GraphM package: approximate graph matching algorithms

December 14, 2009

1 Problem description

A graph $G = (V, E)$ of size N is defined by a finite set of vertices $V = \{1, \dots, N\}$ and a set of edges $E \subset V \times V$. We consider weighted undirected graphs with no self-loop, i.e., all edges (i, j) have an associated positive real value $w(i, j) = w(j, i)$ and $w(i, i) = 0\ \forall i, j \in V$. Each such graph can be equivalently represented by a symmetric adjacency matrix A where $A_{ij} = w(i, j)$.

Given two graphs G and H with the same number of vertices N[1], the problem of matching G and H consists in finding a correspondence between vertices of G and vertices of H which aligns G and H in some optimal way. The correspondence between vertices may be defined by a permutation matrix P, P_{ij} is equal to 1 if the i-th vertex of G is matched to the j-th vertex of H, and 0 otherwise. After applying the permutation defined by P to the vertices of H we obtain a new graph isomorphic to H which we denote by $P(H)$. The adjacency matrix of the permuted graph, $A_{P(H)}$, is simply obtained from A_H by the equality $A_{P(H)} = PA_HP^T$.

In order to assess whether a permutation P defines a good matching between the vertices of G and those of H, a quality criterion must be defined. We focus in this paper on measuring the discrepancy between the graphs after matching of edges which are present in one graph and not in the other one:

$$F(P) = ||A_G - A_{P(H)}||_F^2 = ||A_G - PA_HP^T||_F^2\,, \tag{1}$$

where $||.||_F$ is the Frobenius matrix norm. Therefore, the problem of graph matching can be reformulated as the problem of minimization of $F(P)$ over the set of permutation matrices.

An interesting generalization of the graph matching problem is the problem of labeled graph matching. Here each graph has associated labels to all its vertices and the objective is to find an alignment that fits well graph labels and graph structures at the same time. If we let C_{ij} denote the matching score (large scores correspond to the best matchings) between the i-th vertex of G and the j-th vertex of H then the matching problem based only on label comparison can be formulated as follows

$$\max_P\ \mathrm{tr}(C^TP) = \sum_{i=1}^N \sum_{j=1}^N C_{ij}P_{ij} = \sum_{i=1}^N C_{i,P(i)}. \tag{2}$$

A natural way of unifying of (2) and (1) is a linear combination

$$\min_P\ \{(1-\alpha)F(P) - \alpha\mathrm{tr}(C^TP)\}. \tag{3}$$

In the following the term 'objectictive function $F_\alpha(P)$' will denote the last linear combination.

2 Algorithms & Parameters

The GraphM package proposes different approximate algorithms designed to solve (3). All algorithms use the linear combination parameter (3) α, this parameters is called `alpha_ldh` in the configration file *config.txt*. Some algorithms use also their own specific parameters.

1. The Umeyama algorithm.
   Originally this algorithm was proposed for weighted graph matching problem without linear term [Ume88]

$$P = \arg\max \mathrm{tr}\{|U_G|^T|U_H|P\}. \tag{4}$$

[1] Otherwise the smallest may be completed with dummy nodes.

This approach may naturally modified to include the linear term C

$$P = \arg\max \operatorname{tr}\{(1-\alpha)|U_G|^T|U_H| - \alpha C^T)P\}. \tag{5}$$

