GraphAlignment Package Manual

Jörn P. Meier, Michal Kolář, Ville Mustonen,
Michael Lässig, and Johannes Berg

Institut für Theoretische Physik, Universität zu Köln
Zülpicherstr. 77, 50937 Köln, Germany

November 24, 2026

Contents

1 Introduction **3**

2 Releases **4**

3 Installation **4**
3.1 Importing the package 5
3.2 Documentation 5
3.3 License 6

4 Definitions **6**
4.1 Networks and network alignments 6
4.2 Alignment scores 7
4.3 Scoring parameters 8

5 Sample sessions **9**

6 Implementation **16**

7 Package Contents **18**
7.1 Functions 18
7.1.1 GenerateExample 18
7.1.2 InitialAlignment 19
7.1.3 InvertPermutation 19
7.1.4 Permute 19

7.1.5 GetBinNumber 19
7.1.6 VectorToBin 20
7.1.7 MatrixToBin 20
7.1.8 ComputeLinkParameters 20
7.1.9 ComputeNodeParameters 20
7.1.10 EncodeDirectedGraph 21
7.1.11 ComputeM 21
7.1.12 AlignNetworks 22
7.1.13 AlignedPairs 22
7.1.14 LinearAssignment 23
7.1.15 ComputeScores 23
7.1.16 AnalyzeAlignment 23
7.1.17 Trace 23
7.1.18 CreateScoreMatrix 24

1 Introduction

GraphAlignment is an extension package for the R programming environment. It provides functions for finding an alignment between two networks based on link and node similarity [1].

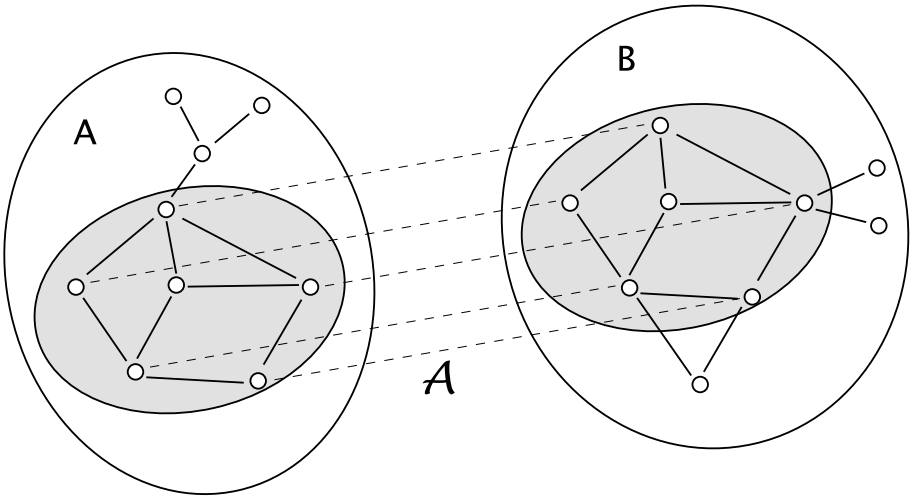


Figure 1: An alignment $\mathcal{A}$ between two networks A, B is a mapping (indicated by dashed lines) between nodes of the subsets $\hat{A}$, $\hat{B}$.

Finding similarities between networks is an algorithmically hard problem well known in computer science. In this package, the assessment of link and node similarity (scoring) is designed specifically for the cross-species analysis of biological networks. A key feature is that network nodes may be aligned on the basis of their link similarity only, without any sequence similarity. It thus goes beyond selecting node pairs with high link similarity from sequence homologs. The algorithmic problem of finding the alignment of two networks with the maximum score is solved using an iterative heuristic based on the linear assignment problem [2]. The iterative scheme a small amount of noise in order to escape possible local score maxima. For details on the scoring and the algorithmic method implemented here, see [1].

R is a language and environment for statistical computing and graphics, particularly useful for exploratory data analysis. It is available freely and for many different platforms from the website `http://www.r-project.org/`. The website also offers a manual and introductions to the R language. Very little knowledge of the R language is required to use this package. Users experienced with Matlab or Mathematica will find the worked examples in this manual self-explanatory.

The following sections cover downloading the source, the installation process, and definitions related to *GraphAlignment*. Examples sessions are given and the functions

provided by this package are discussed in detail.

2 Releases

Release tarballs of the package can be obtained from the following location:

`http://www.thp.uni-koeln.de/~berg/GraphAlignment/distfiles/releases/`

You can always get the latest development version of the *GraphAlignment* package from the Subversion repository at:

`http://gap:gap@xi.ionflux.org/svn/GraphAlignment/trunk/`

To get the *GraphAlignment* package from Subversion, you need to have Subversion installed (`http://subversion.tigris.org/`). Then change to a directory of your choice (for example `GraphAlignment`) and type:

```
svn co http://gap:gap@xi.ionflux.org/svn/GraphAlignment/trunk/
```

This, however, is a development version, and might possibly be unstable. You should not use this unless you want to help with debugging and testing of new features.

3 Installation

The *GraphAlignment* package can be installed using the standard R package installation procedure. If you have downloaded a release tarball of the package (named, for example, `GraphAlignment_1.0-0.tar.gz`), use the following command line to install in the specified location:

```
R CMD INSTALL PATH_TO_PACKAGE --library=R_EXTENSION_PATH
```

`PATH_TO_PACKAGE` is the relative or absolute path to the package tarball (for example `GraphAlignment_1.0-0.tar.gz` or `/home/someuser/downloads/GraphAlignment_1.0-0.tar.gz`). `R_EXTENSION_PATH` is the path where the package should be installed. This option may be omitted if you intend to install the package in the default location.

