A6. Appendix

A6.1. Implementation 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 node $|\tilde{V} \cap S| = 1$ 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 $\tilde{w}(e_{uv}) = \mathcal{W}_{S_u \cup S_v}$, where S_u denotes the clustering including node u. For the linkage criteria tested in this article, 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 $\mathcal{G}$:

$$\mathcal{W}(S_u \cup S_v \cup S_t) = f\Big[\mathcal{W}(S_u \cup S_t), \mathcal{W}(S_v \cup S_t)\Big] \quad (5)$$

$$= f(\tilde{w}(e_{ut}), \tilde{w}(e_{vt})) \quad (6)$$

In Fig. A6 we show an example of edge contraction and in Table A4 we list the update rules associated to the linkage criteria we introduced in Table 1.

Implementation Our implementation of GASP is based on an union-find data structure and a heap allowing deletion of its elements. In Phases 2 and 3, GASP is equivalent to a standard hierarchical agglomerative clustering algorithm with complexity $\mathcal{O}(N^2 \log N)$. In Algorithm 2, we show our implementation of phase 1, involving cannot-link constraints. In phase 1, 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, 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. A6. If, on the other hand, the interaction is repulsive ($\tilde{w}_{uv} \leq 0$) and the option `addCannotLinkContraints` of Alg. 2 is `True`, then the edge e_{uv} is flagged as cannot-link constraint.

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:	$f(\tilde{w}_1, \tilde{w}_2) = \text{weightAvg}\{\tilde{w}_1, \tilde{w}_2\}$
Single:	$f(\tilde{w}_1, \tilde{w}_2) = \max\{\tilde{w}_1, \tilde{w}_2\}$
Complete:	$f(\tilde{w}_1, \tilde{w}_2) = \min\{\tilde{w}_1, \tilde{w}_2\}$

Table A4. 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.

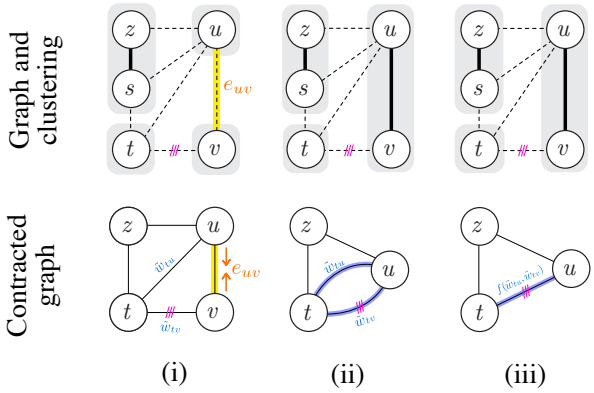

Figure A6. Example of edge contraction. First row: original graph $\mathcal{G}$; clustering Π (gray shaded areas) with dashed edges on cut; cannot-link constraints (violet bars). Second row: contracted graph $\tilde{\mathcal{G}}_\Pi$. In step ii), edge e_{uv} is contracted and node v deleted from $\tilde{\mathcal{G}}_\Pi$. In step iii), double edges e_{tu} and e_{tv} resulting from the edge contraction are replaced by a single edge with updated interaction.

Complexity In the main loop of Phase 1, 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 `merge`(u, v) and `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 - Phase 1 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

Algorithm 2 Implementation of GASP - Phase 1

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

$\tilde{\mathcal{G}}(\tilde{V}, \tilde{E}) \leftarrow \mathcal{G}(V, E, w^+, w^-)$ ▷ Init. contracted graph
`UF` $\leftarrow$ initUnionFind(V) ▷ Init. data structure representing clustering
PQ.push($|w_e|, e$) $\quad \forall e \in E$ ▷ Init. priority queue in desc. order of $|w_e| = |w_e^+ - w_e^-|$, $\mathcal{O}(|E|)$
`canBeMerged`$[e] \leftarrow$ `True` $\quad \forall e \in E$ ▷ Init. cannot-link constraints

while PQ is **not** empty **do**
  $\tilde{w}, e_{uv} \leftarrow$ PQ.popHighest() ▷ $\mathcal{O}(\log |E|)$
  assert `UF`.find(u) $\neq$ `UF`.find(v) ▷ Edges in PQ always link nodes in different clusters
  if ($\tilde{w} > 0$) **and** `canBeMerged`$[e_{uv}]$ **then**
    PQ, `canBeMerged`, $\tilde{E} \leftarrow$ UpdateNeighbors(u, v)
    $\tilde{V} \leftarrow \tilde{V} \setminus \{v\}, \quad \tilde{E} \leftarrow \tilde{E} \setminus \{e_{uv}\}$ ▷ Update contracted graph
    `UF`.merge(u, v) ▷ Merge clusters, $\mathcal{O}(\alpha(|E|))$
  else if ($\tilde{w} \leq 0$) **and** `addCannotLinkConstraints` **then**
    `canBeMerged`$[e_{uv}] \leftarrow$ `False` ▷ Constrain the two clusters
return Final clustering given by union-find data structure `UF`