2. The Rank algorithm [RJB07]. This algorithm is based on the power method, and sometimes it does not converge, so there is a hard constraint on the numeber of iterations used in the code (1000 iteration). Usually the Rank algorithm converges, if there is a significant linear term (α is not too small).
3. The LP (Linear programming) algorithm [AS93]. This algorithm has the complexity $O(N^7)$, so it is not recomended to use it for graphs of size more than 50 vertices.
4. The QCV (Quadratic convex relaxation) algorithm [ZBV08]. Parameters: α (linear combintation (3)). This algorithm use Frank-Wolfe method for convex function minimization, the stop criterion of the FW method is defined by two parameters: `algo_fw_xeps` and `algo_fw_feps`. The stop criterion is $dx < x *$ `algo_fw_xeps` & $|df| < |f| *$ `algo_fw_feps`. Another important parameter is `hungarian_max`, it defines the integer diapason used in the hungarian method to represent the initial real valued gradient matrix. The more the the value of this parameter, the more precise the Hungarian method, the more time it takes.
5. The PATH algorithm [ZBV08]. The PATH algorithm uses the parameters of the Frank-Wolfe method defined above, and its own paramters: `qcvqcc_lambda_M` and `qcvqcc_lambda_min`. These paramteres define the behavior of adaptative path following strategy. The idea of the adaptative strategy is that the choice of *dlambda* (see the schema of the PATH algorithm [ZBV08]) is depending on the behavior of $F^{\lambda}_{\alpha}(P)$ function. If the current value of $d\lambda$ changes the function $F^{\lambda}_{\alpha}(P)$ only a little, then it is better to use larger value of $d\lambda$ to do larger steps. Or if the current $d\lambda$ changes $F^{\lambda}_{\alpha}(P)$ too much then we should decrease $d\lambda$. The minimial increment of $d\lambda$ is defined by `qcvqcc_lambda_min`, and the larger paramter `qcvqcc_lambda_M`, the larger steps are allowed.

Formally speaking, there are four other algorithms which are not true algorithms but they may be used to provide an idea about the shape of the objective function.

1. Identity matching IDEN. This algorithm returns the identity permutation.
2. Random matching RAND. This algorithm returns a random permutation matrix.
3. Uniform matching UNIF. This algorithm does not produce a permutation matrix, it returns $\frac{1}{N}1_N1_N^T$ — $N \times N$ matrix with all elements equal to 1/N. We use this algorithm as the initial point for other graph matching algorithms.

3 Common parameters

Here we describe common parameters for all graph matching algorithms. All parameters are usulually defined in a configuration file, but they may be also given in the command line. Each line of the configuration file corresponds to one parameter and has four components: parameter name, sign '=', parameter value and parameter type. There are four different parameter types: 's'—string, ' d' — double, ' i' — integer, ' c' — character.

3.1 Basic parameters

Parameter=Value Type	Description
`graph_1=../qap/m_a_1EWK s`	Adjacency matrix $N \times N$ of the first graph (ascii file)
`graph_2=../qap/m_a_1U19 s`	Adjacency matrix $M \times M$ of the second graph (ascii file)
`C_matrix=../qap/1 s`	Matrix of vertex similarities C $N \times M$ (ascii file)
`algo=U QCV RANK PATH s`	List of graph matching algorithms
`algo_init_sol=unif rand U unif s`	List of graph matching initialization algorithms. Each graph macthing algorithm may be used as an initialization algorithm, so here for example, starting points for U and PATH are given by the unif algorithm, QCV is initalized by a random matrix, and the initial point of the RANK algorithm is the solution of the Umeyama algorithm
`alpha_ldh=0.5 d`	α parameter of the linear combination (3)
`dummy_nodes=0 i`	0 — just add $\|N-M\|$ nodes to the smallest graph, 1 — add M nodes to the first graph and N nodes to the second. Depending on your problem different choices are possibles. If the problem is to find an embeding of all nodes of the smallest graph into the largest graph, so all vertices of the smallest graph should be matched to something in the largest, then you have to use 'dummy_nodes=0 i'. If you want to autorize to the vertices of the smallest graph to be matched to nothing, then 'dummy_nodes=1 i' should be used
`dummy_nodes_fill=0 d`	0 — all dummy nodes are isolated, $0 < dummy\_nodes\_fill \leq 1$ dummy nodes are connected to all others by edges with the weight const*(min_weight+max_weight). An interpretation of this parameter is the topological penality for vertces to be matched to dummy nodes.
`dummy_nodes_c_coef=0 d`	values of the C matrix associated to dummy nodes: min(C)+dummy_nodes_c_coef*(max(C)-min(C)). This parameter is used to set the vertex similarity for dummy nodes. The smaller the value of this parameter, the less preferable the association to a dummy node.
`exp_out_file=qap_out s`	Output file.
`exp_out_format=Parameters Compact Permutation s`	The format of the output file, 'Parameters'—print used parameters, 'Compact' — print only the value of the objective function for each used algorithm, 'Permutation' — print the optimal permutation. For more details see section 4
`verbose_mode=1 i`	verbose mode. 1 - on/0 - off.
`verbose_file=cout s`	cout — standard output (screen), otherwise — the name of the verbose output file

