does not show the desired effect of improving fitness. Therefore we use the row on which the split occours as pivot-row to ensure the right order.
A genome-length adaptation mechanism ensures that we work always on a

optimal genome length. It is clear, that a too short genome-length can result in a not optimal alignment, on the other hand, a too large genome-length enlarges the search-space enormously and slows down the optimization process. Another advantage is that the genome-length must not be specified by the user. We ensure (cf. initialization) that individuals always carry one dummy-column. Such a dummy-column, if dummies are required, will be integrated in the alignment. Because this integration needs time we check with small probability the number of dummy-columns. Three cases can occour:

1. We have exactly one dummy-column: Then nothing has to be done.
2. We have $k > 1$ dummy-columns: We can delete $k - 1$ of them, since we have a clue, that we deal with to much dummys.
3. We have no dummy-columns: All dummys are in use, so that we have to add a new dummy-column.

At least we want to mention that we use the sum-of-pairs scoring scheme to evaluate individuals. This scoring scheme has the disadvantage that its calculation needs time which increase with number of vertices as well as number of graphs. Therefore we use two techniques that allow us to decrease the computational effort. First we calculate for each column a histogram of vertex-labels in linear time. With help of a simple arithmetical expression we can calculate the score of the column in constant time. Overall this results in a reduction from $\mathcal{O}(m^2)$ to $\mathcal{O}(m)$. A similar trick is used for decreasing the runtime for the calculation of the edge-score. Edges are treated as match if their distance is less than ϵ. To check an assignment of edges we once again need $\mathcal{O}(m^2)$ time. By sorting the edges according to their length we could decrease the time to $\mathcal{O}(m \cdot log(m))$.

References

[1] Thomas Bartz-Beielstein, 2006: Experimental Research in Evolutionary Computation. The New Experimentalism, Springer, Berlin

[2] Hans-Georg Beyer, Hans-Paul Schwefel, 2002: Evolution strategies – A comprehensive introduction, Journal Natural Computing, Issue Volume 1, Number 1, Springer Berlin, ISSN 1567-7818

[3] Thomas Fober, Eyke Hüllermeier, Marco Mernberger, 2007: Evolutionary Construction of Multiple Graph Alignments for the Structural Analysis of Biomolecules, Proceedings, 17. Workshop Computational Intelligence, Bommerholz

[4] Johann Gasteiger, Thomas Engel, 2003: Chemoinformatics, Wiley-VCH, Weinheim