APPENDIX A
OPTIMAL RECOVERY

Before proving our main result, Proposition 1, we make an observation on a pair of valid partitions that merge into another valid partition.

Lemma 1. *Let $\{p_u, p_w\}$ be a consistent pair of valid partitions of nodes $u, w \in V$ whose merged partition $p_u \circ p_w$ is also a valid partition of the merged node $x \in V'$. Then each connected component of the p-graph $p_{u,w}$ is an arborescence.*

Proof. For convenience of discussion, we shall enrich the structure of the p-graph $p_{u,w}$ by additional nodes as follows. Let E_u and E_w be edges incident to u and w prior to merging. We call the set $E_u \ominus E_w$ *exterior edges*, where $\ominus$ is the disjunctive union. For each exterior edge in $p_{u,w}$, we attach an *exterior node* to the end that is not an instance of u or w. Accordingly, we call an instance of u or w in $p_{u,w}$ an *interior node*. We call an exterior node a *sink* if its incident exterior edge is directed towards that node, and a *source* otherwise. Figure 15 (b) shows an enriched p-graph of a consistent pair of valid partitions of nodes u, w in 15 (a).

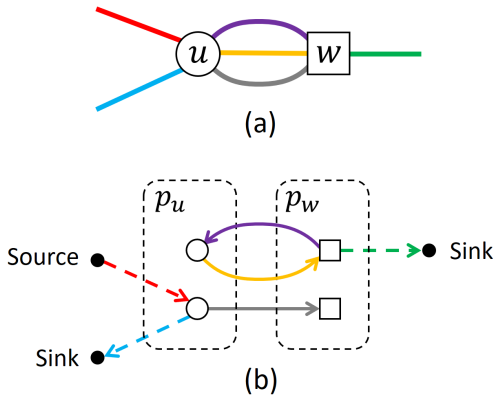

Fig. 15. (a): Two nodes u, w to be merged. (b): An enriched p-graph of two consistent and valid partitions p_u, p_w of u, w. Interior nodes of the p-graph are instances of u (◯) and w (□), and exterior nodes (•) are classified as either source or sink. Exterior edges $E_u \ominus E_w$ are dashed.

As both p_u and p_w are valid partitions, every (interior or exterior) node of $p_{u,w}$ has at most one incoming edge, and only an instance of the root $r \in V$ or a source exterior node has no incoming edges. Let V_M, E_M be the nodes and edges of a connected component M of $p_{u,w}$. Since no two edges in E_M are directed towards the same node, we can count the number of nodes without incoming edges, k_M, as

$$k_M = |V_M| - |E_M|.$$

On the other hand, the number of undirected cycles in M, denoted by g_M, is related to the number of nodes and edges by

$$g_M = |E_M| - |V_M| + 1.$$

Combining the two equations yields

$$k_M = 1 - g_M.$$

Both k_M and g_M are non-negative integers. Hence we only have two possibilities to consider.

We first consider $\{g_M = 0, k_M = 1\}$. In this case, the undirected graph of M is acyclic and there is a single node without incoming edges. This node, denoted by r_M, can be a source exterior node or an instance of r. Since every other node of M has exactly one incoming edge, M is an arborescence rooted at r_M. An example of M is the bottom component in Figure 15 (b), which is rooted at a source exterior node.

Next consider $\{g_M = 1, k_M = 0\}$. In this case, M has one undirected cycle and every node has one incoming edge. It follows that every exterior node of M (if there are any) is a sink. An example of M is the top component in Figure 15 (b). We will show that this case cannot arise if the merged partition $p_u \circ p_w$ is valid. As M has no source exterior nodes, its corresponding instance of x in $p_u \circ p_w$, denoted by x_M, has no incoming edges. The validity of $p_u \circ p_w$ implies that x is the root of the contracted graph, and hence $r \in \{u, w\}$. Since both p_u and p_w are valid, there is exactly one interior node of $p_u \circ p_w$ (an instance of r), denoted by i, that has no incoming edges. Note that i cannot lie in M, because every node of M has an incoming edge. Let M' be the component of $p_u \circ p_w$ containing i. Then M' falls into the case above, and it is an arborescence. Since the arborescence is rooted at an interior node i, M' has no source exterior node, and hence its corresponding instance of x in $p_u \circ p_w$, denoted by $x_{M'}$, has no incoming edges. Therefore there exist two instances of x in $p_u \circ p_w$, x_M and $x_{M'}$, that have no incoming edges, which contradicts the validity of $p_u \circ p_w$.

