8. Supplementary material

8.1. Implementation details and complexity of GASP

Update rules During the agglomerative process, the interaction between adjacent clusters has to be properly updated and recomputed, as shown in Algorithm 1. An efficient way of implementing these updates can be achieved by representing the agglomeration as a sequence of *edge contractions* in the graph. Given a graph $\mathcal{G}(V, E, w)$ and a clustering Π, we define the associated *contracted graph* $\tilde{\mathcal{G}}_\Pi(\tilde{V}, \tilde{E}, \tilde{w})$, such that there exists exactly one representative $|\tilde{V} \cap S| = 1$ for every cluster $S \in \Pi$. Edges in $\tilde{E}$ represent adjacency-relationships between clusters and the signed edge weights $\tilde{w}_e$ are given by inter-cluster interactions $\tilde{w}(e_{uv}) = \mathcal{W}_{S_u,S_v}$. For the linkage criteria tested in this work, when two clusters S_u and S_v are merged, the interactions between the new cluster $S_u \cup S_v$ and each of its neighbors depend only on the previous interactions involving S_u and S_v. Thus, we can recompute these interactions by using an *update rule* f that does not involve any loop over the edges of the original graph $\mathcal{G}$:

$$\begin{aligned}\mathcal{W}(S_u \cup S_v, S_t) &= f\Big[\mathcal{W}(S_u, S_t), \mathcal{W}(S_v, S_t)\Big] \quad (1)\\ &= f(\tilde{w}(e_{ut}), \tilde{w}(e_{vt})) \quad (2)\end{aligned}$$

In Fig. 7 we show an example of edge contraction and we list the update rules associated to the linkage criteria we introduced in Table 1.

Implementation As we show in Algorithm 2, our implementation of GASP is based on an union-find data structure and a heap allowing deletion of its elements. The algorithm starts with each node assigned to its own cluster and sorts all edges $e \in E$ in a heap/priority queue (PQ) by their absolute weight $|w_e| = |w_e^+ - w_e^-|$ in descending order, so that the most attractive and the most repulsive interactions are processed first. It then iteratively pops one edge e_{uv} from PQ and, depending on the priority $\tilde{w}_{uv}$, does the following: in case of attractive interaction $\tilde{w}_{uv} > 0$, provided that e_{uv} was not flagged as a cannot-link constraint, then merge the connected clusters, perform an edge contraction of e_{uv} in $\tilde{\mathcal{G}}_\Pi$ and update the priorities of new double edges as explained in Fig. 7. If, on the other hand, the interaction is repulsive ($\tilde{w}_{uv} \le 0$) and the option `addCannotLinkContraints` of Alg. 2 is `True`, then the edge e_{uv} is flagged as cannot-link constraint.

Complexity In the main loop, the algorithm iterates over all edges, but the only iterations presenting a complexity different from $\mathcal{O}(1)$ are the ones involving a merge of two clusters, which are at most $N-1$. By using a union-find data structure (with path compression and union by rank) the time complexity of $\texttt{merge}(u, v)$ and $\texttt{find}(u)$ operations is $\mathcal{O}(\alpha(N))$, where α is the slowly growing inverse Ackerman function. The algorithm then iterates over the neighbors of the merged cluster (at most N) and updates/deletes values in the priority queue ($\mathcal{O}(\log |E|)$). Therefore, similarly to a heap-based implementation of hierarchical agglomerative clustering, our implementation of GASP has a complexity of $\mathcal{O}(N^2 \log N)$. In the worst case, when the graph is dense and $|E| = N^2$, the algorithm requires $\mathcal{O}(N^2)$ memory. Nevertheless, in our practical applications the graph is much sparser, so $\mathcal{O}(|E|) = \mathcal{O}(N)$. With a single-linkage, corresponding to the choice of the *Maximum* update rule in our framework, the algorithm can be clearly implemented by using the more efficient Kruskal's Minimum Spanning Tree algorithm with complexity $\mathcal{O}(N \log N)$. Moreover, in the next section, we present an efficient implementation of GASP with *Absolute Maximum* linkage that has empirical $\mathcal{O}(N \log N)$ complexity.