function UpdateNeighbors(u, v)
  $\mathcal{N}_u = \{t \in \tilde{V} | e_{ut} \in \tilde{E}\}$
  $\mathcal{N}_v = \{t \in \tilde{V} | e_{vt} \in \tilde{E}\}$
  for $t \in \mathcal{N}_v$ **do** ▷ Loop over neighbors in $\tilde{\mathcal{G}}$ of deleted node v
    $\tilde{E} \leftarrow \tilde{E} \setminus \{e_{vt}\}$
    $\tilde{w}_{vt} \leftarrow$ PQ.delete(e_{vt}) ▷ Delete edge e_{vt} from PQ and get the old edge weight, $\mathcal{O}(\log |E|)$
    `canBeMerged`$[e_{ut}] \leftarrow$ `canBeMerged`$[e_{ut}]$ **and** `canBeMerged`$[e_{vt}]$
    if $t \in \mathcal{N}_u$ **then** ▷ Check if t is a common neighbor of u and v
      $\tilde{w}_{ut} \leftarrow$ PQ.delete(e_{ut}) ▷ $\mathcal{O}(\log |E|)$
      PQ.push($|f(\tilde{w}_{ut}, \tilde{w}_{vt})|, e_{ut}$) ▷ $\mathcal{O}(\log |E|)$
    else
      $\tilde{E} \leftarrow \tilde{E} \cup \{e_{ut}\}$
      PQ.push($|\tilde{w}_{vt}|, e_{ut}$) ▷ $\mathcal{O}(\log |E|)$
  return PQ, `canBeMerged`, $\tilde{E}$

Algorithm 3 Mutex Watershed Algorithm proposed by [75]

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

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

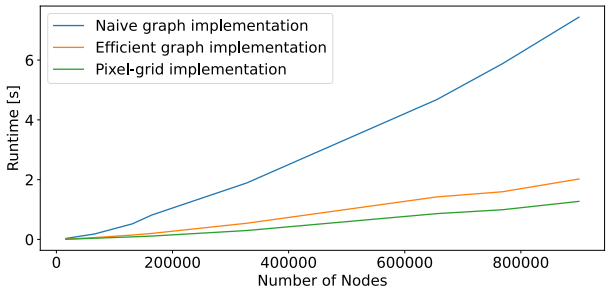

Figure A7. Runtimes for different implementation of GASP with AbsMax linkage criterion. Runtimes are averaged over 5 runs.

$\mathcal{O}(|E|) = \mathcal{O}(N)$. With a single-linkage, corresponding to the choice of the *Maximum* update rule in our framework, the algorithm can be implemented by using the more efficient Kruskal's Minimum Spanning Tree algorithm with complexity $\mathcal{O}(N \log N)$, but only when cannotLinkConstraints are not used. Moreover, GASP with *Absolute Maximum* linkage can be implemented more efficiently (see next section).

Efficiency of different GASP implementations with AbsMax linkage criteria In Fig. A7, we compare the runtimes of three implementations of the AbsMax criteria: the implementation from [74] for pixel graphs (*Pixel-grid implementation*) and for general graphs (*Efficient graph implementation*) as well as the HC implementation with AbsMax linkage (*Naive graph implementation*). The specialized implementations can exploit the properties of the underlying graph and are faster. But our generalization does not carry a large computational penalty and only requires a few extra seconds for partitioning graphs of a million nodes. Note that we have always used the most efficient implementation for the results reported in the paper. We will clarify this fact.

Median linkage We implemented median linkage in our library from the beginning but did not report on it in the main paper for two reasons: we consider the other criteria to span the range of interesting behavior well; and it performs no better than some of the other criteria (like average linkage) which are faster to evaluate.

A6.2. GASP relation to the multicut objective

For some of the linkage criteria, e.g. sum and average, GASP can be understood as a local search to the objective of the multicut optimization problem 1, see [51]. But this does not hold in general: the Abs Max linkage for example does not always decrease the MC objective (see counter example in Fig. A8). Moreover, GASP cannot be seen as a k-approximation, because it is a polynomial algorithm and

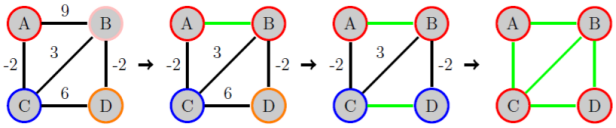

Figure A8. GASP agglomeration with the Abs Max criterion: contracted edges are marked green. The last contraction increases the MC objective from -1 to 0.

Chawla, et al. Computational complexity, 2006 has shown that approximating the multicut objective with any constant factor is in itself NP-hard.