Otherwise, if you have downloaded a snapshot release or got the latest development version from the respository, change to the directory which contains the (unpacked and untarred) *GraphAlignment* distribution directory (named `GraphAlignment`) and type

```
R CMD INSTALL GraphAlignment --library=R_EXTENSION_PATH
```

Please note that since *GraphAlignment* is a source package, a working R command line interface and build tools such as a bash compatible shell, make and a C compiler are required for installation.

3.1 Importing the package

The package is imported into the R workspace by starting up R and entering:

```
library("GraphAlignment", lib.loc="R_EXTENSION_PATH")
```

`R_EXTENSION_PATH` is the path where the package has been installed. The extension path may be omitted if the package has been installed in the default location.

3.2 Documentation

The *GraphAlignment* package provides two types of documentation: Documentation for the R user interface and API documentation for the C implementation of some of the features provided by the package.

User documentation, besides this manual, is available through the built-in help system of the R programming environment, usually by typing `help(<function name>)` or `?<function name>`, where `<function name>` is the name of a function which is provided by the *GraphAlignment* package.

The user documentation generated by R can also be browsed online in HTML format here:

```
http://www.thp.uni-koeln.de/~berg/GraphAlignment/R-docs/
```

Documentation for the C implementation part is available in Doxygen format and may be extracted from the source files by typing `doxygen` in the distribution base directory. HTML documentation will be generated in the docs/html directory and can be viewed with a web browser. Doxygen is free software and can be obtained from `http://www.doxygen.org/`.

A current version of the code documentation is also available online at `http://www.thp.uni-koeln.de/~berg/GraphAlignment/`.

3.3 License

Authors: Jörn P. Meier (mail@ionflux.org), Michal Kolář, Ville Mustonen, Michael Lässig, and Johannes Berg.

The package can be used freely for non-commercial purposes. If you use this package, the appropriate paper to cite is

> J. Berg and M. Lässig, "Cross-species analysis of biological networks by Bayesian alignment", PNAS 103 (29), 10967-10972 (2006)

available from `http://www.pnas.org/cgi/content/full/103/29/10967`.

This software is made available in the hope that it will be useful, but without any warranty, without even the implied warranty of merchantability or fitness for any particular purpose.

This software contains code implementing the Jonker-Volgenant algorithm [2] to solve linear assignment problems (LAP). The code was written by Roy Jonker, MagicLogic Optimization Inc.. and is copyrighted, © 2003 MagicLogic Systems Inc., Canada. It may be used freely for non-commercial purposes.See `http://www.magiclogic.com/assignment.html` for the latest version of the LAP code and details on licensing.

4 Definitions

4.1 Networks and network alignments

Networks are represented by their adjacency matrices. The rank `dimA` of the adjacency matrix of matrix a equals the number of nodes in the network. The adjacency matrix may be symmetric (undirected networks), or asymmetric (directed networks). For directed networks, only binary links are currently implemented. Simple networks for trial purposes can be generated randomly using the function `GenerateExample`.

A (local) alignment between two graphs A and B is defined as a mapping $\mathcal{A}$ between two subgraphs $\hat{A} \subset A$ and $\hat{B} = \mathcal{A}(\hat{A}) \subset B$ as shown in Fig. 1. Aim of the alignment is to detect cross-species functional relationships between aligned node pairs based on the similarity of either nodes or links. Due to gain or loss of genes in either species, not every gene in one network has a functional equivalent in the other, and the alignment algorithm has to determine the aligned subnetworks $\hat{A}$ and $\hat{B}$ with significant correlations. Currently the package supports only one-to-one mappings $\mathcal{A}$, which is appropriate for most gene pairs but neglects multi-valued functional relationships resulting, e.g., from gene duplications. The scoring scheme and algorithm can in principle deal with many-to-many mappings, the implementation will follow in a later version.

The pairwise similarity between genes in networks A and B is given by a matrix $\mathbf{R}$, whose entries R_{ij} quantify, for example, the overall sequence similarity between the gene sequences $i \in A$ and $j \in B$ or a biochemical similarity between the corresponding proteins.

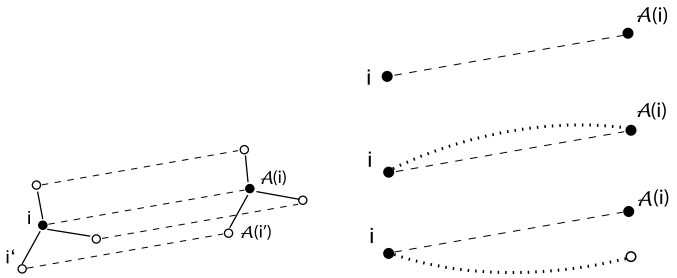
4.2 Alignment scores

For details on scoring and the statistical models behind the scoring parameters see [1]. Each aligned pair of links a_{ij}, $b_{\mathcal{A}[i]\mathcal{A}[j]}$ contributes a link score of $s^l(a_{ij}, b_{\mathcal{A}[i]\mathcal{A}[j]})$, yielding a total link score of

$$\begin{aligned} S^l &= c_0 \sum_{i,i' \in \mathcal{A}, i \neq i'} s^l(a_{ii'}, b_{\mathcal{A}(i)\mathcal{A}(i')}) \\ &+ \sum_{i \in \mathcal{A}} s^l_{self}(a_{ii}, b_{\mathcal{A}(i)\mathcal{A}(i)}) \ , \end{aligned} \tag{1}$$

where $c_0 = (1/2)$ if the adjacency matrix for the network is symmetric, i.e. the network is undirected, and $c_0 = 1$ for directed networks. The notation $i \in \mathcal{A}$ means that the sum is over all aligned nodes in network A, i.e. $i \leq$ `dimA` and $\mathcal{A}(i) \leq$ `dimB`.

