Graph Matching Framework

Ekaterina Tikhoncheva

Version of 26.08.2015

1 Experimental Evaluation

In this chapter we present the evaluation results of the proposed algorithm (we call it further *2LevelGM*) on some synthetic data and on some real images.

1.1 Synthetic Point set Matching

For the first test we adopted a commonly used approach of evaluation Graph Matching algorithms on the synthetic generated set of nodes (see [1], [2], [5]).
For this propose one generates first a set of n_1 normal distributed points $V_1 \subset \mathbb{R}^2$ with zero mean and standard deviation 1. The second set V_2 is created from the first one by adding noise $\mathcal{N}(0, \sigma^2)$ to the positions of points in V_1 and m additional normal distributed points with $\mathcal{N}(0, 1)$. That means, that the set V_2 consists of $n_2 = n_1 + \bar{n}$ nodes, where n_1 points are inliers and $\bar{n}$ points are outliers. The task is to find the correspondences between points in two sets.
In this test we follow the setup in [1] and compare our approach with following well known methods: *MPM* [1], *RRWM* [2], *SM* [4], *IPFP* [5]. By performing the comparison, one should consider following differences between the selected algorithms and our one.
First of all, it is time and memory consuming to perform tests for graphs with more than 200 nodes each, because of the *MPM*, *RRWM*, *SM*, *IPFP* algorithms work with the full affinity matrix of the Graph Matching Problem, whose size is equal to $n_1n_2 \times n_1n_2$.[1]
Our algorithms, however, was created to work with graphs bigger than that. To be able to perform the comparison, we fixed the number n_1 of points in the first set to 100 and vary the number of outliers $\bar{n}$ in the second set from 0 to 50.
Secondary, we need to include initialization time and time for solution discretization into time measurement of the other algorithms. That was not initially done in [1], but as *2LevelGM* does this steps almost[2] at each iteration, this change makes the comparison

[1]The affinity matrix between two graphs with 200 nodes each needs approximately $12Gb$ memory (double precision).
[2]Except the iterations where the local solution did not change

more fair. We use also greedy assignment based on [4] to discretize a solution and not Hungarian Algorithm as it was done in [1].
Thirdly, the used implementations of *2LevelGM*, *SM* and *IPFP* are purely $MATLAB$ implementations, where some steps of *MPM* and *RRWM* were written using C++. This have important influence on the running time performance of the algorithms: optimized $C++$ code can be much faster, than vectorized $MATLAB$ version ref.
We perform two (three) kinds of tests. In the first test we set number of outliers $\bar{n}$ to zero and vary only the deformation noise σ^2. We call this test *deformation test*. In the second test, *outlier test*, we do not have deformation noise ($\sigma^2 = 0.0$) and compare the behavior of the algorithms in case of increasing number of outliers $\bar{n}$. At least, in the third test, we perform Graph Matching in presence of both *outliers* and *deformation*. For this we fix deformation noise $\sigma^2 = 0.03$ and increase iteratively the number of outliers $\bar{n}$.
During the work on the topic we had tried out different modifications of the initial idea of the Two Level Graph Matching Algorithm described in the section ref. In the following we shortly describe different setups of the algorithm and present results of the described tests.