A6.3. Proofs of Propositions 3.1, 3.2, A6.1, and 3.3

Lemma A6.1. *If GASP Algorithm 1 with* ***Complete linkage criteria*** *enforces a constraint between two clusters in* Phase 1*, then the interaction between the clusters will never become positive over the course of the following agglomeration steps.*

Proof. Two clusters are constrained in *Phase 1* only if their interaction is repulsive and, with complete linkage, the signed interaction between two clusters can only decrease over the course of the agglomeration. Thus, if two clusters are constrained by the algorithm, their negative interaction cannot increase and become positive later on in the agglomeration process. □

Lemma A6.2. *If GASP Algorithm 1 with* ***AbsMax linkage criteria*** *enforces a constraint between two clusters in* Phase 1*, then the interaction between the clusters will never become positive over the course of the following agglomeration steps.*

Proof. During the agglomeration the interaction between two clusters can only increase in absolute value. Thus, the negative interaction $\mathcal{W}(S_i \cup S_j) < 0$ between two constrained clusters can possibly become positive over the course of next agglomeration steps only if there is at least another pair of clusters in the graph that has a positive interaction $\mathcal{W}(S_l \cup S_t) > 0$ higher in absolute value: $|\mathcal{W}(S_l \cup S_t)| > |\mathcal{W}(S_i \cup S_j)|$. If such clusters S_l, S_t with positive interaction exist, we note that they must also be constrained (in the opposite case, the algorithm would have already merged them before to constrain S_i and S_j, because their priority is higher). In other words, a constrained negative interaction can become positive only if there is already another positive constrained interaction: but this can never be the case because initially all constrained interactions are negative. □

Lemma A6.3. *In the GASP Algorithm 1 with AbsMax or Complete linkage criteria (see linkage definition in Table 1),*

the same final clustering is returned whether or not cannot-link constraints are enforced.

Proof. In phase 1 of Algorithm 2, two clusters are merged only if the condition at line 9 is satisfied (i.e. when an interaction is both positive and not constrained). From Lemma A6.2 and Lemma A6.2 follows that with Complete and AbsMax linkage an interaction can never be both positive and constrained at the same time, so we directly conclude that the constrained and unconstrained versions of the algorithm will perform precisely the same agglomeration steps in phase 1. In phase 2 (after constraints have been removed) no clusters are merged because all interactions are already negative (whether they previously constrained or not). Thus, both constrained and unconstrained versions of GASP return the same clustering Π^*. □

Proposition 3.1. *The GASP Algorithm 1 with AbsMax linkage, with or without cannot link constraints, returns the same final clustering* Π^*_{AbsMax} *also returned by the Mutex Watershed Algorithm (MWS) [75], which has empirical complexity* $\mathcal{O}(N \log N)$.

Proof. From Lemma A6.3 it directly follows that GASP with AbsMax linkage criterion returns the same final clustering whether or not cannot-link constraints are enforced. In the following, we prove that MWS (see pseudocode 3) and the constrained AbsMax version of GASP also return the same clustering. 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. Thus, both algorithms perform precisely the same steps and return the same clustering. □

Proposition 3.2. *We call an agglomerative algorithm "weight-shift invariant" if the dendrogram* T *returned by the algorithm is invariant w.r.t. a shift of all edge weights* w_e *by a constant* $\alpha \in \mathbb{R}$. *Among the variations of GASP, only hierarchical clustering with Average (HC-Avg), Single (HC-Single), and Complete linkage (HC-Complete) are weight-shift-invariant (see green box in Table 1).*

Proof. Theorem 1 in [12] proves that hierarchical clustering with Average (HC-Avg), Single (HC-Single), and Complete linkage (HC-Complete) are weight-shift invariant.

The same is not true for GASP with Sum linkage criteria (GAEC and HCC-Sum), because by adding a constant α to all edge weights w_e, the interaction between two clusters S_i and S_j is increased by a factor $\alpha|E_{ij}|$, which depends on the number of edges $|E_{ij}|$ connecting the two clusters.

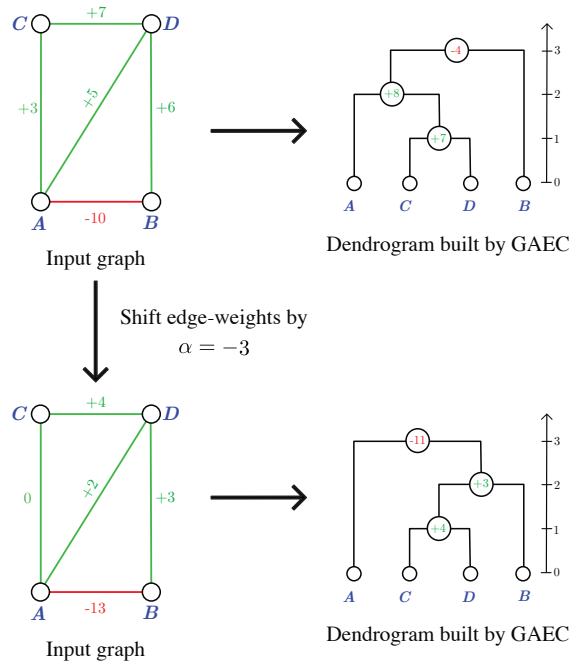

Figure A9. Counter-example showing that GAEC is not weight-shift invariant.