The function $s(a, b)$ is a scoring parameter.



(a) Link strength (b) Node similarity

Figure 2: (a) The local link score $S^l_{i,\mathcal{A}(i)}$ evaluates all pairwise similarities between links $a_{ii'}$ and $b_{\mathcal{A}(i)\mathcal{A}(i')}$ (solid lines) for a given pair of aligned nodes. (b) The local node similarity score $S^n_{i,\mathcal{A}(i)}$ evaluates the overlap of the alignment with the node similarity $R_{i,\mathcal{A}(i)}$ (dotted line). Top to bottom: Aligned node pairs (i) without similarity to any other node, (ii) with mutual node similarity, (iii) with (at least one) node similarity mismatch.

The node score assesses the sequence similarity between aligned genes. Each aligned node pair $i, \mathcal{A}(i)$ contributes a score $s_1(R_{i\mathcal{A}(i)})$. In order to assess the sequence similarities with nodes other than the alignment partner, the score $s_2(R_{ij})$ is summed over

all nodes in B i is not aligned to (an vice versa). See figure 2 for details. The total node score is

$$\begin{aligned} s^n &= \sum_{i\in\mathcal{A}} s_1(R_{i\mathcal{A}(i)}) + \sum_{i\in\mathcal{A}, j\leq \texttt{dimB}, j\neq\mathcal{A}(i)} w_{ij} s_2(R_{ij}) \\ &+ \sum_{j\in\mathcal{A}, i\leq \texttt{dimA}, i\neq\mathcal{A}^{-1}(j)} w_{ij} s_2(R_{ij}) \ . \end{aligned} \tag{2}$$

j denotes a node in network B, so $j \in \mathcal{A}$ means $j \leq$ `dimB` and $\mathcal{A}^{-1}(j) \leq$ `dimA`. $\mathcal{A}^{-1}$ denotes the mapping from nodes in B to nodes in A. w_{ij} takes on the value 1 if i or j have an alignment partner (but not both). w_{ij} takes on the value $1/2$ if i and j both have an alignment partner. This is done in order to avoid double counting.

Given an alignment and the scoring parameters the link and node score can be computed using the function `ComputeScores`.

4.3 Scoring parameters

As in the alignment of biological sequences, the choice of scoring parameters is highly non-trivial. Setting, for instance, $s_1(R)$ to infinity for all values of R above a threshold will always align nodes with a partner with high sequence similarity. Depending on the evolutionary dynamics since the last common ancestor of the two networks, different choices of the scoring parameters will be appropriate.

Given a (preliminary) alignment, the log-likelihood estimates of the optimal scoring parameters can be computed using the functions `ComputeLinkParameters` and `ComputeNodeParameters`. These log-likelihood estimates of the scoring parameters are computed as follows.

We describe the statistics of correlated networks by a probabilistic ensemble. Q_{kl} gives the probability of having $a_{ii'} = k$ between a given pair of nodes $i \neq i'$ in network A and $b_{\mathcal{A}(i)\mathcal{A}(i')} = l$ between the alignment partners of these nodes in network B. For binary links $k, l \in \{0, 1\}$ so Q_{kl} is a 2×2 matrix. For weighted networks, where links have continuous values, k and l will be real numbers; Q_{kl} is a probability density. The continuous link may be assigned to discrete bins defined by the user, see sections 5 and 6.

We may further distinguish between self links (links going from a node to itself), with a distribution $Q_{kl,self}$. We denote the corresponding marginal distributions $p_k^A = \sum_l Q_{kl}$, $p_l^B = \sum_k Q_{kl}$ and correspondingly for $Q_{kl,self}$.

For a given alignment, these probability ensembles can be estimated directly from the data. Pseudocounts are employed to avoid empty matrix entries. Log-likelihood scores are set up in the usual way, comparing the likelihoods of links k and l the two ensembles, Q_{kl}, describing links correlated between the two networks, and $p_k^A p_l^B$ describing uncorrelated links The elements of the resulting link score matrices are given

by $s^l_{kl} = \log(Q_{kl}/(p^A_k p^B_l))$ for the link score and $s^l_{kl,self} = \log(Q_{kl,self}/(p^A_{k,self} p^B_{l,self}))$ for the self link score.

For node scores, $q_1(R)$ describes the distribution of the node similarity R_{ij} of aligned node pairs $i, j = \mathcal{A}(i)$. The function $q_0(R)$ gives the distribution of R for node pairs i, j with at least one aligned node (either the one in A, or the one in B, or both), where the nodes i, j are *not* aligned to each other. The function $p_0(R)$ describes the distribution of the node similarity R of all nodes pairs i, j in A,B with at least one aligned node (either the one in A, or the one in B or both). The corresponding node scores are then given by $s_1(R) = \log\left(\frac{q_1(R)}{p_0(R)}\right)$ and $s_0(R) = \log\left(\frac{q_0(R)}{p_0(R)}\right)$. Having obtained an alignment with some parameters derived from an initial alignment, one may compute the scoring parameters with the new alignment and repeat the procedure until convergence.

5 Sample sessions

Here we give three simple examples of possible applications of this package. The detailed description of the functions used follows in section 7.

To access the code examples from within R, you can use the following commands:

```
> vignette(all=FALSE)
> edit(vignette("GraphAlignment"))
```

The following code configures some output options and then loads the *GraphAlignment* library.

```
> options(width = 40)
> options(digits = 3);
> library(GraphAlignment)
```

We start by generating example networks, here weighted symmetric networks.

```
> library(GraphAlignment)
> ex<-GenerateExample(dimA=22, dimB=22, filling=.5,
+     covariance=.6, symmetric=TRUE, numOrths=10, correlated=seq(1,18))
```

generates the two matrices `ex$a` and `ex$b`, both of rank 22, with random entries taken from a Gaussian distribution with mean zero and variance one. The entries of the first 18 rows and columns are pairwise correlated with covariance 0.6. To make the task harder a fraction $1 - 0.5$ of the entries has been set to zero. The adjacency matrix can be visualized with `image(ex$a)`.