1.1.1 2LevelGM version 4.2

This version of the 2*LevelGM* uses *HEM* aggregation schema to create the Higher Level Graph. Each anchor has two types of the descriptors: appearance based descriptor and structure based descriptor (see Sec. ref for more information). We use the Algorithm 1 ref to update subgraphs after one iteration of our algorithm and following Simulated Annealing algorithm to avoid be caught in a local maximum (see Algorithm 2): each matched anchor pair (a_k, a_p) has an assigned error based on the results of the affine transformation estimation between the anchors; we randomly select one node in the first graph G^I and one in the second G^J and move them to the other anchors; this leads to changes in at most 4 subgraph in each graph; for the changed subgraphs we reestimate a affinity transformation, which gives us new error for the changed subgraphs; we accept the changes, if the error difference of the subgraphs within one graph is negative; otherwise we accept changes with some probability, when at least one subgraph has got smaller error.

```
Algorithm 2: SimulatedAnnealing
  Input: correspondence matrices between nodes of the initial
         graphs and anchors: $U^{Ia}$, $U^{Ja}$
         list of matches between the subgraphs: $m^a = \{(a_k, a_p)\}$;
         list of matches between the nodes: $m = \{(v_i, v_j)\}$;
         list of estimated affine transformations for each matched
         pair: $\{(T_{kp}, T_{pk})\}$
         current temperature $t$
  Output: updated $U^{Ia}$, $U^{Ja}$
1 $K = 3$
2 assign to each anchor $a_k \in V^{Ia}$ and $a_p \in V^{Ja}$, where $(a_k, a_p) \in m^a$ an error equal
  to $E = \min(err(T_{kp}), err(T_{pk}))$
3 randomly select one nodes $v_i$ in the first graph $G^I$; move $v_i$ from the anchor $a_k$ to
  the one of its $K$ nearest anchors: $a_{k'}$
4 randomly select one nodes $v_j$ in the second graph $G^J$; move $v_i$ from the anchor $a_p$
  to the one of its $K$ nearest anchors: $a_{p'}$
5 estimate new affine transformations between the anchors based on the this
  changes:$\{(\bar{T}_{kp}, \bar{T}_{pk})\}$
6 new error of the anchor pair $(a_k, a_p)$ is $E^{new} = \min(err(\bar{T}_{kp}), err(\bar{T}_{pk}))$
7 calculate the difference between the errors of the anchors $a_k, a_p$, affected by the
  move of the nodes $v_i$, $v_j$:
  $\Delta E_{a_k} = E^{new}_{a_k} - E_{a_k}$, $\Delta E_{a_{k'}} = E^{new}_{a_{k'}} - E_{a_{k'}}$
  $\Delta E_{a_p} = E^{new}_{a_p} - E_{a_p}$ $\Delta E_{a_{p'}} = E^{new}_{a_{p'}} - E_{a_{p'}}$
8 Are both $\Delta E_{a_k}$, $\Delta E_{a'_k}$ ($\Delta E_{a_p}$, $\Delta E_{a'_p}$) negative, accept the changes in the $G^I$ ($G^J$)
9 Is only one of the differences negative, accept the changes in $G^I$ ($G^J$) with
  probability $\exp(-\frac{\max(\Delta E_{a_k}, \Delta E_{a'_k})}{t})$ $(\exp(-\frac{\max(\Delta E_{a_p}, \Delta E_{a'_p})}{t}))$
  return updated $U^{Ia}$, $U^{Ja}$
```

The changes in the temperature depending on the iteration number *it* are defined by the function $t(it) = \frac{1}{it}$. To compensate possible error, accepted by the Simulated Annealaing, we apply afterwards again the Algorithm 1 ref.

1.1.2 2LevelGM version 4.2.1

This version of the 2*LevelGM* also uses Algorithm 1 ref to update the subgraphs, but another annealing algorithm based on [3] (see Algorithms 3, 4). The whole subgraph updating procedure is the sequnce of following steps: apply first the Algorithm 3, than Algorithm 1 ref to update subgraph and add afterwards additional distortion by using Algorithm 4.

Algorithm 3: Simulated annealing in a subgraph

```
1 calculate current global matching score $M$
2 select randomly a matched anchor $a_k \in V^{Ia}$
3 select randomly two matched nodes inside the subgraph associated with $a_k$:
  $(v_i^1, v_j^1)$ and $(v_i^2, v_j^2)$
4 delete those two matches and add two new one: $(v_i^1, v_j^2)$ and $(v_i^2, v_j^1)$
5 Calculate new global matching score $\bar{M}$
6 If $\Delta M = M - \bar{M}$ is negative accept the change, otherwise accept the change
  with probability $\exp(-\Delta M/t)$, where $t$ is the current temperature of the system
```