As only the first case $\{g_M = 0, k_M = 1\}$ is possible, every component of $p_u \circ p_w$ must be an arborescence. □

We now prove Proposition 1. The proposition states that the optimal p-set of a graph $G = \{V, E\}$ and partition costs h can be recovered by Equation 4 from the optimal p-set of the graph $G' = \{V', E'\}$ after merging $u, w \in V$ into $x \in V'$ and its costs h' defined by Equations 2 and 3.

Proof of Proposition 1. The proof consists of two parts.

Part I We first prove that P constructed in Equation 4 is a valid p-set of G. We start by showing that P is a consistent p-set. Since the pair of partitions $\{p_u, p_w\}$ is

a generator and P'_x is their merged partition, p_u and p_w must be consistent and they assign the same directions as P'_x to exterior edges $E_u \ominus E_w$. As P'_x is consistent with all other partitions $P_v = P'_v$ for $v \in V \setminus \{u, w\}$, so are p_u and p_w, and hence P is consistent.

It remains to show that the p-graph of P, G_P, is an arborescence. G_P can be constructed by replacing the partition P'_x of x in the p-graph $G'_{P'}$ by the p-graph $p_{u,w}$. Note that $G'_{P'}$ is an arborescence, since P' is a valid p-set of G', and each connected component of P'_x (a valid partition of x) is an arborescence. By Lemma 1, each component of P'_x corresponds to a component of $p_{u,w}$ that is also an arborescence. Hence replacing P'_x in $G'_{P'}$ by $p_{u,w}$ results in another arborescence, G_P.

Part II We next show that $h(P) \leq h(\overline{P})$ for any other valid p-set $\overline{P}$ of G. Consider the partitions $\overline{P}_u$ and $\overline{P}_w$ of u and w, and denote their merged partition $\overline{P}_u \circ \overline{P}_w$ as p_x. We can define a p-set of G' from $\overline{P}$ as

$$\overline{P}' = \overline{P} \setminus \{\overline{P}_u, \overline{P}_w\} \cup p_x.$$

We will show that $\overline{P}'$ is a valid p-set of G'. $\overline{P}'$ is consistent because $\overline{P}$ is consistent and p_x retains the directions of exterior edges $E_u \ominus E_w$ in $\overline{P}_u$ and $\overline{P}_w$. The p-graph $G'_{\overline{P}'}$ of $\overline{P}'$ can be obtained from the p-graph $G_{\overline{P}}$ of $\overline{P}$ by replacing the p-graph $\overline{P}_{u,w}$ of the partition pair $\{\overline{P}_u, \overline{P}_w\}$ with the merged partition p_x. As $G_{\overline{P}}$ is an arborescence, and since each component of p_x contains a single node that not only corresponds to a component of $\overline{P}_{u,w}$ but also maintains the component's exterior edge directions, $G'_{\overline{P}'}$ is an arborescence too.

By the definition of partition cost h'_x in Equation 3, and since the partition pair $\{\overline{P}_u, \overline{P}_w\}$ is consistent and valid, we know that $h'_x(p_x) \leq h_u(\overline{P}_u) + h_w(\overline{P}_w)$ and hence $h'(\overline{P}') \leq h(\overline{P})$. Due to optimality of P', and since $\overline{P}'$ is a valid p-set of G, $h'(\overline{P}') \geq h'(P')$. On the other hand, the pair $\{p_u, p_w\}$ being the generator of P'_x implies $h'_x(P'_x) = h_u(p_u) + h_w(p_w)$, and hence $h'(P') = h(P)$. Combining these results yields

$$h(P) = h'(P') \leq h'(\overline{P}') \leq h(\overline{P}),$$

which concludes the proof. □

Appendix B
Counting Partitions

We shall prove that:

Proposition 2. *The number of all possible valid partitions of a degree-d non-root node is* $\sum_{k=1}^{d} \binom{d}{k} k^{d-k}$.

Proof. The number of decompositions of d elements into k groups, with x_i number of variations for each group of i elements, is known as the *Bell polynomial* $B_{d,k}(x_1, \ldots, x_{d-k+1})$. For a non-root node, each instance in a valid partition has exactly one incoming edge, and hence there are i possible assignments of the incoming edge for an instance with i incident edges. Therefore the total number of valid partitions with k instances is $B_{d,k}(1, \ldots, d-k+1)$. This number is also known as the *idempotent number* and has the form $\binom{d}{k} k^{d-k}$. The total number of valid partitions, for any number of instances, is the sum of the idempotent number for all $k = 1, \ldots, d$. □

Note that the root node r has fewer valid partitions than a non-root node, because one of its instances is incident to only outgoing edges and thus has fewer variations than other instances (where any incident edge may be an incoming edge).