A third matrix `ex$r`, a matrix of artificial node similarities with entries set equal to 0 or 1, specifies 10 sequence homologs between the two networks.

We chose an initial alignment given by the reciprocal best sequence matches

```
> pinitial<-InitialAlignment(psize=34, r=ex$r, mode="reciprocal")
```

In the present case, `pinitial` simply specifies the 10 homologs encoded in matrix r. The size of the alignment `psize=34` was chosen such that each of the $22 - 10 = 12$ nodes without a reciprocal best sequence match has a formal alignment partner in a so-called dummy node. In total $10 + 2 \times 12 = 34$ nodes are required, see section 6. The routine `InitialAlignment` will give a suitable error message if `psize` is chosen too small. The choice $psize = dimA + dimB$ is always correct, yet it is computationally the most intensive one.

The score is evaluated by binning the link weights according to a grid specified by `lookupLink`, see figure 4. A fairly wide-meshed grid is created by

```
> lookupLink<-seq(-2,2,.5)
```

The log-likelihood parameters for the link score, based on the alignment `pinitial` can be calculated using

```
> linkParams<-ComputeLinkParameters(ex$a, ex$b, pinitial, lookupLink)
```

The routine `ComputeLinkParameters` returns an estimate of the score `linkParams$ls` of interactions between distinct nodes as well as the score `linkParams$lsSelf` of self-interactions. Since, in the present example, self-interactions follow the same statistics as links between distinct nodes we use only the `linkParams$ls`. The score matrix can be plotted using `image(linkParams$ls)`.
The nodescore is computed analogously,

```
> lookupNode<-c(-.5,.5,1.5)
> nodeParams<-ComputeNodeParameters(dimA=22, dimB=22, ex$r,
+     pinitial, lookupNode)
```

The command

```
> al<-AlignNetworks(A=ex$a, B=ex$b, R=ex$r, P=pinitial,
+     linkScore=linkParams$ls,
+     selfLinkScore=linkParams$ls,
+     nodeScore1=nodeParams$s1, nodeScore0=nodeParams$s0,
+     lookupLink=lookupLink, lookupNode=lookupNode,
+     bStart=.1, bEnd=30,
+     maxNumSteps=50)
```

computes the alignment resulting from `maxNumSteps=50` iterations of the algorithm, starting from `pinitial`.

The algorithm uses a certain amount of random noise at each step in order to escape from local maxima of the score. The amplitude of the noise is parameterized by $T = 1/\beta$. The parameter T may be interpreted as an artificial temperature; taking the system slowly to low temperatures yields final states close to the global optimum. This procedure is known as simulated annealing [3]. Here, we take the inverse temperature β linearly from `bStart=.1` to `bEnd=30`. The output `al` of the algorithm gives the alignment finally reached by the algorithm.

The element i of vector `al` specifies the alignment partner j in network B of node i in network A. If $j >$ `dimB` then i has no alignment partner. Typically, all of the 18 nodes with correlated interactions are correctly aligned, the remaining 4 nodes are given no alignment partner.

The command

```
> ComputeScores(A=ex$a, B=ex$b, R=ex$r, P=al,
+     linkScore=linkParams$ls,
+     selfLinkScore=linkParams$ls,
+     nodeScore1=nodeParams$s1, nodeScore0=nodeParams$s0,
+     lookupLink=lookupLink, lookupNode=lookupNode,
+     symmetric=TRUE)

$sl
[1] 61.1

$sn
[1] 29.5
```

computes the resulting link and node scores.

```
> AnalyzeAlignment(A=ex$a, B=ex$b, R=ex$r, P=al, lookupNode,
+     epsilon=.5)

$na
[1] 17

$nb
[1] 7

$nc
[1] 0
```

returns the number of aligned nodes pairs n_a, the number of aligned node pairs where neither partner has appreciable sequence similarity with any node in the other network, n_b, and the number of aligned node pairs with no significant similarity among each other, but where one node (or both) have significant similarity with some other node, n_c. The required sequence similarity is set by `epsilon`.

In the second example we align directed binary networks. The commands

```
> ex<-GenerateExample(30, 30, filling=1, covariance=.95,
+     numOrths=20, symmetric=FALSE)
> a=ex$a;b=ex$b
> a[a>.5]=1;a[a<=.5]=0
> b[b>.5]=1;b[b<=.5]=0
> pinitial<-InitialAlignment(psize=40, r=ex$r, mode="reciprocal")
> lookupLink<-c(-.5,.5,1.5)
> linkParams<-ComputeLinkParameters(a, b, pinitial, lookupLink,
+     clamp=FALSE)
```

generate two binary networks `a,b` of size 30 with 20 homologs, an initial alignment and the 2×2 matrices of link scores. As node scores we choose [1]

```
> lookupNode<-c(-.5,.5,1.5)
> nsS0<-c(.025,-.025)
> nsS1<-c(-.025,4)
> al<-AlignNetworks(A=a, B=b, R=ex$r, P=pinitial,
+     linkScore=linkParams$ls,
+     selfLinkScore=linkParams$lsSelf,
+     nodeScore1=nsS1, nodeScore0=nsS0,
+     lookupLink=lookupLink, lookupNode=lookupNode,
+     bStart=.1, bEnd=100,
+     maxNumSteps=500, clamp=FALSE, directed=TRUE)
```

Depending on the difference between the networks (set by `covariance`) and the fraction of node pairs with sequence similarity (set by `numOrths`) the algorithm recovers the correct alignment. Exceptions are node pairs with low connectivity or a larger-than-average number of links which differ between the two networks. For details of the algorithm for directed networks, see section 7 and [1].