Algorithm 4: Simulated annealing in a graph

```
1 calculate current global matching score $M$
2 select randomly two pairs of matched nodes from the current list of matches
  $m = \{(v_i, v_j)\}$: $(v_i^1, v_j^1)$ and $(v_i^2, v_j^2)$
3 delete those two matches from $m$ and add two new one: $(v_i^1, v_j^2)$ and $(v_i^2, v_j^1)$
4 Calculate new global matching score $\bar{M}$
5 If $\Delta M = M - \bar{M}$ is negative accept the change, otherwise accept the change
  with probability $\exp(-\Delta M/t)$, where $t$ is the current temperature of the system
```

We also changed the cooling law of the temperature: $t(it) = t_0 \times 0.95^{it}$, where t_0 is initial temperature at the start of the algorithm. We set it to 200.
The results of test are represented on the Fig.1-2. The vertical sticks show the variation in the performance of the $2LevelGM$ Algorithms in all runs.

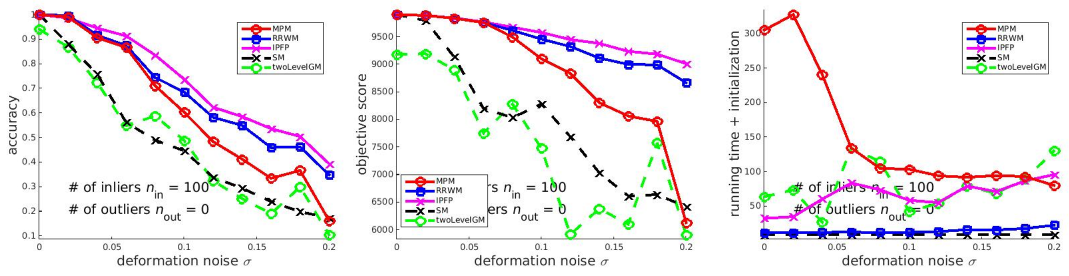


Figure 1: Deformation test: $n_1 = 100$, $n_2 = 100$, $\sigma^2 \in [0, 0.2]$

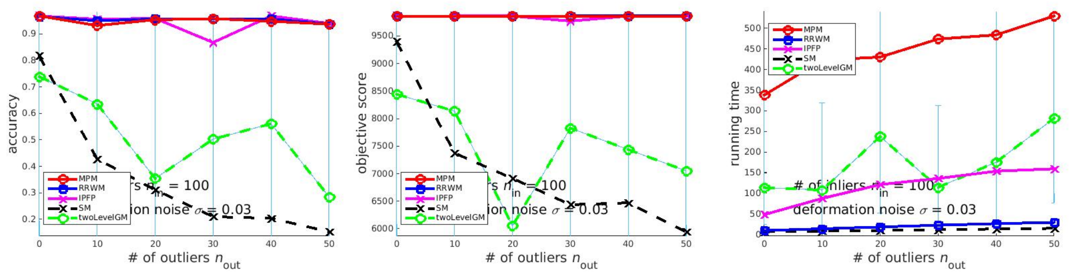
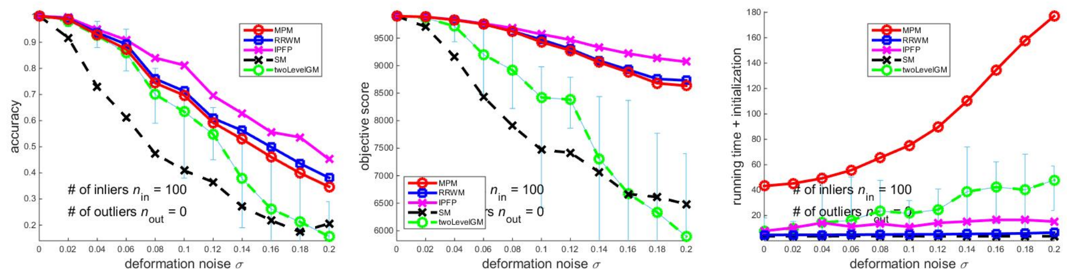


Figure 2: Average of 10 outlier tests: $n_1 = 100$, $\bar{n} \in [0, 50]$, $\sigma^2 = 0.03$