8.2. Properties of GASP with Absolute Maximum linkage

Remark on graph notation The definition of a graph proposed by [90] makes a distinction between a set of positive edges E^+, associated with a set W^+ of positive scalar attributes representing merge affinities, and a set of negative edges E^-, associated with a set W^- of positive attributes representing split tendencies. On the other hand, in our definition $\mathcal{G}(V, E, w^+, w^-)$ each edge have both an attractive w_e^+ and a repulsive w_e^- attribute, so we can make them equivalent by defining:

$$\begin{aligned}E^+ &= \{e \in E \text{ s.t. } w_e = w_e^+ - w_e^- > 0\}, \quad (3)\\ E^- &= \{e \in E \text{ s.t. } w_e = w_e^+ - w_e^- \le 0\}, \quad (4)\\ W^+ &= \{|w_e| \text{ s.t. } e \in E^+\}, \quad (5)\\ W^- &= \{|w_e| \text{ s.t. } e \in E^-\}. \quad (6)\end{aligned}$$

Proposition 8.1. *The Mutex Watershed Algorithm 3 (MWS) with empirical $\mathcal{O}(N \log N)$ complexity introduced by [90] returns the same final clustering given by the GASP Algorithm 2 with the use of cannot-link constraints and an Absolute Maximum update rule:*

$$f_{\text{Abs.Max.}}(\tilde{w}_1, \tilde{w}_2) = \begin{cases} \tilde{w}_1 & \textit{if } |\tilde{w}_1| > |\tilde{w}_2| \\ \tilde{w}_2 & \textit{otherwise} \end{cases} \quad (7)$$

Proof. Both algorithms sort edges in descending order of the absolute interactions $|w_e|$ and then iterate over all of them. The only difference is that MWS, after merging two clusters, does not update the interactions between the new cluster and its neighbors. However, since with an Abs. Max. linkage the interaction between clusters is simply given by the edge with highest absolute weight $|w_e|$, the order by which edges are iterated over in GASP is never updated.

Algorithm 2 Implementation of GASP, generalized algorithm for signed graph partitioning

```
Input: $\mathcal{G}(V, E, w^+, w^-)$ with $N$ nodes and $M$ edges; boolean addCannotLinkConstraints
Output: Final clustering

 1: $\tilde{\mathcal{G}}(\tilde{V}, \tilde{E}) \leftarrow \mathcal{G}(V, E, w^+, w^-)$          ▷ Init. contracted graph
 2: UF $\leftarrow$ initUnionFind($V$)          ▷ Init. data structure representing clustering
 3: PQ.push($|w_e|, e$)   $\forall e \in E$          ▷ Init. priority queue in desc. order of $|w_e| = |w_e^+ - w_e^-|$, $\mathcal{O}(|E|)$
 4: canBeMerged[$e$] $\leftarrow$ True   $\forall e \in E$          ▷ Init. cannot-link constraints
 5:
 6: while PQ is not empty do
 7:   $\tilde{w}, e_{uv} \leftarrow$ PQ.popHighest()          ▷ $\mathcal{O}(\log |E|)$
 8:   assert UF.find($u$) $\neq$ UF.find($v$)          ▷ Edges in PQ always link nodes in different clusters
 9:   if ($\tilde{w} > 0$) and canBeMerged[$e_{uv}$] then
10:     PQ, canBeMerged, $\tilde{E} \leftarrow$ UPDATENEIGHBORS($u, v$)
11:     $\tilde{V} \leftarrow \tilde{V} \setminus \{v\}$,   $\tilde{E} \leftarrow \tilde{E} \setminus \{e_{uv}\}$          ▷ Update contracted graph
12:     UF.merge($u, v$)          ▷ Merge clusters, $\mathcal{O}(\alpha(|E|))$
13:   else if ($\tilde{w} \leq 0$) and addCannotLinkConstraints then
14:     canBeMerged[$e_{uv}$] $\leftarrow$ False          ▷ Constrain the two clusters
15: return Final clustering given by union-find data structure UF

 1: function UPDATENEIGHBORS($u, v$)
 2:   $\mathcal{N}_u = \{t \in \tilde{V} | e_{ut} \in \tilde{E}\}$
 3:   $\mathcal{N}_v = \{t \in \tilde{V} | e_{vt} \in \tilde{E}\}$
 4:   for $t \in \mathcal{N}_v$ do          ▷ Loop over neighbors in $\tilde{\mathcal{G}}$ of deleted node $v$
 5:     $\tilde{E} \leftarrow \tilde{E} \setminus \{e_{vt}\}$
 6:     $\tilde{w}_{vt} \leftarrow$ PQ.delete($e_{vt}$)          ▷ $\mathcal{O}(\log |E|)$
 7:     canBeMerged[$e_{ut}$] $\leftarrow$ canBeMerged[$e_{ut}$] and canBeMerged[$e_{vt}$]
 8:     if $t \in \mathcal{N}_u$ then          ▷ $t$ is a common neighbor of $u$ and $v$
 9:       $\tilde{w}_{ut} \leftarrow$ PQ.delete($e_{ut}$)          ▷ $\mathcal{O}(\log |E|)$
10:       PQ.push($|f(\tilde{w}_{ut}, \tilde{w}_{vt})|, e_{ut}$)          ▷ $\mathcal{O}(\log |E|)$
11:     else
12:       $\tilde{E} \leftarrow \tilde{E} \cup \{e_{ut}\}$
13:       PQ.push($|\tilde{w}_{vt}|, e_{ut}$)          ▷ $\mathcal{O}(\log |E|)$
14:   return PQ, canBeMerged, $\tilde{E}$
```