[1] The log-likelihood scores based on the initial alignment do not yield suitable scoring parameters in this example. In the initial alignment no nodes without sequence similarity are aligned; hence the corresponding parameters $s_0(1)$ and $s_1(0)$ are large and negative. An alternative strategy is to update the scoring parameters during the alignment procedure.

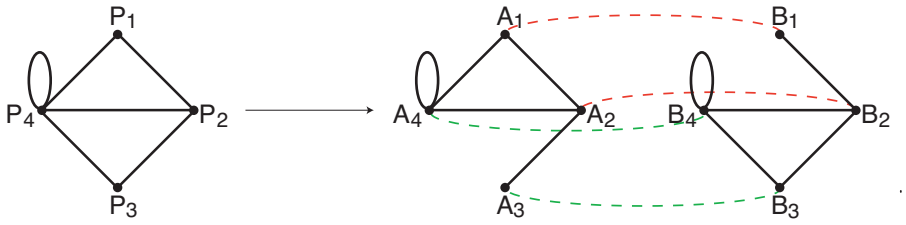


Figure 3: The ancestral (progenitor) network P has after speciation diverged into species' networks A and B. While the sequence (node) similarity of the proteins A_1 and B_1, and A_2 and B_2 is still detected (red connections), there is no such similarity for the other two nodes. We try to infer the correct relationship (green connections) only from the interaction neighbourhood (links) of the two nodes. Interaction network is considered more evolutionarily robust than the sequence. Still, some changes may have occurred: In the lineage A, the interaction (A_3, A_4) has disappeared, and in the lineage B the link (B_1, B_4) has been lost.

In the third example, two small protein interaction networks are aligned. Protein interaction networks are by their nature undirected, and discrete—with links either being present or absent. Here we align two interaction networks A and B which have descended from the common ancestral network P, see Figure 3.

There is a strong homology between sequences of A_1 and B_1, and A_2 and B_2, which allows us to state that the proteins are homologous. On the other hand, we can't find any similarity between sequences of proteins A_3, A_4 and B_3, B_4. Thus the only indices we have, are protein interactions of the four proteins. In the Figure 3, we see that A_4 and B_4 both interact with themselves and have interaction with one respective protein from the homologues pair A_2, B_2. On the contrary, the link connecting A_4 and A_1 does not have an interolog in the network B, and there is an unpaired link (B_3, B_4) in the network B. The missing interactions have been lost after the speciation event. We have to find out, if the evidence from shared links is strong enough to align the proteins A_4 with B_4 and A_3 with B_3. In order to do so, we have to (1) estimate score parameters, and (2) evaluate the optimal alignment.

We represent the networks as two matrices A and B, and store the information on homology in the similarity matrix R:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

We create these matrices by evaluating

```
> A <- matrix(c(0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1), ncol = 4);
> B <- matrix(c(0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1), ncol = 4);
> R <- matrix(c(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), ncol = 4);
```

In the next step we pick an initial guess `p0` of the mapping. One possible (random) choice is

```
> p0 <- c(1,3,4,5,2);
> psize <- 5;
```

By calling the function `AlignedPairs` we see, that the guessed proteins pairs are (A_1, B_1), (A_2, B_3), (A_3, B_4):

```
> AlignedPairs(A, B, p0);
```

```
     [,1] [,2]
[1,]    1    1
[2,]    2    3
[3,]    3    4
```

The size `psize` is chosen in such a way, that it accommodates the three mapped pairs and the other two positions for unmapped proteins (mapped to dummy nodes). The correct choice of `psize` is crucial for the speed of the algorithm only, we can always choose `psize` $= \dim(A) + \dim(B)$, which is always correct, yet it may make the algorithm slow.

To improve the mapping we have to calculate the scoring parameters for the link and node score. The links in the protein interaction network are without direction and may be either present (1) or absent (0). Thus we choose the look-up table to be:

```
> lookupLink <- c(-0.5, 0.5, 1.5);
```

The absent links fall into the first bin $(-0.5, 0.5)$, the present links into the second bin $(0.5, 1.5)$. For the node score we define the look-up table similarly, as the nodes are either homologous (1) or not (0),

```
> lookupNode <- c(-0.5, 0.5, 1.5);
```

With the specified look-up tables and the initial guess of the alignment we can calculate the node score and the link score parameters:

```
linkParams <- ComputeLinkParameters(A, B, p0, lookupLink);
nodeParams <- ComputeNodeParameters(dim(A)[1], dim(B)[1], R, p0, lookupNode);
```

The calculated values are:

```
> linkParams;

$lsSelf
       [,1]   [,2]
[1,]  0.272 -0.981
[2,] -0.981  0.811

$ls
       [,1]  [,2]
[1,]  0.287 -1.55
[2,] -1.551  1.05

> nodeParams;

$s0
[1]  0.0297 -2.3682

$s1
[1] -2.45  3.35
```

We utilise these parameters in calculation of the optimal alignment of the networks A and B:

```
> al <- AlignNetworks(A, B, R, p0, linkParams$ls, linkParams$lsSelf, nodeParams$s
```

To show that we have obtained the optimal alignment we have to show that it is self–consistent. By self–consistent we mean that the same alignment results when the link and node score parameters are inferred from the alignment itself and the mapping does not change with the new parameters. We recalculate the parameters

```
> linkParams <- ComputeLinkParameters(A, B, al, lookupLink);
> nodeParams <- ComputeNodeParameters(dim(A)[1], dim(B)[1], R, al, lookupNode);
```