3.2 Additional parameters

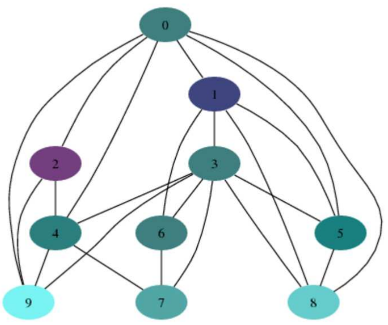
Sometimes the similarity matrix C is used to define acceptable ($C(i,j) > 0$) vertex associations, it means that all final associations $i - j$ should have a positive vertex similarity score. In order to assure that the final solution have positive vertex similarity score, we can use the following two parameters:

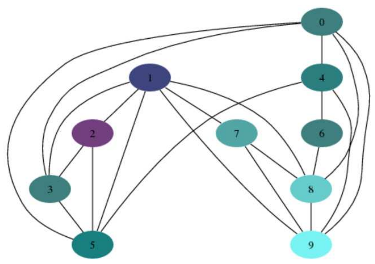
- 'blast_match_proj=1 i' means that the final solution will be projected on $P_{C>0}$ (the set of permutation matrices producing positive vertex matching scores).
- 'blast_match=1 i' restrict the initial optimization set of all permutations to $P_{C>0}$. It means that on each step of the FW algorithms a matrix from $P_{C>0}$ will be used as the new direction. In other words, not only the final solution, but also each intermediate step is projected on $P_{C>0}$.

4 Example

Let's consider a simple example. Suppose that we have two graphs G and H defined by the following adjacency matrices

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$





Suppose that we do not have any additional information about vertex similarities. In this case the configuration file `config.txt` may have the following form

```
//*********************GRAPHS**********************************
//graph_1,graph_2 are graph adjacency matrices,
//C_matrix is the matrix of local similarities  between vertices of graph_1 and graph_2.
//If graph_1 is NxN and graph_2 is MxM then C_matrix should be NxM
graph_1=../simple_test/G s
graph_2=../simple_test/H s
C_matrix=../simple_test/C s
//*******************ALGORITHMS********************************
//used algorithms and what should be used as initial solution in corresponding algorithms
algo=I U RANK QCV rand PATH s
algo_init_sol=unif unif unif unif unif unif s
solution_file=solution_im.txt s
//coeficient of linear combination between (1-alpha_ldh)*||graph_1-P*graph_2*P^T||^2_F +alpha_ldh*C_matrix
alpha_ldh=0 d
cdesc_matrix=A c
cscore_matrix=A c
C_matrix_dist=0 i
//**************PARAMETERS SECTION*****************************
hungarian_max=10000 d
algo_fw_xeps=0.01 d
algo_fw_feps=0.01 d
//0 - just add a set of isolated nodes to the smallest graph, 1 - double size
dummy_nodes=0 i
// fill for dummy nodes (0.5 - these nodes will be connected with all other by edges of weight 0.5(min_weight+max_weight))
dummy_nodes_fill=0 d
// fill for linear matrix C, usually that's the minimum (dummy_nodes_c_coef=0),
// but may be the maximum (dummy_nodes_c_coef=1)
dummy_nodes_c_coef=0.01 d

qcvqcc_lambda_M=10 d
qcvqcc_lambda_min=1e-5 d


//0 - all matching are possible, 1-only matching with positive local similarity are possible
blast_match=1 i
blast_match_proj=0 i
//****************OUTPUT***************************************
//output file and its format
exp_out_file=../simple_test/exp_out_file s
exp_out_format=Parameters Compact Permutation s
//other
graph_dot_print=1 i;
debugprint=0 i
debugprint_file=debug.txt s
```

```
verbose_mode=1 i
//verbose file may be a file or just a screen:cout
verbose_file=cout s
```

Six graph matching methods are going to be used: ‘algo=I U RANK QCV rand PATH s’.