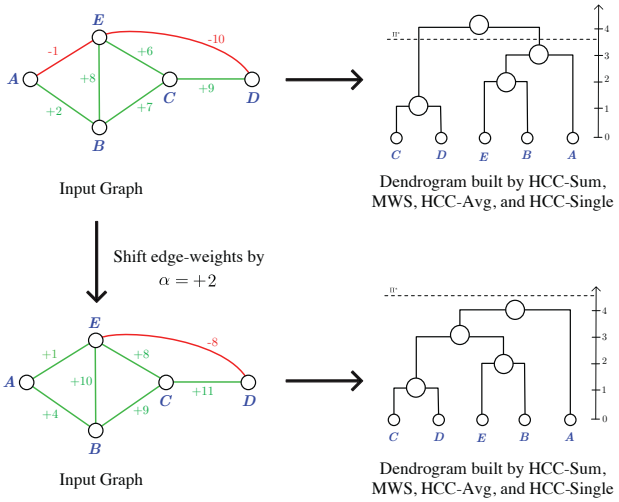

Figure A10. Counter-example showing that HCC-Sum, MWS, HCC-Avg, and HCC-Single are not weight-shift invariant.

Thus, when all edge weighs of the graph are shifted, the agglomeration order may change. For a simple example of this, it is enough to consider the toy graph in Fig. 1a and shift the weights of the graph by $\alpha = -3$ (see Fig. A9).

The constrained versions of GASP (HCC-Avg and HCC-Single) are also not weight-shift invariant: here, the algorithm merges or constrains clusters in a given order, de-

pending on the absolute interactions $|\mathcal{W}(S_i \cup S_j)|$ between clusters; so, when edge weights are shifted by a constant α, the sorting by absolute value can change arbitrarily together with the agglomeration order, as we show in the counter-example of Fig. A10. Similarly, the Mutex Watershed algorithm is not weight-shift invariant because it uses a linkage criterion that compares weights by their absolute values (see again counter-example in Fig. A10. □

Proposition A6.1. *Consider a graph $\mathcal{G}(V, E, w_e)$, a linkage criterion $\mathcal{W}$, and an agglomerative algorithm returning a binary rooted tree T with height h_T. Then, (V, d_T) defined in Eq. 3 is an ultrametric if and only if the following is true:*

$$\forall u, v, t \in V$$
$$h_T(u, v) < h_T(u, t) \Rightarrow \mathcal{W}_T(u, v) \geq \mathcal{W}_T(u, t) \quad (7)$$

In words, condition 7 means: if the algorithm merges nodes u, v before to merge nodes u, t, then the signed interaction $\mathcal{W}_T(u, v)$ between u and v has to be higher or equal than $\mathcal{W}_T(u, t)$.

Proof. From the definition of d_T, it follows that:

$$d_T(u, u) = 0 \qquad \forall u \in V \quad (8)$$
$$d_T(u, v) \geq 0 \qquad \forall u, v \in V \quad (9)$$
$$d_T(u, v) = d_T(v, u) \qquad \forall u, v \in V. \quad (10)$$

In order to show that (V, d_T) is an ultrametric, we only need to prove the ultrametric property:

$$d_T(u, v) \leq \max\{d_T(u, t), d_T(v, t)\} \quad \forall u, v, t \in V. \quad (11)$$

When at least two of the three nodes $u, v, t \in V$ are the same, this property follows from Eq. 8 and Eq. 9. When nodes $u, v, t \in V$ are distinct, from the definition of d_T it follows that Eq. 11 is equivalent to:

$$\mathcal{W}_T(u, v) \geq \min\{\mathcal{W}_T(u, t), \mathcal{W}_T(v, t)\}. \quad (12)$$

In the following, we prove both sides of the *if and only if* statement in the proposition. First, we prove the ($\Leftarrow$) side, i.e. that if assumption 7 holds, then (V, d_T) is an ultrametric and 12 holds.

Case 1: in Eq. 12, $t \in V$ is part of the sub-tree $T[u \vee v]$. In other words, the algorithm first merges node t with either node u or v, and then u and v are merged together. Let us assume that t is first merged with u (the following proof also holds for the opposite case in which t is first merged with v):

$$h_T(u, t) < h_T(u, v) = h_T(v, t). \quad (13)$$

Thus, by combining the last equation with assumption (7), it follows that

$$\mathcal{W}_T(u, t) \geq \mathcal{W}_T(v, t) \quad \text{and} \quad \mathcal{W}_T(u, v) = \mathcal{W}_T(v, t) \quad (14)$$

and Eq. 12 follows (becoming an equality in this case).