Algorithm 3 Mutex Watershed Algorithm proposed by [90]

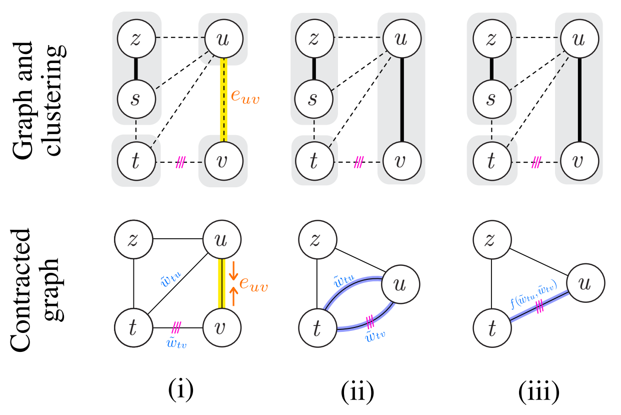
```
Input: $\mathcal{G}(V, E, w^+, w^-)$ with $N$ nodes and $M$ edges
Output: Final clustering

1: UF $\leftarrow$ initUnionFind($V$)
2: for $(u, v) = e \in E$ in descending order of $|w_e| = |w_e^+ - w_e^-|$ do
3:   if UF.find($u$) $\neq$ UF.find($v$) then          ▷ Check if $u, v$ are already in the same cluster
4:     if ($w_e > 0$) and canBeMerged($u, v$) then          ▷ Check for cannot-link constraints
5:       UF.merge($u, v$) and inherit constraints of parent clusters
6:     else if ($w_e \leq 0$) then
7:       Add cannot-link constraints between parent clusters of $u, v$
8: return Final clustering given by union-find data structure UF
```

Thus, both algorithms perform precisely the same steps and return the same clustering. □

Proposition 8.2. *The GASP Algorithm 2 with the Absolute Maximum linkage defined in Eq. 7 returns the same final clustering whether or not cannot-link constraints are enforced.*

Proof. In the GASP Algorithm 2, the clustering is updated only when two clusters are merged and the condition at line 9 is satisfied. We also observe that, in the unconstrained version of GASP, the predicate canBeMerged at line 9



Linkage criteria	Update rule f
Sum:	$f(\tilde{w}_1, \tilde{w}_2) = \tilde{w}_1 + \tilde{w}_2$
Absolute Maximum:	$f(\tilde{w}_1, \tilde{w}_2) = \begin{cases} \tilde{w}_1 & \text{if } \lvert\tilde{w}_1\rvert > \lvert\tilde{w}_2\rvert \\ \tilde{w}_2 & \text{otherwise} \end{cases}$
Average:	$f(\tilde{w}_1, \tilde{w}_2) = \text{weightAvg}\{\tilde{w}_1, \tilde{w}_2\}$
Maximum:	$f(\tilde{w}_1, \tilde{w}_2) = \max\{\tilde{w}_1, \tilde{w}_2\}$
Minimum:	$f(\tilde{w}_1, \tilde{w}_2) = \min\{\tilde{w}_1, \tilde{w}_2\}$