To run the program:

```
./graphm config.txt
```

The results file `exp_out_file` may have three different parts.

If there is the word ‘Parameters’ in the ‘exp_out_format’ list then all used parameters will be listed:

```
***********************************
Experiment parameters:
graph_1=../simple_test/G
graph_2=../simple_test/H
C_matrix=../simple_test/C
algo=I U RANK QCV rand PATH
algo_init_sol=unif unif unif unif unif unif
solution_file=solution_im.txt
alpha_ldh=0
cdesc_matrix=A
cscore_matrix=A
hungarian_max=10000
algo_fw_xeps=0.01
algo_fw_feps=0.01
dummy_nodes=0
dummy_nodes_fill=0
dummy_nodes_c_coef=0.01
qcvqcc_lambda_M=10
qcvqcc_lambda_min=1e-05
blast_match=1
blast_match_proj=0
exp_out_file=../simple_test/exp_out_file
exp_out_format=Parameters Compact Permutation
graph_dot_print=1
debugprint=0
debugprint_file=debug.txt
verbose_mode=1
verbose_file=cout
```

Then if there is the word ‘Compact’, then the algorithm matching scores will be printed

```
Experiment results:
         Alpha          I               U               RANK            QCV             rand            PATH
Gdist   0.000000e+00    5.000000e+01    3.400000e+01    3.800000e+01    1.400000e+01    4.200000e+01    6.000000e+00
F_perm  0.000000e+00    5.813953e-01    3.953488e-01    4.418605e-01    1.627907e-01    4.883721e-01    6.976744e-02
F_exact 0.000000e+00    5.813953e-01    3.953488e-01    4.418605e-01    1.364375e-02    4.883721e-01    6.976744e-02
Time:                   0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
```

Here each graph matching method has four associated values: ‘Gdist’ — $||A_G - PA_HP^T||_F^2$, ‘ F_perm’ — $F_\alpha(P)$ objective function value,‘ F_exact’ — some graph matching methods like QCV produce a doubly stochastic matrix P_e (an approximation of permutation matrix) and then project it on the set of permutation matrices, so ‘ F_exact’ is the value of the objective function in P_e. Line ‘Time’ represents algorithm timing in seconds. The first column ‘ Alpha’ is the value of the linear combinaison parameter α.

Note, that because of possible numeric scaling problems, we use the normalized version of the objective function $F_\alpha(P)$. If $||A_G - PA_HP^T||_F^2$ and $\mathrm{tr}C^TP$ have completely different scales then it may be difficult to find a good α. The normalized version, that we use, has the following form

$$F_\alpha(P) = (1-\alpha)\frac{1}{||A_G||_F^2 + ||A_H||_F^2}||A_G - PA_HP^T||_F^2 + \alpha\frac{1}{||C||_F}\mathrm{tr}C^TP \tag{6}$$

The last part contains solutions (vertex matching, permutations) produced by different algorithms, this part will be printed if there is the word ‘Permutation’ in the ‘exp_out_format’ list.

```
Permutations:
I U RANK QCV rand PATH
1 2 9 2 1 2
2 10 10 6 6 9
3 4 7 8 5 3
4 1 2 1 4 1
5 3 6 9 8 6
6 8 5 3 9 10
7 7 8 7 7 7
8 5 3 5 3 5
9 6 4 4 2 8
10 9 1 10 10 4
```

Permutations produced by different algorithms are printed in columns. For example, the permutaion produced by the Umeyama algorithm 'U' is the second column $(2, 10, 4, 1, 3, 8, 7, 5, 6, 9)$. It means that the vertex number 1 of the graph `graph_1` is matched to the vertex number 2 of `graph_2`, $2 \to 10$, $3 \to 4$ and so on.

