This code is easy to use. We have used a general format for the input ARGs, so that the users can design their own features for their tasks and easily encode these features into the ARGs for graph mining.

We have tested the code in both Windows and Linux systems.

1. How to run the demo

Please put positive images and negative images into the folder of \DAP_produce_SIVAL\ImgSet\"CategoryName", with image names 001.jpg, 002.jpg, ..., neg_001.jpg, neg_002.jpg...

Intermediate results will be stored in the .\mat folder, and the final model will be stored in the .\mat\Models folder.

```
compile; % compile mex files
featureExtraction("CategoryName"); % extract features
% in real applications, users can define their own features instead
% of using the featureExtraction() function.
produce_model("CategoryName"); % label the initial graph template
main_aog("CategoryName"); % main program for graph mining
```

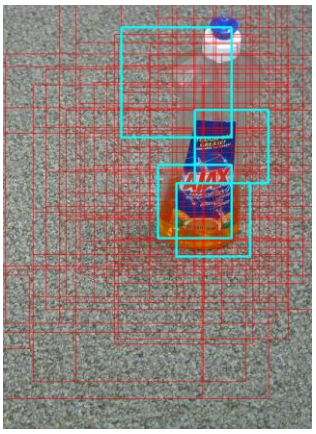
e.g.

```
compile;
featureExtraction('ajaxorange');
produce_model('ajaxorange');
main_aog('ajaxorange');
```

1.1 Label an initial template

The produce_model.m function is to label an object as the initial template

1. The system shows a figure, and a sentence "Input 'y' to label this image or input others to continue:"
2. If you want to label the object in this figure, type 'y' in the command window (go to step 3); otherwise type any other key to jump to another image (go to step 2).
3. Use the mouse to draw squares to label different parts of the object as the graph nodes in the initial model. Please do not label too many parts (please label about 3--5 parts), as the graph-mining algorithm can automatically detect new object parts, which are more reliable.



2. Support of parallel computing

You can use the command

>> matlabpool

to open some workers before running the program.
Note that main_aog.m can use all kinds of workers, but featureExtraction.m can only use local workers in your own computer.

3. Parameter settings

Please mainly focus on these parameters

In ARGSet_produce_DAP.m, you need to initialize a weight for unary attributes using

model.NodeType.unaryW=ones(1,1).*0.2;

In file of getGraphMiningParameters.m for graph mining

Para.IsFlip=true; % If Para.IsFlip=true, Setting_for_Flip () function produces two ARGs for a single positive image (one for the original image, and the other for the flipped image). Then, the main_aog() function selects the best matched one between the two ARGs as the true ARG and uses it for graph mining.

Para.PenaltyThreshold=1.0; % parameter \tau in the article

Para.Lambda=1.0; % parameter \lambda in the article

% Considering that in noisy data (e.g. web images), not all the ARGs contains the target subgraph pattern, we design the parameters, Para.StartMaxARGNumPerIteration and Para.MaxARGNumPerIteration.
% In the first iteration of graph mining, we extract the AoG using the top-15.0% positive ARGs

(i.e. the top positive ARGs with the lowest energies, which are the most reliable ARGs).
Para.StartMaxARGNumPerIteration=0.15;

% in the final iteration of graph mining, we extract the AoG using the top-30.0% positive ARGs (i.e. the top positive ARGs with the lowest energies, which are the most reliable ARGs).
Para.MaxARGNumPerIteration=0.30;

Parameters in the file "ParaSetting_ImgSet_SIVAL.m" are set for middle-level feature extraction:

4. Data format for graph mining & how to use the code with your own features

4.1 Input format:

ARGs: In the ./mat/ folder, the MAT files ARGs_"CategoryName".mat and ARGs_neg_"CategoryName".mat contain the positive ARGs and negative ARGs, respectively. Let us take the positive ARGs for example.

"ARGSet: 1 x K struct array" contains K ARGs.
For the i-th ARG,
ARGSet(i).flip: indicating that the i-th ARG represents the original image or the flipped image
ARGSet(i).arg: the i-th ARG
ARGSet(i).arg.ARGInfo.name: the name of the image that corresponds to the ARG
ARGSet(i).arg.nodeNum: the node number in the ARG; (let us assume that the i-th ARG has N nodes, i.e. ARGSet(i).arg.nodeNum=N)
ARGSet(i).arg.NodeType.type: the node type
ARGSet(i).arg.NodeType.nodeList: should be set as [1,2,...,N]
ARGSet(i).arg.NodeTypeInfo.unaryF(j).f: the j-th unary attributes (j=1,2,...), which is a (d x T) matrix. There are T nodes in the ARG. For each node, its j-th unary attribute is a d-dimensional vector.
ARGSet(i).arg.pairwiseF(j).f: the j-th pairwise attributes (j=1,2,...), which is a (d x N x N) matrix. ARGSet(i).arg.pairwiseF(j).f(:,a,b) represents the d-dimensional pairwise attribute between the a-th node and the b-th node.

Initial graph template: In the ./mat/ folder, the MAT file model_"CategoryName".mat contains the initial graph template. The data format of the initial graph template is the same as the "record(1).model_set" introduced in Section 4.2.

Note that unlike the AoG that was mined after server iterations, each OR node in initial graph template only has one terminal node.