Case 2: in Eq. 12, $t \in V$ is *not* part of the sub-tree $T[u \vee v]$. Thus, the algorithm first merges nodes u and v, and then it merges node t together with the cluster containing u and v:

$$h_T(u, v) < h_T(u, t) = h_T(v, t). \quad (15)$$

Thus, from assumption 7 we have that

$$\mathcal{W}_T(u, v) \geq \mathcal{W}_T(u, t) \quad \text{and} \quad \mathcal{W}_T(u, v) \geq \mathcal{W}_T(v, t), \quad (16)$$

so also in this case Eq. 12 follows.

Next, we are left to prove the ($\Rightarrow$) side of the *if and only if* statement: if (V, d_T) is an ultrametric, then assumption 7 holds. To prove this statement, we first rephrase it in the following equivalent form: if assumption 7 does not hold, then (V, d_T) is not an ultrametric and 12 does not hold. If we negate assumption 7, there must be at least three $u, v, t \in V$ such that:

$$h_T(u, v) < h_T(u, t) \quad \text{and} \quad \mathcal{W}_T(u, v) < \mathcal{W}_T(u, t). \quad (17)$$

The first condition, in words, is again assuming that the algorithm first merges nodes u and v, and later it also merges node t with the cluster containing u and v. Thus, we can rephrase this assumption as:

$$\mathcal{W}_T(u, v) < \mathcal{W}_T(u, t) = \mathcal{W}_T(v, t). \quad (18)$$

From this, it follows that

$$\mathcal{W}_T(u, v) < \min\{\mathcal{W}_T(u, t), \mathcal{W}_T(v, t)\}, \quad (19)$$

which is exactly the negation of the ultrametric property 12. □

Proposition 3.3. *Among the algorithms included in the GASP framework (see Table 1), only Mutex Watershed and hierarchical clustering with Average (HC-Avg), Single (HC-Single) and Complete linkage (HC-Complete) define an ultrametric (V, d_{T^*}), where d_{T^*} is defined in Eq. 3 and T^* is the tree returned by the GASP Algorithm 1.*

Proof. Thanks to Prop. A6.1, we know that (V, d_{T^*}) is an ultrametric if and only if assumption 7 holds. Thus, in the following, we will prove which variations of the GASP Algorithm 1 satisfy assumption 7. In other words, we need to prove in which cases GASP merges clusters according to a monotonously decreasing order of signed interactions $\mathcal{W}$.

GASP puts clusters in a priority queue (Algorithm 1, lines 5 and 15) and merges them starting from those with the highest interaction (lines 9, 19, and 26). However, the priority queue is updated each time two clusters are merged (lines 10, 20, and 27). Thus, to ensure a monotonously decreasing merging order, updated interactions involving a

merged cluster should always be lower or equal than previously existing interactions (**condition 1**):

$$\forall S_i \in \Pi \setminus \{S_1, S_2\},$$
$$\mathcal{W}(S_1 \cup S_2 \cup S_i) \leq \max\{\mathcal{W}(S_1 \cup S_i), \mathcal{W}(S_2 \cup S_i)\} \tag{20}$$

where Π is a clustering, $\mathcal{W}$ is a linkage criteria, and $S_1, S_2 \in \Pi$ are two clusters merged by the algorithm at a given iteration. If this condition is true then, in the following iterations, GASP can only merge clusters with lower (or equal) interaction values.

We also note that, in phase 1, the algorithm skips interactions that are both positive and constraint (condition at line 8 in Algorithm 1) and merges them only later in phase 2 (line 19), when constraints are removed. Clearly, whenever this happens, a decreasing merging order is no longer ensured. Thus, on top of condition 1, we also have that no merging decisions should be "delayed" from phase 1 to phase 2 (**condition 2**).

Condition 1 always holds for Average, Single, Complete, and AbsMax linkage criteria, but not for a Sum linkage criteria, because the sum of two positive numbers a, b is always higher than $\max\{a, b\}$. This is also demonstrated in the toy example of Fig. 1a, proving that, in general, Sum-linkage algorithms like GAEC or HCC-Sum do not define an ultrametric on the graph.

Thanks to Lemma A6.3, we have that condition 2 always holds for algorithms based on AbsMax and Complete linkage, proving that the Mutex Watershed and HC-Complete algorithms define an ultra-metric (whether or not cannot-link-constraints are enforced). On the other hand, condition 2 does not hold for other variations of GASP involving cannot-link-constraints (HCC-Sum, HCC-Avg, and HCC-Single), which do not then define an ultrametric.