Figure 7: **Left:** Example of edge contraction. First row: original graph $\mathcal{G}$; clustering Π (gray shaded areas) with dashed edges on cut; cannot-link constraints (violet bars). Second row: contracted graph $\tilde{\mathcal{G}}_\Pi$. In step ii), edge e_{uv} is contracted and node v deleted from $\tilde{\mathcal{G}}_\Pi$. In step iii), double edges e_{tu} and e_{tv} resulting from the edge contraction are replaced by a single edge with updated interaction. **Right:** The table lists the update rules $f(\tilde{w}_1, \tilde{w}_2)$ associated to the linkage criteria of Table 1 and that are used to efficiently update the interactions between clusters.

can never be false because cannot-link constraints are never introduced at line 14. Let us now contradict the initial hypothesis and assume by absurd that the constrained version of GASP introduces a cannot-link constraints between two clusters sharing a positive interaction $\tilde{w} > 0$ and outputs a different clustering as compared to the unconstrained version. This can happen only in the situation shown in Fig. 8, when two clusters u and v are merged together and share a common neighboring node t having the following two properties: a) u and t are already constrained and share a repulsive interaction $w_{ut} \leq 0$, b) v and t share an attractive interaction $w_{vt} > 0$ that is higher in absolute value $|w_{vt}| > |w_{ut}|$. Then, according to Eq. 7, the new merged cluster uv and t are constrained and share a positive interaction. But this case can never happen, since if $|w_{vt}| > |w_{ut}|$ then clusters v and t are merged before clusters u and t are constrained. □

8.3. Predicting signed edge weights with a CNN

Our CNN model outputs affinities in the form of pseudo-probabilities $p : E \rightarrow [0, 1]$, where $p = 0$ represents a boundary evidence. In order to use them as input of the algorithms in our framework, we mapped them to positive and negative values[3]. The most common approaches use *additive* [1] or *logarithmic* [26, 2] mappings:

$$w_{e,\text{Add}} = p_e - \beta, \tag{8}$$

$$w_{e,\text{Log}} = \log\left(\frac{p_e}{1-p_e}\right) - \log\left(\frac{\beta}{1-\beta}\right), \tag{9}$$

where $\beta \in [0, 1]$ is a *bias* parameter that allow a tuning between over- and under-segmentation. We evaluated both of them empirically with each of the tested linkage and found that the additive mapping is the best option in all cases apart from the *Sum* linkage. Note that varying the parameter β does not usually define a hierarchy of nested clusterings, thus it is not equivalent to varying a threshold parameter in HAC. This hierarchical property is only valid for GASP without constraints and with *Average*, *Max* or *Min* linkage.

8.4. Neuron segmentation and compared methods

Training details The data from the CREMI challenge is highly anisotropic and contains artifacts like missing sections, staining precipitations and support film folds. To alleviate difficulties stemming from misalignment, we use a version of the data that was elastically realigned by the challenge organizers with the method of [80]. We train a 3D U-Net [79, 14] using the same architecture as [29] and predict long-and-short range affinities as described in [53]. In addition to the standard data augmentation techniques of random rotations, random flips and elastic deformations, we simulate data artifacts. In more detail, we randomly zero-out slices, decrease the contrast of slices, simulate tears, introduce alignment jitter and paste artifacts extracted from the training data. Both [29] and [53] have shown that these kinds of augmentations can help to alleviate issues caused by EM-imaging artifacts. We use L2 loss and Adam optimizer to train the network. The model was trained on all the three samples with available ground truth labels.

THRESH and WSDT The basic post-processing methods we consider cannot take long-range affinities into ac-

[3]Note that in general attractive and repulsive interactions w^+ and w^- can be independently estimated with different classifiers.