After the parameters are updated, we test if the alignment `al` is the same as the alignment calculated with the new parameters:

```
> al == AlignNetworks(A, B, R, al, linkParams$ls, linkParams$lsSelf, nodeParams$s

[1] TRUE TRUE TRUE TRUE TRUE
```

Since the alignment does not change with the update of scoring parameters it is self–consistent and optimal. We can represent it either as the permutation of nodes of the network B as they match nodes of the network A, `al`, or as the list of the aligned proteins:

```
> al;

[1] 1 2 3 5 4

> AlignedPairs(A, B, al);

     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
```

The comparison of interaction networks has resulted in alignment of one pair of proteins in addition to the two pairs with sequence homology. It would not be possible to align the pair (A_3, B_3) without the knowledge of the interaction data. The last pair of proteins (A_4, B_4), which we know descend from the ancestral protein P_4, can't be aligned neither by comparison of the interaction networks. The reason is the strong penalty for two mismatching interactions. These penalties sum up to -1.75 (see `linkParams`). The rewards for one matching (conserved) interaction (A_2, A_4) and the conserved self–interaction (A_4, A_4) sum up to $0.560 + 0.329 = 0.889$, only. If the proteins A_4 and B_4 were aligned, the corresponding alignment score would have been smaller.

To estimate the strength of the alignment we evaluate its log–likelihood score.

```
> ComputeScores(A, B, R, al, linkParams$ls, linkParams$lsSelf, nodeParams$s1, node

$sl
[1] 1.63

$sn
[1] 3.17
```

The total score of the alignment has two contributions, the first coming from the sequence homology (node similarity) and the second from the similarity of interaction networks. The first contribution has value `sn` $= 3.17$, the latter one `sl` $= 1.63$. The total score equals `sn` + `sl` $= 4.8$.

6 Implementation

An alignment $\mathcal{A}$ is represented by the permutation `P`. Since some nodes may not have an alignment partner (so $\mathcal{A}$ is not one-to-one) a simple trick is used to achieve this: the permutation `P` permutes `dim` nodes, with `dim` $\geq$ `dimA`, `dim` $\geq$ `dimB`. Then node i is aligned with $j =$ `P`(i) if $j \leq$ `dimB`, and has no alignment partner if $j >$ `dimB`.

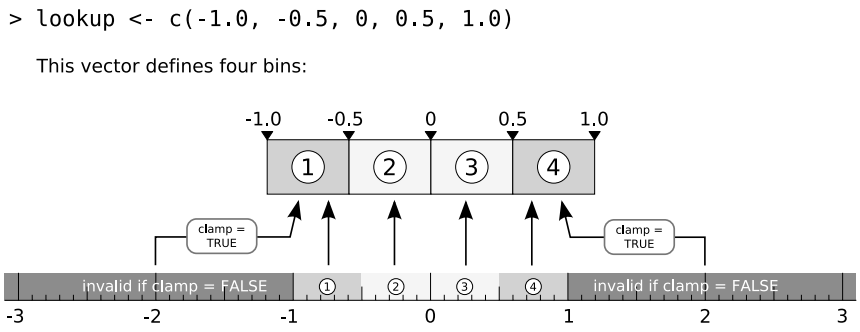


Figure 4: The binning process for link and node scores.

Intuitively, this corresponds to including 'dummy nodes' into the two networks; nodes without alignment partner are formally matched with a dummy node.

The size of the permutation `dim` must be chosen to be sufficiently large. If the value of `dim` is too small, node pairs will be aligned even though they contribute a negative score, if there are no more dummy nodes available. Hence, if the number of aligned nodes in network a equals the minimum number of aligned nodes $\texttt{dimA} + \texttt{dimB} - \texttt{dim}$ consider increasing the value of `dim`.

Score contributions are represented by lookup-tables which assign a score value to a set of bin numbers. Each score table is represented either by a vector (for node similarity scores) or a matrix (for link weight scores). Bin numbers for any input value are obtained by performing a lookup on a lookup vector. This process assigns a bin number to each valid input value and is called binning.

A lookup vector with n elements defines $n - 1$ bins. The first and last entry of the lookup vector define the lower and upper bounds of the range of valid values which can be binned. Clamping may be enabled as an option by setting `clamp=TRUE`. If clamping is enabled, values outside the lookup range are assigned to the outermost bins, thus, no real values will be considered invalid by the binning procedure. Each entry of the lookup vector other than the first and last defines a boundary between two bins. Each of these values is the upper boundary of one bin and the lower boundary of the next bin. An input value is considered to be inside a bin if it is equal or greater than the lower boundary of that bin and less than the lower boundary of the next bin. However, input values are always assigned to the last bin if they are equal to the last value in the lookup vector.

If clamping is not enabled, values outside the lookup range, that is, less than the first value of the lookup vector or greater than the last value of the lookup vector, are assumed to be invalid. Such values will cause an error to be reported if encountered during binning.

After binning has been completed, scores are obtained from a score table using the

bin number as an index. Thus, the score tables must have at least as many elements as there are bins (or combinations of bins, if the score table is a matrix).

There are four kinds of score tables which are used by functions in the package. These are `linkScore`, `selfLinkScore`, `nodeScore0` and `nodeScore1`. The link score tables are two-dimensional matrices. The node score tables are vectors. `linkScore` defines the score for link strengths between pairs of different nodes from a network. `selfLinkScore` defines the score for link strengths of links of a node to itself. `nodeScore1` defines the score for similarity between nodes from both networks which are aligned to each other. `nodeScore0` defines the score for similarity between nodes from both networks which are aligned to some other node, but not to each other.