Finally, the remaining not constrained versions of GASP (HC-Avg, HC-Single, and HC-Complete) satisfy both conditions, so they define an ultrametric, confirming the well-known results of related work in hierarchical clustering on unsigned graphs [33, 58].

□

A6.4. Mutex Watershed on SSBM graphs

Proposition A6.2. *Consider a graph generated by an Erdős-Rényi signed stochastic block model (SSBM) as described in Section 4.1, with N nodes, edges added with probability p, sign-flip probability $\eta < 0.5$, k ground-truth clusters, and edge weights Gaussian-distributed with standard deviation σ. Then, at every iteration, GASP with Absolute Maximum linkage (or, in other words, the Mutex Watershed algorithm) always makes a mistake with at least probability η.*

Proof. Thanks to Lemma A6.3 we know that GASP with Absolute Maximum linkage returns the same clustering whether or not cannot-link-constraints are used. Thus, in the following, we prove the proposition considering the version enforcing constraints. Let us consider a generic iteration of the algorithm, where two clusters S_α and S_β have the highest priority and are popped from priority queue. Then, the MWS algorithm will either merge or constrain them depending on the fact that their interaction $\mathcal{W}_{\text{AbsMax}}(S_\alpha \cup S_\beta)$ is positive or negative (note that, with AbsMax linkage, an interaction can never be positive and constrained, as shown in Lemma A6.3). By construction of the SSBM, every edge $e \in E$ in the graph has a absolute weight distributed as $|w_e| \sim \mathcal{N}(1, \sigma^2)$. Thus, every edge $e' \in (S_\alpha \times S_\beta) \cap E$ connecting the two clusters has the same probability to have the highest absolute weight, and the sign of the interaction $\mathcal{W}_{\text{AbsMax}}(S_\alpha \cup S_\beta)$ will only depend on the sign of this highest edge. Therefore, the probability that the MWS merges two clusters is simply given by the fraction of positive weighted edges connecting them.

Let $\tilde{\Pi} = \{\tilde{S}_1, \dots, \tilde{S}_k\}$ denote the ground truth clustering, and $\tilde{S}_{\alpha i} = S_\alpha \cap \tilde{S}_i$ denote the intersection between cluster S_α and a ground-truth cluster $\tilde{S}_i$. If the generated graph is dense, i.e. $p = 1$, then the total number of edges connecting clusters S_α and S_β that have a true attractive or repulsive weight is (according to the ground truth labels)

$$\Gamma^+ = \sum_{i=1}^{k} |\tilde{S}_{\alpha i}||\tilde{S}_{\beta i}|, \quad \Gamma^- = \sum_{i=1}^{k} \sum_{j=1, j\neq i}^{k} |\tilde{S}_{\alpha i}||\tilde{S}_{\beta j}|. \tag{21}$$

When the edges in the graph are randomly added with a probability p, then the actual number of true attractive and repulsive interactions connecting the two clusters is (according to the ground truth labels):

$$\gamma^+ \sim \mathcal{B}(\Gamma^+, p), \qquad \gamma^- \sim \mathcal{B}(\Gamma^-, p), \tag{22}$$

where $\mathcal{B}(\Gamma, p)$ is the binomial distribution:

$$\mathcal{B}(\gamma; \Gamma, p) = \frac{\Gamma!}{\gamma!(\Gamma - \gamma)!} p^\gamma (1-p)^{\Gamma - \gamma}. \tag{23}$$

Here, we only assume that $\gamma^+ + \gamma^- > 0$, i.e. there is at least one edge connecting the two clusters (otherwise their interaction would be zero and the MWS would not have popped them from priority queue).

So far we have been talking about attractive and repulsive connections according to the ground truth labels. In our SSBM however every edge has a uniform probability η to have its sign flip, so the actual number of attractive interactions connecting the two clusters will be instead given by the sum of the true attractive interactions $\gamma^+_{\text{nf}} \sim \mathcal{B}(\gamma^+, 1-\eta)$ that have not been flipped, plus the true negative interactions $\gamma^-_{\text{f}} \sim \mathcal{B}(\gamma^-, \eta)$ that have been

flipped. Putting everything together, given two clusters with γ^+ true attractive interactions and γ^- true negative ones, the highest-absolute-weight edge connecting them has the following probability to be positive:

$$\begin{aligned}\mathbb{P}[\mathcal{W}_{\text{AbsMax}}(S_\alpha \cup S_\beta) > 0; \gamma^+, \gamma^-] = \\ = \sum_{\gamma^+_{\text{nf}}=0}^{\gamma^+} \sum_{\gamma^-_{\text{f}}=0}^{\gamma^-} \mathcal{B}(\gamma^-_{\text{f}}; \gamma^-, \eta)\mathcal{B}(\gamma^+_{\text{nf}}; \gamma^+, 1-\eta) \cdot \\ \cdot \left(\frac{\gamma^+_{\text{nf}} + \gamma^-_{\text{f}}}{\gamma^+ + \gamma^-}\right) \\ \overset{(*)}{=} \frac{\gamma^+(1-\eta) + \gamma^-\eta}{\gamma^+ + \gamma^-}\end{aligned} \tag{24}$$

where in $(*)$ we used the fact that the expected value of a binomial distribution $\mathcal{B}(\gamma, \eta)$ is $\gamma\eta$.