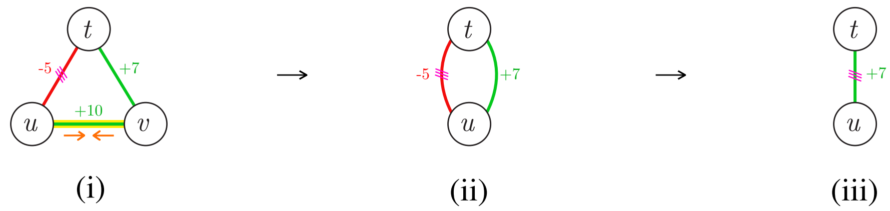


Figure 8: GASP with *AbsMax* linkage: Example representing the only case of edge contraction e_{uv} that would introduce a positive attractive interaction between two constrained clusters. Note this can actually never happen with an *AbsMax* linkage, because edge e_{ut} has a lower absolute priority as compared to e_{uv}, so clusters u and t cannot have been constrained before u and v are merged.

count, so we only consider direct neighbors affinities and generate a boundary map by taking an average over the 3 directions. Based on this boundary map, we run connected components (THRESH) and we also run a watershed algorithm seeded at the maxima of the smoothed distance transform (WSDT). For WSDT, the degree of smoothing was optimized such that each region receives as few seeds as possible, without however causing severe under-segmentation. Due to the anisotropy of the data, we generate 2D WSDT superpixels by considering each 2D image in the stack singularly.

Multi-step pipelines Given the 2D WSDT superpixels, we build a 3D region-adjacency graph such that each node represents a superpixel. The weights of the edges connecting neighboring superpixels are computed by taking an average over both short- and long-range affinities connecting the two regions. We then convert the edge probabilities to signed weights using the logarithmic mapping defined in Eq. 9 and solve the multicut problem on this graph. For our experiments, we use the approximate Kernighan-Lin solver [40, 39] (WSDT+MC). In some cases, the long-range affinities predicted by the CNN can connect two superpixels that are not direct-neighbors. Thus, in these cases we introduce additional *lifted* edges in the graph and an instance of the lifted multicut problem (WSDT+LMC). This time, similarly to the methods mentioned in [7], we used a combination of approximate solvers consisting in GAEC and Kernighan-Lin.

8.5. GASP on the full CREMI dataset

Pre-merge processing For the predictions on the full dataset from the CREMI challenge, we used the padded volumes provided by the challenge. The crops on which we performed a prediction have a size of $1500 \times 1500 \times 127 = 2.86 \cdot 10^8$ voxels or larger. Building a graph with 10^8 nodes can easily incur a large use of memory, so we decided to perform a preprocessing step by initially merging some nodes together. Simply down-sampling the predictions of the CNN would have led to a loss of resolution and performances in the most difficult parts of the dataset. Thus, we decided to pre-merge the most connected components of the graph that would be anyway clustered during the first iterations of GASP. To do this, we used a simple approach: we generated a boundary probability map by taking for each voxel an average over affinities in all directions (both short- and long-range ones) and we run THRESH with a conservative threshold parameter to find the connected components. With this approach, pixels are pre-clustered only when they are far away (in all directions) from predicted boundaries. To make sure that in this preprocessing step different neurons are never merged together by mistake, we intersected the segments given by THRESH with the segments given by WSDT. We tested GASP both on the full grid-graph and on this preprocessed graph and we did not notice any major differences in the final clustering or achieved scores, although the version with a pre-processed graph was significantly faster. To reduce the runtime and memory requirements even further, we used only 10 % of the long-range connections in the pixel-graph, since adding all of them did not improve the scores.

Removing small segments After running GASP, we use a simple post-processing step to delete small segments on the boundaries, most of which are given by single-voxel clusters. On the neuron segmentation predictions, we deleted all regions with less than 200 voxels and used a seeded watershed algorithm to expand the bigger segments.

Enforcing local merge In 2D images of urban-scenes, due to partial occlusion, one object instance can be given by multiple components that are not directly connected in the image plane. This is not the case in neuron-segmentation, where each neuron should be given by a single 3D connected component in the volume. In order to enforce it, we modified the implementation of GASP so that two clusters are merged only when they represent two adjacent supervoxels in the 3D volume and if this condition is not satisfied, the merge is postponed until there is a direct connection. This then avoids the introduction of "air-bridges" between segments due to attractive long-range connections in the initial voxel grid-graph. This approach achieved superior performances to the one proposed in [90], where all long-range connections in the grid-graph are associated to a negative repulsive edge weight.