Link strength and node similarity scores may be calculated using the `ComputeLinkParameters` and `ComputeNodeParameters` functions from the package.

7 Package Contents

7.1 Functions

This section provides an overview of the functions provided by the package and gives examples for their basic usage.

7.1.1 GenerateExample

The `GenerateExample` function creates example matrices for two networks `A`, `B` and the node similarity matrix `R`. The size of the example networks, the correlation between link strengths across the two networks, number of zero entries, as well as whether links between nodes are symmetric, can be specified as arguments. If a vector is specified as the 'correlated' argument, only the specified rows and columns of `A` and `B` will have correlated values. Leaving this argument blank will result in pairwise correlations for all entries in A, B (or, if the matrices are of different rank, all elements of the smaller matrix will be correlated with the corresponding parts of the larger matrix).

It is also possible to set the number of diagonal entries of $\mathbf{R}$ which should be set to 1 by specifying `numOrths`.

The following example creates symmetric matrices A, B of size 10 with filling 1 and covariance .5. A node similarity matrix $\mathbf{R}$ will be created, with the first 4 diagonal entries set to 1.

```
example <- GenerateExample(dimA=10, dimB=10, filling=1,
      covariance=0.5, symmetric=TRUE, numOrths=4)
a <- example$a
b <- example$b
R <- example$r
```

7.1.2 InitialAlignment

To create a random initial alignment of size `psize`, the `InitialAlignment` function can be used with the `mode` argument set to "random".

```
p <- InitialAlignment(psize=10, mode="random")
```

If `mode` is set to "reciprocal", a reciprocal best match algorithm is applied to the input matrix `R` to find an initial alignment. This mode requires that the `psize` argument is sufficiently large to allow for the addition of dummy nodes to which unaligned nodes can formally be aligned.

```
p <- InitialAlignment(psize=10, R, mode="reciprocal")
```

7.1.3 InvertPermutation

This is a convenience function for inverting a permutation which is specified by a vector.

```
pInv <- InvertPermutation(p)
```

7.1.4 Permute

This function permutes rows and columns of a matrix using the specified permutation vector. The inverse of the permutation will be applied if the `invertp` argument is set to `TRUE`.

The following example permutes the rows and columns of the matrix `b`, which has been generated with `GenerateExample` using a random initial alignment `p`. The columns of `r` are permuted using the inverse `pInv` of the alignment `p`.

```
bPerm <- Permute(b, p, invertp=TRUE)
rPerm <- R[,pInv]
```

7.1.5 GetBinNumber

This function returns the bin number corresponding to the input value. The bin number is obtained by performing a lookup in the specified lookup vector.

The lookup vector defines the lower and upper boundaries for each bin. The first entry in the lookup vector is the lower boundary of the first bin, while the last value in the lookup vector is the upper boundary of the last bin. For all other entries, entry i of the lookup vector defines the upper boundary of the $(i-1)$-th bin and the lower boundary of the i-th bin. The number of bins is therefore $n-1$, where n is the length

of the lookup vector. A lookup vector must have at least two elements.

If clamping is enabled (`clamp=TRUE`), arguments which fall below the lower boundary of the first bin are treated as if they are actually in the first bin. Likewise, values which are above the upper boundary of the last bin are treated as if they are actually in the last bin. If clamping is disabled (`clamp=FALSE`), values outside the lookup range cause an error.

```
n <- GetBinNumber(x, lookup)
```

7.1.6 VectorToBin

This function transforms a vector of arbitrary values into a vector of bin numbers corresponding to the data in the input vector. Bin numbers are found using the specified lookup table.

```
bx <- VectorToBin(x, lookup)
```

7.1.7 MatrixToBin

This function transforms a matrix of arbitrary values into a matrix of bin numbers corresponding to the data in the input matrix. Bin numbers are found using the specified lookup table.

```
bm <- MatrixToBin(m, lookup)
```

7.1.8 ComputeLinkParameters

This function computes optimal link score parameters for use with `ComputeM` and `AlignNetworks`. It takes two matrices as well as an initial alignment `p` and the lookup table for link binning, `lookupLink`, as parameters. See section 4.3 for details.

```
lookupLink <- c(-1, 0, 1)
linkParams <- ComputeLinkParameters(a, b, p, lookupLink)
linkScore <- linkParams$ls
selfLinkScore <- linkParams$lsSelf
```

7.1.9 ComputeNodeParameters

This function computes optimal node score parameters for use with `ComputeM` and `AlignNetworks`. It takes the size of the networks, a matrix of node similarities `r`, an initial alignment `p`, and the lookup table for node binning, `lookupNode`, as parameters.

See section 4.3 for details.

```
lookupNode <- c(-1, 0, 1)
nodeParams <- ComputeNodeParameters(dim(a)[1], dim(b)[1], r, p,
      lookupNode)
s0 <- nodeParams$s0
s1 <- nodeParams$s1
```

7.1.10 EncodeDirectedGraph

This function encodes an adjacency matrix for a directed graph into a symmetric matrix. Currently only binary directed graphs are implemented. The adjacency matrix of a binary directed graph has elements $0, 1$. The same graph can be represented by a symmetric adjacency matrix with elements $-1, 0, 1$, with the sign of the entry indicating the direction of the link. This is done by setting entries $m'_{ij} = m'_{ji} = 1$ if $m_{ij} = 1$ and $p[i] > p[j]$ and $m'_{ij} = m'_{ji} = -1$ if $m_{ij} = 1$ and $p[j] > p[i]$.

```
sg <- EncodeDirectedGraph(dg, 1:dim(dg)[1])
```

7.1.11 ComputeM

This function computes the score Matrix $M = M^l + M^l_{self} + M^n$ from the network adjacency matrices `a` and `b`, the node similarity matrix `r`, an alignment `p` and the node and link scores with their associated binning information. The alignment `p` is either generated by the previous iterative step, or, initially, by using `InitialAlignment`. The matrix M is then given to the linear assignment solver to compute the new alignment, see [1]

$M^l_{ij} = \sum_{k=1,k\neq j,p[k]\neq i}^{\texttt{dimA}} s^l(a_{jk}, b_{ip[k]})$

If $i >$ `dimB`, $j >$ `dimA`, or $P[k] >$ `dimB` the contribution to M^l from this sum is zero.