Now we note that this probability is bounded in the interval $[\eta, 1-\eta]$. So, regardless of whether the two clusters S_α and S_β should be merged or constraint according to ground truth labels, the probability not to make the correct decision is always at least η. Remarkably, while the exact probability in Eq. 24 depends on the number of edges connecting the two clusters $\gamma^+ + \gamma^-$ and thus on the cluster sizes, the bounds do not. Thus, this result shows that, unlike Sum or Avg linkage methods, the MWS algorithm is unable to reliably correct for the sign flip noise even for big clusters linked by many edges. □

A6.5. Application to neuron segmentation

Training and data augmentation 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 *S. Saalfeld, et al. Nature methods, 2012*. In addition to the standard data augmentation techniques of random rotations, random flips and elastic deformations, we simulate data artifacts. We randomly zero-out slices, decrease the contrast of slices, simulate tears, introduce alignment jitter and paste artifacts extracted from the training data. Both [28] and [50] 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 three samples with available ground truth labels.

CREMI-gridRag instances Our 3D UNet model predicts the same set of 12 long-and-short range affinities as described in [50]. When building the pixel-grid graph, we add both direct neighbors connections and the long-range connections predicted by our model (every voxel is connected to other six voxels via direct connections and other 18 voxels via long-range edges). Empirically, when long-range predictions of the CNN are added as long-range connections in the graph, GASP achieves better scores as compared to when only direct-neighbors predictions are used. Our intuitive explanation of this is that, where there is a clear boundary evidence between two segments, the long-range predictions of the CNN model are more certain than the direct-neighbor ones, because it is often impossible to estimate the exact ground-truth label transition for pixels that are very close to a boundary evidence. However, empirically, we also find that GASP achieves the best scores when only 10% of the long-range connections are randomly sampled and added to the grid-graph. When all the long-range connections predicted by the CNN are added to the graph (18 connections for every voxel), all versions of GASP tend to perform more over-clustering errors. In practice, we explain this by observing that many challenging parts of the studied neuron segmentation data involve thin and elongated segments, and our model sometimes fails to connect distant pairs of pixels that, according to the ground-truth labels, should belong to the same segment (even though, in this case, the direct neighboring predictions are correct). To sum up, the scores we report in Tables 3a) are obtained by using only 10% of the long-range predictions, since this was the setup that performed the best. 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.

CREMI-3D-rag instances We build these clustering problems by generating superpixels and then building a 3D region adjacency graph. Due to the anisotropy of the data, we generate 2D superpixels by considering each 2D image in the stack singularly. First, we generate a boundary-evidence map by taking an average over the two direct-neighbor predictions of the CNN model (one for each direction in the 2D image of the stack) and applying some additional smoothing. Then, we threshold the boundary map, compute a distance transform, and run a watershed algorithm seeded at the maxima of the distance transform (WSDT). The degree of smoothing was optimized such that each region receives as few seeds as possible, without however causing severe under-segmentation. The computed 2D superpixels are then used to build a 3D region-adjacency graph (3D-rag). The weights of the edges are given by averaging the CNN affinities over the boundaries of adjacent superpixels.

A6.6. Adding structured noise to CNN predictions

Additionally to the comparison on the full training dataset, we performed more experiments on a crop of the

Clustering problem	GAEC [39]	HCC-Sum	MWS [75]	HC-Avg	HCC-Avg	HC-Single	HCC-Single	HC-Complete
Modularity Clustering	-0.457	-0.453	-0.073	**-0.467**	**-0.467**	0.000	0.000	-0.201
Image Segmentation	**-2,955**	-2,953	-2,901	-2,903	-2,896	-1,384	-1,384	-2,102
Knott-3D (150-300-450)	**-36,667**	-36,652	-35,200	-35,957	-35,631	-2,522	-2,522	30,629
CREMI-3D-rag	**-1,112,287**	-1,112,286	-1,109,731	-1,112,177	-1,112,100	-1,038,709	-1,038,709	-748,734,869
Fruit-Fly Level 1-4	**-151,022**	-151,017	-150,879	-150,909	-150,876	-71,477	-71,997	-128,733
CREMI-gridGraph	-73,317,601	-73,328,867	-73,330,568	**-73,502,947**	-73,474,856	-45,194,180	-45,194,443	311,598,700
Fruit-Fly Level Global	**-151,688**	-151,596	-146,315	-150,466	-150,171	-4,422	-	6,876

