GraphM user's guide: approximate graph matching algorithms

Mikhail Zaslavskiy*†‡
mikhail.zaslavskiy@mines-paristech.fr

Francis Bach§
francis.bach@mines.org

Jean-Philippe Vert*‡
jean-philippe.vert@mines.org

September 8, 2008

1 Introduction

GraphM is a software to solve the labeled graph matching problem, i.e., finding a matching between the vertices of two graphs that conserves the structures of the graphs and matches vertices with similar vertices. This problem arises in many situations, such as the comparison of molecules, of biological or social networks, or of images. It is known to be computationally challenging, and the best matching can usually not be found in reasonable time as soon as the graphs have more than a few tens of vertices. GraphM implements several algorithms to solve approximately this problem, and allows the user to implement new algorithms. The package is written in C++ and was designed to maximally simplify the implementation of new graph matching algorithms.

GraphM is freely available under the GNU General Public License licence at http://cbio.ensmp.fr/graphm

2 Problem description

A graph $G = (V, E)$ of size N is defined by a finite set of vertices $V = \{1, \ldots, N\}$ 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, where 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 here on measuring the discrepancy between the graphs after matching by counting the number 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

*Mines ParisTech, CBIO - Centre for Computational Biology, 35 rue Saint-Honoré, 77305 Fontainebleau cedex, France
†Mines ParisTech, CMM - Centre de Morphologie Mathmatique, 35 rue Saint-Honoré, 77305 Fontainebleau cedex, France
‡Institut Curie, Centre de recherche, Biologie du developpement - U900, 26 rue d'Ulm, 75248 Paris cedex 05, France
§INRIA, Willow project, Département d'informatique, Ecole normale supérieure, 45, rue d'Ulm, 75230 Paris Cedex, France
[1]Otherwise the smallest may be completed with dummy nodes.

labels and graph structures at the same time. If we let C_{ij} denote the cost of fitness between i-th vertex of G and j-th vertex of H then the matching problem based only on label comparison can be formulated as follows

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

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

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

In the following the term 'objective function $F_\alpha(P)$' will refer to the function minimized in (3).

3 Algorithms & Parameters

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

1. The Umeyama algorithm (`U`).
   Originally this algorithm was proposed for weighted graph matching problem without linear term [Ume88], i.e., to solve:
   $$P = \arg\max \mathrm{tr}\{|U_G|^T |U_H| P\}. \tag{4}$$
   `GraphM` implements a natural extension to this method to include the linear term C and solve the following problem, which is equivalent to (3):
   $$P = \arg\max \mathrm{tr}\{(1-\alpha)|U_G|^T |U_H| + \alpha C^T)P\}. \tag{5}$$

2. The Rank algorithm (`RANK`)
   This algorithm, proposed by [RJB07], is based on the power method. It does not always converge, therefore we implemented a hard constraint on the number of iterations used in the code (1000 iteration). Usually the Rank algorithm converges, if there is a significant linear term.

3. The Linear programming method (`LP`).
   This algorithm, proposed by [AS93], has a complexity $O(N^7)$. It is therefore not recommended to use it for graphs with more than 50 vertices.

4. The quadratic convex relaxation algorithm (`QCV`).
   This method was proposed by [ZBV08]. It uses the Frank-Wolfe (FW) method for convex function minimization, the stopping criterion of the FW method being defined by two parameters: `algo_fw_xeps` and `algo_fw_feps`. The stopping criterion is $dx < x * $ `algo_fw_xeps` & $|df| < |f| *$ `algo_fw_feps`. Another important parameter is `hungarian_max`, which defines the integer diapason used in hungarian method to represent the initial real valued gradient matrix. The larger this parameter, the more precise is the Hungarian method, but the slower is its speed.

5. The PATH algorithm (`PATH`).
   This method was proposed by [ZBV08]. It uses the parameters of Frank-Wolfe method defined above, and its own parameters: `qcvqcc_lambda_M` and `qcvqcc_lambda_min`. These parameters define the behavior of adaptive path following strategy. The idea of the adaptive strategy is that the choice of $d\lambda$ (see the schema of the PATH algorithm [ZBV08]) depends on the behavior of $F_\alpha^\lambda(P)$. If the current value of $d\lambda$ changes the function $F_\alpha^\lambda(P)$ 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 $F_\alpha^\lambda(P)$ then we should decrease $d\lambda$. The minimal increment of $d\lambda$ is defined by `qcvqcc_lambda_min`. The larger the parameter `qcvqcc_lambda_M`, the larger steps are allowed.

Formally speaking there are four other algorithms which are not true algorithms but which may be used to provide an idea about the shape of 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, the returned value is $\frac{1}{N}1_N 1_N^T$, that is, the $N \times N$ matrix with all elements equal to 1/N. This algorithm is used as the starting point for other graph matching algorithms.

4 Common parameters

Here we describe common parameters for all graph matching algorithms. All parameters are usually 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 parts: parameter name, sign '=', parameter value and parameter type. There are four different parameter types: 's'—string, ' d' — double, ' i' — integer, ' c' — character.