If we consider the permutation produced by the PATH algorithm then the corresponding permutation matrix has the following form

$$P_{path} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

And it can be checked that Gdist=$||A_G - P_{path}A_HP_{path}^T||_F^2$=6,F_perm=$F_\alpha(P_{path}) = \frac{1}{||A_G||_F^2+||A_H||_F^2}||A_G - P_{path}A_HP_{path}^T||_F^2$=0.0697.

Now, let's suppose that in addition to graph adjacency matrices we have the similarity matrix C_{GH}

$$C_{GH} = \begin{bmatrix} 0.50 & 0.20 & 0.60 & 0.70 & 1.00 & 0.20 & 0.30 & 0.10 & 0.30 & 0.60 \\ 0.70 & 0.60 & 0.30 & 0.90 & 0.90 & 0.10 & 0.50 & 0.50 & 0.90 & 0.60 \\ 0.10 & 0.70 & 0.90 & 0.10 & 0.00 & 0.10 & 0.30 & 0.90 & 0.40 & 0.60 \\ 1.00 & 0.20 & 0.50 & 0.00 & 0.10 & 0.30 & 0.80 & 0.30 & 0.20 & 0.20 \\ 0.30 & 0.40 & 0.80 & 0.30 & 0.60 & 1.00 & 0.40 & 0.80 & 0.10 & 0.20 \\ 0.50 & 0.50 & 1.00 & 0.30 & 0.10 & 0.80 & 0.50 & 0.50 & 0.70 & 0.60 \\ 0.60 & 0.50 & 0.40 & 0.30 & 0.10 & 0.30 & 0.80 & 0.80 & 0.50 & 0.70 \\ 0.70 & 0.00 & 0.10 & 0.60 & 1.00 & 0.30 & 0.10 & 0.10 & 0.80 & 0.80 \\ 0.60 & 0.80 & 0.30 & 0.10 & 0.50 & 0.50 & 0.70 & 0.60 & 0.90 & 0.00 \\ 0.10 & 0.40 & 0.50 & 0.20 & 0.40 & 0.20 & 0.10 & 0.50 & 0.80 & 0.60 \end{bmatrix}$$

Also we have to set up the value of parameter α, for example, `alpha_ldh=0.44`. All values may be changed in the `config.txt` or it can be defined directly in the command line without changing the configuration file

```
graphm config.txt "C_matrix=../simple_test/C_GH s;alpha_ldh=0.44 d;"
```

Contents of the output file is presented below

```
Experiment parameters:
graph_1=../simple_test/G
graph_2=../simple_test/H
C_matrix=../simple_test/C_GH
algo=I U RANK QCV rand PATH
algo_init_sol=unif unif unif unif unif unif
solution_file=solution_im.txt
alpha_ldh=0.44
cdesc_matrix=A
cscore_matrix=A
C_matrix_dist=0
hungarian_max=10000
algo_fw_xeps=0.01
algo_fw_feps=0.01
dummy_nodes=0
dummy_nodes_fill=0
dummy_nodes_c_coef=0.01
qcvqcc_lambda_M=10
qcvqcc_lambda_min=1e-05
blast_match=1
blast_match_proj=0
exp_out_file=../simple_test/exp_out_file
exp_out_format=Parameters Compact Permutation
graph_dot_print=1
debugprint=0
debugprint_file=debug.txt
verbose_mode=1
verbose_file=cout
Experiment results:
        Alpha          I              U              RANK           QCV            rand           PATH
Gdist   4.400000e-01   5.000000e+01   3.400000e+01   5.000000e+01   3.400000e+01   4.200000e+01   2.600000e+01
```

```
F_perm  4.400000e-01    -1.394961e-01   -3.158493e-01   -3.960905e-01   -4.842394e-01   -7.932901e-02   -4.962396e-01
F_exact 4.400000e-01    -1.394961e-01   -3.158493e-01   -3.960905e-01   -5.964503e-01   -7.932901e-02   -4.962396e-01
Time:                   0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00    1.000000e+00
Permutations:
I U RANK QCV rand PATH
1 2 5 10 1 2
2 10 4 4 6 4
3 4 8 8 5 8
4 1 1 1 4 1
5 3 6 6 8 6
6 6 3 3 9 3
7 7 7 7 7 7
8 5 10 5 3 5
9 8 2 2 2 9
10 9 9 9 10 10
```