Table A5. We compare algorithms in the GASP framework by evaluating which of the obtained clusterings is associated to the lowest value of the multicut objective defined in Eq. 1 (lower is better). Single and complete linkage methods performed much worse than the others. Note that HCC-Single is the algorithm with the highest runtime (see Table 3a)) and it did not scale up to the very large clustering problem *Fruit-Fly Level Global*.

more challenging CREMI training sample B, where we perturbed the predictions of the CNN with noise and we introduced additional artifacts like missing 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[14], 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 following way:

$$\tilde{F}(x; \mathcal{K}) = F(x) + \mathcal{K} \cdot \max\left(N(x), 0\right), \qquad (25)$$

where $N(x) = \text{Logit}[n(x)]$; $F(x) = \text{Logit}[p(x)]$ and $\mathcal{K} \in \mathbb{R}^+$ is a positive factor representing the amount of added noise. The resulting perturbed predictions $\tilde{F}(x; \mathcal{K})$ are then under-clustering biased, such that the probability for two pixels to be in the same cluster is increased only if $N(x) > 0$ (see Fig. A11b and A11c). Note that in these experiments we focused only on predictions perturbed with under-clustering biased noise (and not over-clustering biased noise). The reason is that generating realistic over-clustering biased CNN predictions is more complex and cannot be simply done by adding Perlin noise: as we show in Fig. A11c, by adding Perlin noise we can easily "remove" parts of a boundary evidence, but it is not possible to generate random new realistic boundary evidence.

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 volume of our CNN model is a four-dimensional tensor with n_{nb} channels: for each pixel / voxel, the model outputs n_{nb} values representing affinities of different edge connections. We then generated a 4-dimensional Perlin noise tensor 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.

[14] In our experiments, we used an open-source implementation of simplex noise [64], which is an improved version of Perlin noise [63]

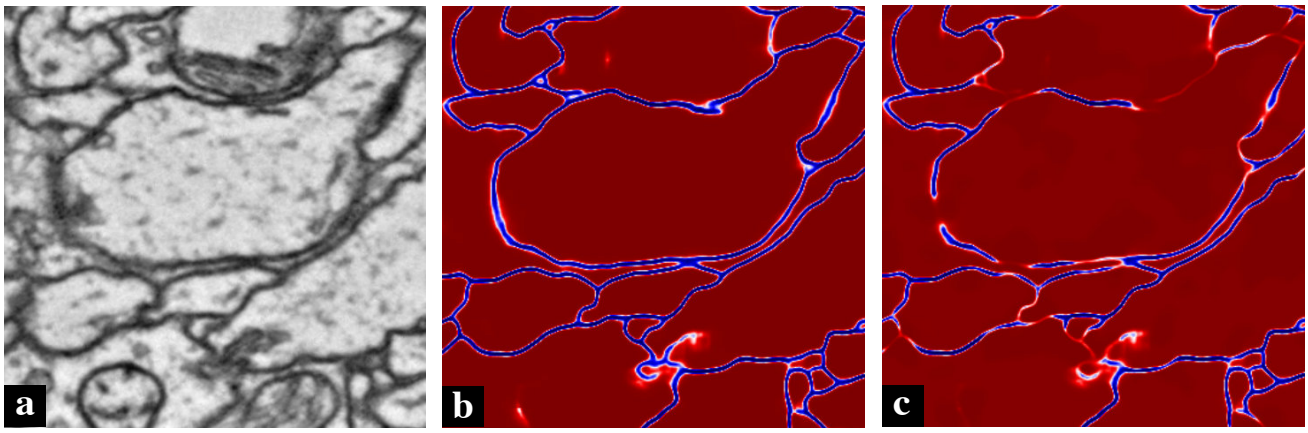

Figure A11. CNN predictions on a slice of the CREMI neuron segmentation challenge with and without additional spatially-correlated noise. (**a**) Raw data (**b**) Original CNN predictions $F(x)$, where blue pixels represent boundary evidence (**c**) Strongly perturbed version $\tilde{F}(x;\mathcal{K})$ of the predictions defined in Eq. 25 with $\mathcal{K} = 8$. Long-range predictions are not shown.

Method	ARAND Error
HC-Avg (GASP with Avg Linkage)	**0.1034**
GAEC [39] (GASP with Sum Linkage)	0.1035
MWS [75] (GASP with AbsMax linkage)	0.1068
SPONGE$_{sym}$ [19]	0.4161
L_{sym} [46]	0.8069
SPONGE [19]	0.9211
BNC [14]	0.9926

Table A6. GASP compared to spectral clustering methods on a small crop of the CREMI neuron segmentation dataset. Since spectral methods cannot scale to the full CREMI dataset, we evaluated them on a smaller $10 \times 100 \times 100$ sub-volume of CREMI training sample B. Despite the fact that the true number of ground truth clusters was given as an input to the spectral methods, GASP significantly outperformed them.