1.1.3 2LevelGM version 4.3

After some tests, we noticed, that the coarsening algorithm used previously to create Higher Level Graphs depends extremely on the graph density img. In the case of the two synthetic points sets, where all points are assumed to be fully connected, this leads to great difference between the subgraph partitions of the initial graphs at the first iteration and therefor to the divergence of the algorithm. To cope with this problem, we replaced the coarsening algorithm by the following approach: we place a grid with the fix number of rows and columns over a graph G and associate nodes inside one cell of the grid with one anchor. The anchor graph created this way describes better the coarse spatial structure of the initial graph.
Additionally to this we decline direct descriptors of the anchors and use the matching score of the two subgraphs as a similarity measure between the corresponding anchors. We also ignore the edges between the anchors.
It is important, that computation of the similarities between the anchors implies the solving Graph Matching Problem between all possible pairs of subgraphs at each iteration. This leads to the same complexity as initial one level matching problem on the whole graphs. Nevertheless we do perform the test of this version to verify the quality of the function for updating the subgraphs.
We use the slightly modified Algorithm 1 ref for updating the subgraphs between the iterations: we first perform the second rule to eliminate subgraphs with less than 4[3] nodes and then apply the rule 1 (see Sec. ref). To help the algorithm not to stay in a local maximum we use the Algorithm 4 afterwards.
The results of test can be seen on the Fig. 3- 4.

[3]Although 3 nodes are sufficient to estimate an affine transformation, but the estimated affine transformation will always overfit



(a) (b) (c)

Figure 3: Outliers test: $n_1 = 100$, $n_2 = 100$, $\sigma^2 \in [0, 0.2]$

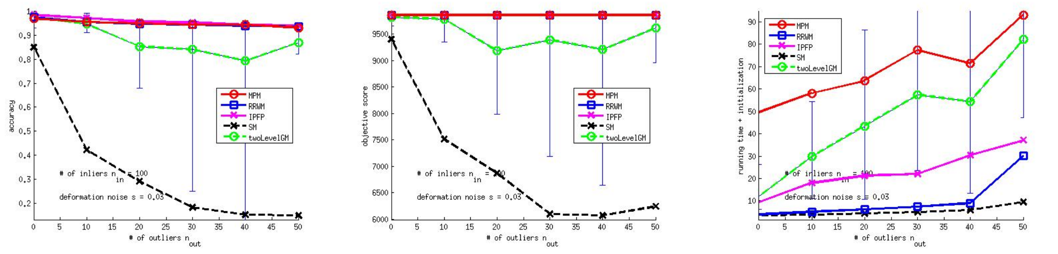


Figure 4: Average of 10 outlier tests: $n_1 = 100$, $\bar{n} \in [0, 50]$, $\sigma^2 = 0.03$

1.1.4 2LevelGM version 4.3.1

In this versions we adopt the main ideas of the previous version, but replace the Algorithm 1 ref with the new one (see Algorithm 5). This algorithm uses the estimated correspondences between the subgraphs to align them relative to each other (see Fig. Fig). Assume, that we have anchor correspondence $(a_k, a_p), a_k \in V^{Ia}, a_p \in V^{Ja}$. According to the estimated transformation between the subgraphs G^I_k, G^J_p we can place G^I_k over G^J_p in the second graph. The nodes in V^J, that are not included in G^J_p, but covered by G^I_k can be then included into the underlying subgraph based on their distance to the nodes in overlying subgraph.