8.6. GASP sensitivity to noise: adding artifacts to CNN predictions

Additionally to the comparison on the full training dataset, we performed more experiments on a crop of the more challenging CREMI training sample B, where we perturbed the predictions of the CNN with noise and we introduced additional artifacts like missing or fictitious boundary evidences.

In the field of image processing there are several ways of adding noise to an image, among which the most common are Gaussian noise or Poisson shot noise. In these cases, the noise of one pixel does not correlate with its neighboring noise values. On the other hand, predictions of a CNN are known to be spatially correlated. Thus, we used Perlin noise[4], one of the most common gradient noises used in procedural pattern generation. This type of noise $n(x) \in [0,1]$ generates spatial random patterns that are locally smooth but have large and diverse variations on bigger scales. We then combined it with the CNN predictions $p(x)$ in the two following ways:

$$\tilde{F}_{\pm}(x;\mathcal{K}) = F(x) \pm \left|\mathcal{K} \cdot \max\left(\pm N(x), 0\right)\right|, \qquad (10)$$

where $N(x) = \mathrm{Logit}[n(x)]$; $F(x) = \mathrm{Logit}[p(x)]$ and $\mathcal{K} \in \mathbb{R}^+$ is a positive factor representing the amount of added noise. $\tilde{F}_+(x;\mathcal{K})$ represents then a under-clustering biased prediction, such that the probability for two pixels to be in the same cluster is increased only if $N(x) > 0$ (see Fig. 9b), whereas $\tilde{F}_-(x;\mathcal{K})$ is a over-clustering biased prediction with decreased probabilities when $N(x) < 0$ (Fig. 9c). In the implementation we used, the noise can be generated in an arbitrary number of dimensions and a smoothing factor can be specified for each direction independently. In our experiments, each pixel is represented by a node in the grid-graph and it is linked to n_{nb} other nodes by short- and long-range edges. Thus, the output of our CNN model has n_{nb} channels: for each pixel / voxel, it outputs n_{nb} values representing the weights of different edge connections. We then generated a 4-dimensional noise that matches the dimension of the CNN output. The data is highly anisotropic, i.e. it has a lower resolution in one of the dimensions. Due to this fact, we chose different smoothing parameters to generate the noise in different directions.

[4]In our experiments, we used an open-source implementation of simplex noise [73], which is an improved version of Perlin noise [72]

The experiments summarized in Fig. 4 were performed in the following way: for each value $\mathcal{K}$, 30 random noise samples were drawn, from which median and percentiles statistics were computed for each different linkage criteria. For each sample, we randomly selected some of the long-range predictions from the CNN and added them to pixel grid-graph.

8.7. Fine-tuning the GMIS pipeline on CityScapes

For our experiments, we used the model from GMIS [63] that is publicly available. The model consists of two neural networks with similar structures, one predicting pixel level semantic scores and the other predicting pixel affinities between instances. We also used all the affinity post-processing methods proposed in [63], e.g. excluding background, resizing regions of interest or the proposed "affinity-refinement" method, which combines semantic and instance outputs. The instance-branch of the model was trained with a Binary Cross-Entropy loss, but we noticed how the short-range affinities were biased towards high probabilities, so that a strong short-range boundary evidence was never predicted by the model. In [63], they handle this problem by proposing a modified version of HAC that is done in stages (MultiStepHAC): initially only short-range affinities are used to run HAC and a low threshold in the hierarchy is chosen to define a first clustering; then a new HAC problem including long-range affinities is initialized with the first clustering; in the method proposed by [63], these steps are repeated three times.