In both cases the PATH algorithm gives the best approximate solution. This example can be found in `test_simple`. Other examples are presented in `test_qap` (graphs from QAP becnhmark library),`test` and `test_large` (large size graphs). In each directory you can just run `./test_script` to see how it works.

5 Installation

1. First, the GSL (GNU scientific library) should be installed, see http://www.gnu.org/software/gsl/. Usually, it can be automatically installed by system package managers, for example, `apt-get install gsl` (Debian, Ubuntu) or `yum install gsl` (Fedora, RedHat).
2. download and unpack `graphm-*.tar.gz`
3. change directory to `graphm-*`
4. run `./graphm_install`

The binary file `graphm` will be created in `bin` directory. By default, LP algorithm is not included because it needs the glpk solver. If you want to use LP algorithm, first, you have to install the glpk solver (see www.gnu.org/software/glpk/, or use a system package manager). On the last step of the installation process you should use `./graphm_install LP`.

6 Package extension

This is very easy to add your own algorithm to the package. There are three principal steps

1. Create a child class from the abstract class **algorithm** (`algorithm.h`)

   ```
   class algorithm_thebest : public algorithm
   {
   public:
       virtual match_result match(graph &g,graph &h,gsl_matrix* gm_P_i=NULL, gsl_matrix* gm_ldh=NULL,

   };
   ```

   You may find this example at `algorithm_ext.h`

2. Write your own graph matching algorithm by redefining the virtual function **match** (see `algorithm_ext.cpp`)

   ```
   match_result algorithm_thebest::match(graph& g, graph& h,gsl_matrix* gm_P_i, gsl_matrix* _gm_ldh,d
   {
   if (bverbose) *gout<<"The best matching algorithm"<<std::endl;
   match_result mres; //class with results
   gsl_matrix* gm_Ag_d=g.get_descmatrix(cdesc_matrix);//get the adjacency matrix of graph g
   gsl_matrix* gm_Ah_d=h.get_descmatrix(cdesc_matrix);//get the adjacency matrix of graph h
   //the similarity matrix C is defined in the algorithm class memeber gm_ldh
   //dalpha_ldh is corresponding to the linear combination coefficent alpha
   ```

```
//YOUR OPERATIONS  WITH MATRICES, RESULT IS A PERMUTATION MATRIX P

//do not forget to release the memory
gsl_matrix_free(gm_Ag_d);
gsl_matrix_free(gm_Ag_h);

mres.gm_P=P;//save the solution
mres.gm_P_exact=NULL; //you can save here the matrix which was used as an approximation for P


mres.dres=graph_dist(g,h,mres.gm_P,cscore_matrix);// distance between graph adjacency matrices
return mres;
}
```

3. Add `if (salgo.compare("THEBEST")==0){ return new algorith m_thebest;}` into `experiment::get_algorithm(std::string salgo)` (`experiment.h`).

4. That's all ! Now, you have to recompile the package by using `graphm_install`, and you can use your algorithm. For example, you can modify the configuration file by setting `algo=THEBEST s`.

References

[AS93] H.A. Almohamad and S.O.Duffuaa. A linear programming approach for the weighted graph matching problem. *TPAMI*, 15, 1993.

[RJB07] R.Singh, J.Xu, and B.Berger. Pairwise global alignment of protein interaction networks by matching neighborhood topology. *Research in Computantional Molecular Biology*, 4453:16–31, 2007.

[Ume88] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *Transaction on pattern analysis and machine intelligence*, 10, 1988.

[ZBV08] Mikhail Zaslavskiy, Firancis Bach, and Jean-Philippe Vert. A path following algorithm for graph matching problem. *arXiv:0801.3654v1*, 2008.