Algorithm 5: UpdateSubgraphs2

```
Input: correspondence matrices between nodes of the initial
       graphs and anchors: $U^{Ia}$, $U^{Ja}$
       list of matches between the subgraphs: $m^a = \{(a_k, a_p)\}$;
       list of estimated affine transformations for each matched
       pair: $\{(T_{kp}, T_{pk})\}$
Output: new correspondence matrices $\hat{U}^{Ia}$, $\hat{U}^{Ja}$
1 Set all entries in $\hat{U}^{Ia}$, $\hat{U}^{Ja}$ to be equal $\infty$
2 foreach matched subgraph pair $(a_k, a_p)$ do
      project all nodes in $G^I_k$ according to the $T_{kp}$ into nodes of $G^J$
      foreach projection $pv_i$ of the node $v_i \in V^I_k$ do
          find its nearest neighbor $v_j \in G^J$
          $\hat{U}^{Ja}(v_j, a_p) = dist(pv_i, v_j)$
      project all nodes in $G^J_p$ according to the $T_{pk}$ into nodes of $G^I$
      foreach projection $pv_j$ of the node $v_j \in V^J_p$ do
          find its nearest neighbor $v_i \in G^I$
          $\hat{U}^{Ia}(v_i, a_k) = dist(v_i, pv_j)$
3 set in each row of $\hat{U}^{Ia}(\hat{U}^{Ja})$ the minimum element to 1 and all other to 0
4 replace the rows with the minimum element $\infty$ with the same rows in $U^{Ia}(U^{Ja})$
  return $\hat{U}^{Ia}$, $\hat{U}^{Ja}$
```

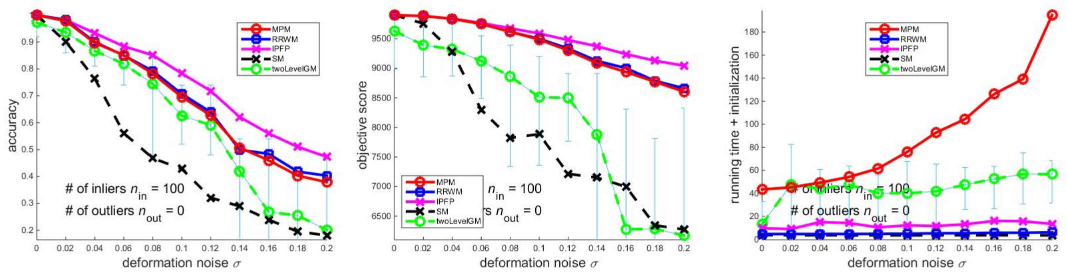


Figure 5: Deformation test: $n_1 = 100$, $n_2 = 100$, $\sigma^2 \in [0, 0.2]$

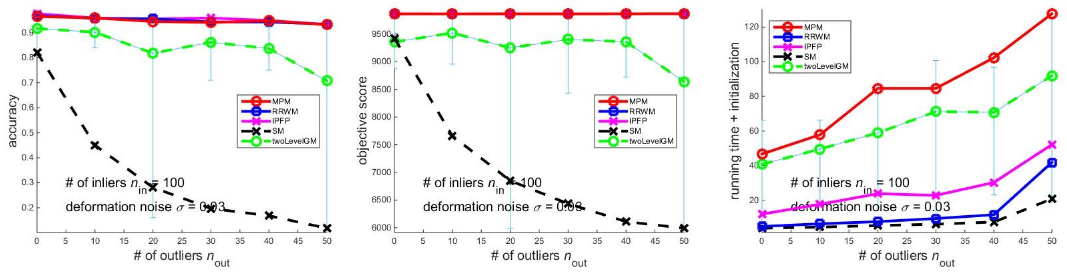


Figure 6: Average of 10 outlier tests: $n_1 = 100$, $\bar{n} \in [0, 50]$, $\sigma^2 = 0.03$

1.2 Image Affine Transformation

1.3 Real Images

References

[1] M. Cho and O. Duchenne. Finding Matches in a Haystack : A Max-Pooling Strategy for Graph Matching in the Presence of Outliers. *CVPR*, 2014.

[2] M. Cho, J. Lee, and K. M. Lee. Reweighted Random Walks for Graph Matching. *ECCV*, 2010.

[3] L. Hérault, R. Horaud, F. Veillon, and J.-J. Niez. Symbolic Image Matching by Simulated Annealing. In *4th British Machine Vision Conference (BMVC '90)*, pages 319–324, Oxford, United Kingdom, Sept. 1990. The British Machine Vision Association (BMVA).

[4] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *ICCV*, 2005.

[5] M. Leordeanu, M. Hebert, R. Sukthankar, and M. Herbert. An Integer Projected Fixed Point Method for Graph Matching and MAP Inference. In *NIPS*, 2009.