4.2 Output format:

The MAT file in the ./mat/Models/ folder contains the results of graph mining.

"record: 1xn struct array" contains the AoG results after different number of iterations.
record(1): the initial graph template
record(2): the AoG mined after 1 iteration.
record(3): the AoG mined after 2 iterations.
record(4): the AoG mined after 3 iterations.

...

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
record(n).model_set contains the AoG model.
record(n).model_set.ARGInfo.name: the name of the image that was labeled to produce the initial template.
record(n).model_set.nodeNum: the number of the OR nodes (let us assume that there are N OR nodes in the AoG, i.e., record(n).model_set.nodeNum=N)
record(n).model_set.NodeInfo: unary attributes of terminal nodes
record(n).model_set.NodeInfo(m): unary attributes of terminal nodes under the m-th OR node (m=1,2,...,N)
record(n).model_set.NodeInfo(m).type: the node type
record(n).model_set.NodeInfo(m).unaryF(i).f: the i-th unary attributes (i=1,2,...), which is a (d x T) matrix. There are T terminal nodes under the m-th OR node. For each terminal node, its i-th unary attribute is a d-dimensional vector.
record(n).model_set.NodeType.type: the node type
record(n).model_set.NodeType.nodeList: should be set as "[1,2,...,N]"
record(n).model_set.penalty.unmatch: parameter u_none in the paper
record(n).model_set.unaryW: a (1 x N_u) vector, parameters [w_1^u,w_2^u,...,w_{N_u}^u] in the paper
record(n).model_set.pairwiseF(j).f: the j-th pairwise attributes, which is a (d x N x N) matrix. record(n).model_set.pairwiseF(j).f(:,a,b) represents the d-dimensional pairwise attribute between the a-th OR node and the b-th OR node.
record(n).model_set.pairwiseF(j).w: parameter w_j^p in the paper
record(n).model_set.penalty.unmatchPair: parameter p_none in the paper
record(n).model_set.penalty.large/largelarge: large penalties to avoid many-to-one matches.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

record(n).Log.Insert: the number of new OR nodes that have been discovered from the ARGs and been inserted into the AoG after (n-1) iterations.

record(n).Log.Delete: the number of redundant OR nodes that have been removed from the AoG after (n-1) iterations.

record(n).GoodARGList.list: the indexes of the top-x positive ARGs that are selected for graph mining in Iteration (n-1). The number of ARGs used in each iteration is controlled by parameters Para.StartMaxARGNumPerIteration and Para.MaxARGNumPerIteration (see Section 3) for details.

record(n).GoodARGList.match: the matching assignments to the positive ARGs.

4.3 Graph Mining function

DAP_main(model_set,Name_batch,ARGSet,Para,TargetDir,ARGSet_neg);
% This function supports the matlab parallel computing by using matlabpool.

model_set: the initial graph template
Name_batch: the category name
ARGSet: the set of positive ARGs
ARGSet_neg: the set of negative ARGs
Para: graph mining parameters. Para=getGraphMiningParameters(); You can set your own parameters.
TargetDir: the folder to store the AoG file. Please set TargetDir='./mat/Models';

4.4 AoG-based Graph Matching

[match,TheF1,TheF2,Psi]=getMatch(model,ARGSet,IsShown,Para);
%This function matches the AoG model "model" to ARGs in the ARG set "ARGSet".
% This function supports the matlab parallel computing by using matlabpool.

model: the AoG model;

In fact, the final AoG model is stored in the ./mat/Models/"CategoryName"_model.mat file.
You can load this file, and set "model=record(end).model_set;"
ARGSet: the set of the ARGs
IsShown : Please set IsShown=0;
Para: graph mining parameters. Para=getGraphMiningParameters();

match: the matching assignments, a (N x M) matrix
The AoG has N OR nodes, and in the "ARGSet", there are a total of M ARGs. In other words, N=model.nodeNum and M=length(ARGSet).
match(n,m): Let the value of match(n,m) be the number "v." Let the m-th ARG (1<=m<=M) have a total of V nodes, i.e., V=ARGSet(m).arg.nodeNum.
1) If v<=V, it means that the n-th OR node (1<=n<=N) in the AoG matches to the v-th node in the m-th ARG.
2) Otherwise, v=V+1, it means that the n-th OR node in the AoG matches to the choice of "invisible/none."
Psi: the choices of terminals, a (N x M) matrix
Psi(n,m): Let the value of Psi(n,m) be the number "s."
1) If s=0, it means that the n-th OR node in the AoG matches to the choice of "invisible/none."
2) Otherwise, s>=1, it means that the system choices the s-th Terminal node under the n-th OR node in the AoG to match the m-th ARG.

4. Tips

1. If you use good region proposal algorithms to provide a set of common middle-level regions as graph nodes in the ARG, the graph mining algorithm may produce good results.

2. Time cost of graph mining approximates to O(M^2 N^2), where M is the average node number in the ARGs and N is the number of the OR nodes in the AoG. If the ARG or the AoG is too large (e.g., an ARG has 1000 nodes), the mining process will cost much time.

3. Please label 3-5 parts to construct the initial graph template. The template with too many parts may have many redundant parts. The template only with two nodes may not contain sufficient information to start a stable mining process.