4.1 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 matching algorithm may be used as an initialization algorithm, so here for example, initial points for U and PATH are generated by unif algorithm, QCV is initialized 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 embedding of all nodes of the smallest graph into the largest, 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 authorize 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, $0 < const \leq 1$ dummy nodes are connected to all others by edges with weight $const * (min\_weight + max\_weight)$. An interpretation of this parameter is the topological penalty for vertices to be matched to dummy nodes. The more is the value, the less is the penalty.
`dummy_nodes_c_coef=0 d`	dummy nodes associated values for the C matrix: min(C)+const*(max(C)-min(C)). This parameter is used to set the vertex similarity for dummy nodes. The less is the value of this parameter, the less preferable is the association to a dummy node.
`exp_out_file=qap_out s`	Output file for graph matching results
`exp_out_format=Parameters Compact Permutation s`	List of output results, 'Parameters'—used parameters, 'Compact' — value of objective function for each used algorithm,'Permutation' — optimal permutation. For more details see section 5
`verbose_mode=1 i`	verbose mode. 1 - on/0 - off.
`verbose_file=cout s`	cout — standard output (screen), another value — name for verbose output file

4.2 Additional parameters

Sometimes a C similarity matrix is used to define allowed ($C(i,j) > 0$) vertex associations, it means that all final associations $i-j$ should have a positive vertex similarity score. In this case next the two parameters may be useful. If we denote by $P_{C>0}$ the set of such permutations, then

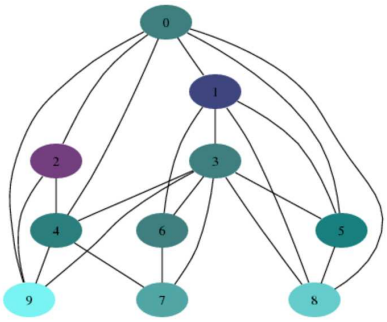
- 'blast_match_proj=1 i' means that the final solution will be projected on $P_{C>0}$.

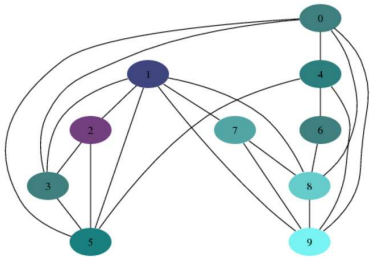
- 'blast_match=1 i' restrict the initial optimization set of all permutations to $P_{C>0}$. It means that on each step of FW algorithms a matrix from $P_{C>0}$ will be used as the new direction. In other words not only the final solution, but also each intermediate step is projected on $P_{C>0}$.

5 Example

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

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





To begin with, let us 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, just type:

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

The results file `exp_out_file` may have three different parts, depending on the words mentioned in the list `exp_out_format` of the configuration file:

- If the word `Parameters` in mentioned, then all parameters used will be listed, e.g.:

```
**********************************
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 the word `Compact` is mentioned, the different values of the objective functions reached by the different algorithms are printed, e.g.:

```
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 for 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 linear combination parameter α.
  Note, that because of possible scale problems, we use 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 alpha. It will be either near zero, or near one depending on which component is bigger. That's why we use the following normalized version

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

- Finally, if the work `Permutation` is mentioned, the solutions produced by each algorithm (i.e., vertex matching and permutations) are printed, e.g.:

```
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
```

The permutations produced by different algorithms are organized in columns. For example, the permutation produced by 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`, $2 \rightarrow 10$, $3 \rightarrow 4$ and so on.

On this example, the permutation produced by the PATH algorithm is the following:

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

It can easily be checked that the values `Gdist` and `F_perm` reached by `PATH` are coherent with this permutation, i.e., that:

$$\texttt{Gdist} = ||A_G - P_{path} A_H P_{path}^T||_F^2 = 6,$$

and

$$\texttt{F\_perm} = F_\alpha(P_{path}) = \frac{1}{||A_G||_F^2 + ||A_H||_F^2} ||A_G - P_{path} A_H P_{path}^T||_F^2 = 0.0697.$$

Now, let us suppose that, in addition to graph adjacency matrices, we have a similarity matrix C:

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

In order to match both the graph structures and the vertex similarities, we have to set up the value of parameter α, for example, `alpha_ldh=0.44`. Remember that all values can be changed directly in the `config.txt`, or can be defined in the command line without changing the configuration file, e.g.:

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

In the last case, each definition have to be followed by ';'. The output file now looks as follows:

```
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
```

An important remark is that if the similarity matrix is used (`C_matrix_dist=0 i`) then the second component is subtracted from the objective function, i.e., we solve the following problem:

$$F_\alpha(P) = (1-\alpha)\frac{1}{||A_G||_F^2 + ||A_H||_F^2}||A_G - PA_HP^T||_F^2 - \alpha\frac{1}{||C||_F}\mathrm{tr}C^TP. \tag{7}$$

In both cases (with or without similarity matrix), the `PATH` algorithm gives the best approximate solution. This example may be found in `test_simple`. Other examples are presented in `test_qap` (graphs from QAP benchmark library),`test` and `test_large` (large size graphs). In each directory you can just call `./test_script` to see how it works.

6 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) or `yum install gsl` (Fedora).
2. Download and unpack `graphm-*.tar.gz`
3. Go to `graphm` directory
4. Launch `./graphm_comp`

The executable file `graphm` will be created in `bin` directory. By default, the `LP` algorithm is not included because it needs the glpk solver. If you want to use the `LP` algorithm, you have first to install the glpk solver (see `http://www.gnu.org/software/glpk`, or use a system package manager). Then proceed as for the normal installation of `graphm`, except that on the last step of the installation process you should use `./graphm_comp LP`.

7 Package extension

It is very simple to add your own algorithm to the package if you are familiar with C++. 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 add this description to `algorithm_ext.h`

2. Write your own graph matching algorithm by redefining the virtual function **match**, this implementation should be done in `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 line `if (salgo.compare("THEBEST")==0){ return new algorith m_thebest;}` into `experiment::get_algorithm(std::string salgo)` (`experiment.h`).

4. That's all ! 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.