$(M^l_{self})_{ij} = s^l_{self}(a_{jj}, b_{ii})$

Again if $i >$ `dimB` or $j >$ `dimA` no contribution results.

$M^n_{ij} = s^n_1(R_{ji}) + \sum_{k\leq\texttt{dimA},p(k)>\texttt{dimB},k\neq j} s^n_2(R_{ki}) + \sum_{k\leq\texttt{dimB},p^{-1}(k)>\texttt{dimA},k\neq i} s^n_2(R_{jk})$

if $i \leq$ `dimB`, $j \leq$ `dimA` (zero otherwise). So in the second term, the sum over k is over all nodes in A, which are without an alignment partner and not equal to j. In the third term the sum is over all nodes in B without alignment partner and not equal to i.

```
M <- ComputeM(a, b, r, p, sl, slSelf, s0, s1, lookupLink,
      lookupNode)
```

For directed networks and the resulting asymmetric matrices, a symmetric representation is obtained by encoding the adjacency matrix as described in the documentation for `EncodeDirectedGraph`.

7.1.12 AlignNetworks

This function finds an alignment between the two input networks by repeatedly calling `ComputeM` and `LinearAssignment`, up to `maxNumSteps` times. Simulated annealing is performed if a range is specified in the `bStart` and `bEnd` arguments. This simple procedure is described in detail in [1]. Different procedures can easily be implemented by the user.

In each step, the matrix `M` is calculated from the scoring parameters and the current permutation vector `P`. The result is then normalized to the range $[-1, 1]$ and, if simulated annealing is enabled, a random matrix depending on the current simulated annealing parameters is added. The linear assignment routine is used to calculate the value of `P` which is used to compute `M` in the next step.

If the flag `directed` is set, directed binary networks are encoded by suitable symmetric matrices using `EncodeDirectedGraph`. The corresponding 3×3 matrices of the link score are computed from the 2×2 matrices given as input.

Simulated annealing is enabled if `bStart` differs from `bEnd`. In this case, a value $b_{step} = (bEnd - bStart)/(maxNumSteps - 1)$ is calculated. In step n, the random matrix which is added to `M` is scaled by the factor $1/[bStart + (n-1)b_{step}]$.

```
pr <- AlignNetworks(a, b, r, p, sl, slSelf, s0, s1,
      lookupLink, lookupNode, bStart, bEnd, maxNumSteps)
```

The returned permutation p should be read in the following way: the node i in the network A is aligned to that node in the network B which label is at the i-th position of the permutation vector p. If the label at this position is larger than the size of the network B, the node i is not aligned.

7.1.13 AlignedPairs

This function creates a matrix containing pairs of aligned nodes from networks A and B using the permutation vector P, where P is in the format returned by `AlignNetworks`.

The return value is a matrix with two columns. The number of rows is equal to the number of aligned node pairs. Each row in the matrix denotes a pair of aligned nodes. In each row, the first element (index 1) is the label of a node in network A,

and the second element (index 2) is the label of a node in network B.

```
alignedPairs <- AlignedPairs(a, b, p)
```

7.1.14 LinearAssignment

This function solves the linear assignment problem [2] defined by the input matrix. The result is a permutation of the columns of the input matrix, where $p_{result} = \text{argmin}_P (MP)$.

The following example calculates the solution for a matrix `M`. The input is scaled before it is passed to the linear-assignment solver.

```
px <- LinearAssignment(round(-1000 * (M / max(abs(M)))))
```

7.1.15 ComputeScores

This function computes scores for an alignment using the specified scoring tables, two networks `a` and `b` and their alignment `p`. The result is a list containing the link score and the node score.

```
scores <- ComputeScores(a, b, r, p, linkScore, selfLinkScore, s0, s1, lookupLink,
      lookupNode, TRUE)
```

7.1.16 AnalyzeAlignment

This function analyzes an alignment and returns various characteristics.

```
result <- AnalyzeAlignment(a, b, R, p, lookupNode, epsilon)
```

The function returns the number of aligned nodes pairs n_a, the number of aligned node pairs where neither partner has appreciable sequence similarity with any node in the other network, n_b, and the number of aligned node pairs with no significant similarity among each other, but where one node (or both) have significant similarity with some other node, n_c. The required sequence similarity is set by `epsilon`. The results are accessed by `results$na`, `results$nb`, and `results$nc`, respectively.

7.1.17 Trace

This is a convenience function to calculate the trace of a matrix.

```
t <- Trace(m)
```

7.1.18 CreateScoreMatrix

This function creates a very simple score matrix containing the product of lookup table values for each row and column as its elements. This can be used for testing purposes.

```
linkScore <- CreateScoreMatrix(lookupX, lookupY)
```

References

[1] Berg, J & Lässig, M. (2006) *Proc. Natl. Acad. Sci. USA* **103**, 10967–10972.

[2] Jonker, R & Volgenant, A. (1987) *Computing* **38**, 325–340.

[3] Kirkpatrick, S., C. D. Gelatt Jr., M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, 220, 4598, 671-680, 1983.