Since MultiStepHAC is a rather complex post-processing method that requires to tune several hyper-parameters, we opted for a different approach to solve the problem of the unbalanced affinities. We added two 1x1 convolutional layers to the instance-branch model and trained them by using the same loss used by [90] and is based on the Sørensen-Dice coefficient [22, 85]. Compared to Hamming-distance based loss like Binary Cross-Entropy or Mean Squared Error, the advantage of this loss is its being robust against prediction and / or target sparsity, that is a desirable quality in this application since boundaries between instances can be sparse. During training, all the affinities involving at least one pixel belonging to the background were ignored in the loss. In this way, these last two layers specialized in improving the predictions of boundary evidence between adjacent instances (especially those belonging to the same class). We then considered an average of these new fine-tuned affinities with the ones predicted by the original model. During the fine-tuning process, only the parameters of the last two convolutional layers were updated.

Before to apply GASP, we performed a parameter-search for the bias β defined in 9. Table 8 lists the best-case performances for each of the methods: note that depending on

GASP linkage	CREMI-Score (higher better)	Rand-Score (higher better)	VI-merge (lower better)	VI-split (lower better)	Runtime (lower better)
Average	**0.226**	**0.936**	0.315	0.494	$3.49 \cdot 10^4$
Sum + CLC [54]	0.282	0.906	0.358	0.510	$4.64 \cdot 10^4$
Abs Max [90]	0.322	0.897	0.286	0.735	$1.24 \cdot 10^4$
Max + CLC	0.324	0.893	0.292	0.698	$6.31 \cdot 10^4$
Sum [40]	0.334	0.872	0.461	0.444	$4.74 \cdot 10^4$
Average + CLC	0.563	0.772	0.259	1.142	$2.95 \cdot 10^4$
Min	2.522	0.030	**0.197**	6.365	$2.97 \cdot 10^3$
Min + CLC	2.522	0.030	**0.197**	6.365	$4.77 \cdot 10^3$
Max	2.626	0.028	7.069	**0.026**	$\mathbf{6.04 \cdot 10^2}$

Table 7: Performances achieved by different versions of GASP on the CREMI 2016 training set. CREMI-Score [28], is given by a combination of the Adapted Rand-Score (Rand-Score) and the Variation of Information Score for under-clustering (VI-merge) and over-clustering (VI-split) [4]. CLC stands for cannot-link constraints. For all algorithms, the chosen value of bias parameter was $\beta = 0$. We used a machine with CPU Intel(R) Xeon(R) E7-4870 @ 2.40GHz for our comparison experiments.

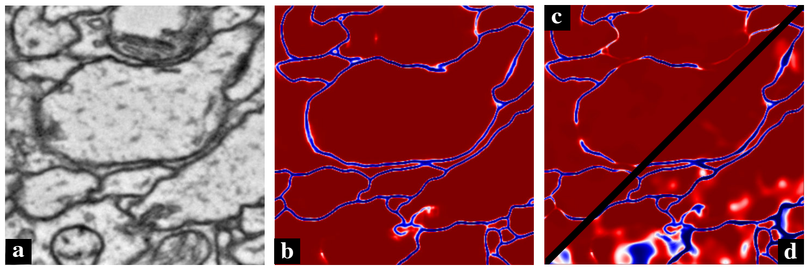


Figure 9: CNN predictions on a slice of the CREMI neuron segmentation challenge with and without additional noise. (**a**) Raw data (**b**) Original CNN predictions $F(x)$, where blue pixels represent boundary evidence (**c**) Under-clustering biased version $\tilde{F}_+(x;\mathcal{K})$ of the predictions defined in Eq. 10 with $\mathcal{K} = 8$ (**d**) Over-clustering biased version $\tilde{F}_-(x;\mathcal{K})$. Long-range predictions are not shown.

GASP linkage	AP	Bias β
Average	34.3	0.35
Average + CLC	33.9	0.25
Max + CLC	32.5	0.50
Abs Max	32.1	0.45
Sum + CLC	31.9	0.55
Sum	31.3	0.55
Max	24.3	0.85
Min	0.00	0.50
Min + CLC	0.00	0.50

Table 8: Average Precision (AP) scores achieved by different versions of GASP and chosen bias parameters β on the cityscapes validation set. A bias value $\beta = 0$ returns one single cluster. CLC stands for cannot-link constraints

the GASP linkage criterion, it was necessary to bias more or less the predicted edge weights.

The semantic categories are assigned to each instance in the same way proposed by [63], i.e. with a majority vote based on the semantic output of the model.