We would like to thank all reviewers and area chairs for help reviewing our submission. We are responding to the main concerns raised by reviewers and we will update our final version accordingly.

Novelty of Alg. 1 (R2). Sorry for overlooking the mentioned graduated assignment multi-graph matching works [a, b] which we find have also been cited in the references [9, 10]. We will add and discuss these two important papers in our final version to better position our work. Below we elaborate their differences and compare the two algorithms directly.

Algorithm 1: GA-MGM (ours)

Randomly initialize $\{\mathbf{U}_i\}$; projector $\leftarrow$ Sinkhorn;
while *True* **do**
  while $\{\mathbf{U}_i\}$ *not converged* **do**
    update $\mathbf{V}$: $\mathbf{V} = \lambda\mathbf{A}(\mathbf{U}\mathbf{U}^\top \circ \mathbb{M})\mathbf{A}\mathbf{U} + (\mathbf{W} \circ \mathbb{M})\mathbf{U}$;
    $\mathbf{U}_i \leftarrow \text{projector}(\mathbf{V}_i, \tau)$;
  if projector == Sinkhorn & $\tau \geq \tau_{min}$ **then**
    $\tau \leftarrow \tau \times \gamma$;
  else if projector == Sinkhorn & $\tau < \tau_{min}$ **then**
    projector $\leftarrow$ Hungarian;
  else
    break;

Output: Joint matching matrices $\{\mathbf{U}_i\}$.

Algorithm 2: GA-CL [a, b] i.e. the mentioned papers

Initialize a namely 'prototype' graph [a, b] for labeling the rest graphs: $\mathbf{U}_0 = \mathbf{I}$, and the rest graphs' matching to the 'prototype' $\{\mathbf{U}_i\}$ as random permutations;
while *True* **do**
  while $\{\mathbf{U}_i\}$ *not converged* **do**
    update $\mathbf{V}$: $\mathbf{V}[i, a, b] = \{\text{for ...}\}\times 8$;
    $\mathbf{U}_i \leftarrow \text{Sinkhorn}(\mathbf{V}_i, \tau)$;
  if $\tau \geq \tau_{min}$ **then**
    $\tau \leftarrow \tau \times \gamma$;
  else
    break;

Output: Joint matching matrices $\{\mathbf{U}_i\}$.

1) [red] First and foremost, the fundamental difference is that [a, b] require a 'prototype' graph as the anchor for labeling the nodes of the rest graphs (in their paper they use the first graph as the 'prototype'). This means they assume the bijection between each pair of graphs which is hard to satisfy in practice. In contrast, our method is fully decentralized with no such constraint, and it can therefore handle the more practical and more challenging partial matching case, i.e. only part of the nodes in each graph can find their correspondence from the other graphs. In fact, our method has one more free variable $\mathbf{U}_0$ which is fixed to an identity matrix in [a,b]. Our method shows state-of-the-art performance on CUB2011 dataset which involves partial matching, yet [a, b] are incapable for such challenging tasks by their design. **2)** [yellow] Our method is clustering-aware by introducing the additional flag variable to account for clustering (also see L6 in Alg. 1 in main paper) but [a, b] are not. **3)** [blue] Our method involves a more concise and compact formulation based on matrix form, which leads to a GPU friendly implementation. This is important to large-scale QAP problem especially for multiple graphs. This reformulation is nontrivial as shown in our supplemental material B.1 for the detailed derivatives. **4)** [green] A few technical details are rather different, e.g. we adopt Hungarian method for early discritization but the related papers contiuously use soft-assign which can be quite time consuming.

About Pascal VOC (R3). We do not compare on Pascal VOC because existing two-graph matching works are built and tested under the setting that only the common keyponits in two graphs are filtered for matching. However, such a setting does not apply to multi-graph matching, therefore our method cannot be fairly compared with existing works.

About memory consumption (R4). The memory consumption is still acceptable with our testbed (max 109 graphs, 10 nodes), however this issue may become severe if the problem size continues to scale up. We are aware of a recent work [25] passing multi-graph matching information via a spanning tree, which may mitigate the memory issue suggested by R4. And we think this issue may be beyond the scope of this paper and we would like to leave it as future work.

About convergence of Alg. 1 and Alg. 2 (R4). The convergence property of graduated assignment is discussed in details by [c]. We cannot include detailed proof on convergence due to limited space, and only major steps are discussed. We define $\mathbf{U}^k$, $\mathbf{U}^{(k+1)}$ as the (relaxed) joint matching matrix at k-th and (k+1)-th iteration respectively. By firstly fixing some $\mathbf{U}^k$, the objective function in Alg. 1 can be proved to grow like an Expectation-Maximization (EM) algorithm:

$$\begin{aligned}\lambda\,\text{tr}(\mathbf{U}^k\mathbf{U}^{k\top}\mathbf{A}\mathbf{U}^k\mathbf{U}^{k\top}\mathbf{A}) + \text{tr}(\mathbf{U}^k\mathbf{U}^{k\top}\mathbf{W}) \leq& \lambda\,\text{tr}(\mathbf{U}^{(k+1)}\mathbf{U}^{k\top}\mathbf{A}\mathbf{U}^k\mathbf{U}^{k\top}\mathbf{A}) + \text{tr}(\mathbf{U}^{(k+1)}\mathbf{U}^{k\top}\mathbf{W})\\ \leq& \lambda\,\text{tr}(\mathbf{U}^{(k+1)}\mathbf{U}^{(k+1)\top}\mathbf{A}\mathbf{U}^{(k+1)}\mathbf{U}^{(k+1)\top}\mathbf{A}) + \text{tr}(\mathbf{U}^{(k+1)}\mathbf{U}^{(k+1)\top}\mathbf{W})\end{aligned}$$

Similar conclusion holds for the multi-graph matching and clustering algorithm in Alg. 2.

Questions by R4. The set up of $\mathbf{A}_i$ follows [32]. We will include matching w/o GT in the final version if space permits.

About our contributions (R5). Eq. 1-5 are actually common formulations in this field and we list these formulations to provide our readers a general picture if they are not familiar with graph matching. Our contributions also include a graduated assignment algorithm for the more challenging multi-graph matching and clustering task (Sec. 2.2).

[a] A. Solé-Ribalta et al. Graduated assignment algorithm for finding the common labelling of a set of graphs, in *SSSPR* (2010).
[b] A. Solé-Ribalta et al. Graduated assignment algorithm for multiple graph matching based on a common labeling, *IJPRAI* (2013).
[c] A. Rangarajan et al. Convergence Properties of the Softassign Quadratic Assignment Algorithm, *Neural Computation* (1999).