Appendix C
Computing Low-Width Carving Decompositions

We detail the first step of our algorithm for computing the m-tree (Section 5.3), which computes a carving-decomposition of a graph $G = \{V, E\}$ with low width using the heuristic of [15]. The heuristic was originally designed for computing a different type of decomposition known as the *branch-decomposition*, whose leaf nodes are edges E instead of nodes V. We adapt the heuristic to carving-decompositions.

The heuristic works by iteratively splitting nodes. We define a *partial carving-decomposition* of G as a tree where each leaf corresponds to a node in V and each interior node has a degree of *at least* 3. Starting from an initial partial carving-decomposition where a single interior node connects to all $|V|$ leaf nodes, the heuristic iteratively splits an interior node whose degree is greater than 3 into two interior nodes with lower degrees. As illustrated in Figure 16, splitting an interior node z of a partial carving-decomposition D involves replacing z by two nodes $\{x, y\}$ and connecting them by an edge. In addition, z's incident edge set Z in D is partitioned into two groups, X and Y, each connected to x or y.

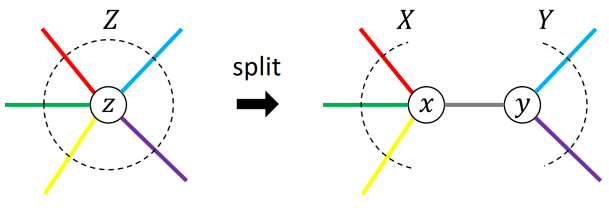

Fig. 16. Splitting an interior node z of a partial carving-decomposition into two interior nodes $\{x, y\}$, each connected with a subset (X or Y) of the incident edges Z of z.

To produce a carving-decomposition with low width, we wish to find a split of z so that the new edge (x, y) has low capacity. In addition, to avoid making short-sighted decisions, both $|X|$ and $|Y|$ cannot be too small. These two goals are achieved by performing a *balanced cut* of the edge set Z.

Specifically, define the *cut set* C_e of an edge e of a partial carving-decomposition D as the set of edges in E that connect two leaf nodes of V whose corresponding leaf nodes in D are connected by a path containing e. Note that the cardinality of C_e, $|C_e|$, is the capacity of e. Consider a weighted complete graph $\mathcal{Z}$ whose nodes correspond to edges in Z, and define the weight of an edge between two nodes $\{a, b\}$ to be $|C_a \cap C_b|$. The capacity of the edge (x, y) for edge groups X and Y can be interpreted as the total weights of edges of $\mathcal{Z}$ that connect between node groups X and Y. We then compute the eigenvector corresponding to the second smallest eigenvalue of the Laplacian matrix of $\mathcal{Z}$, sort the elements of the eigenvector, and compute the min-cut of $\mathcal{Z}$ that partitions Z into two components X, Y such that X (resp. Y) contains the first (resp. last) $\lceil \frac{|Z|}{3} \rceil$ elements of the sorted eigenvector.

APPENDIX D
PIVOTING

We shall prove that the space of all m-trees are connected by the pivoting move introduced in Section 5.3.2.

Proposition 3. *Given two m-trees T, T' of the same graph $G = \{V, E\}$, there exists a finite sequence of pivoting moves that start from T and end with T'.*

Proof. We call the leaf nodes in the subtree of T rooted at an interior node v the *leaves of* v, denoted as $T(v)$. We label a node v "clean" if there is a node v' in T' such that $T(v) = T'(v')$, and "dirty" otherwise. We call v' the *twin* of v. We can label clean and dirty nodes in T' symmetrically. It is easy to see that $T = T'$ if and only if all nodes of T (or T') are clean.

Our proof constructs a sequence of pivots from T that reduce its number of dirty nodes. Consider a pair of clean nodes $\{x, y\} \subseteq T$ such that their twins $\{x', y'\} \subseteq T'$ (also clean) are siblings whose parent w is dirty. Such pair always exists, because all leaf nodes of T' are clean by definition, and hence the dirty node in T' with the shallowest subtree must have two clean children. Let l_x and l_y be the paths from x and y to the root of T, and u be the first node shared by both paths. Denote all nodes on l_x (resp. l_y) between x (resp. y) and u as Q_x (resp. Q_y) (see Figure 17 (a)).

We now show that all nodes in $Q_x \cup Q_y$ are dirty. This is because the leaves of each such node include either $T(x)$ or $T(y)$ but not both. On the other hand, all nodes of $T' \setminus \{x', y'\}$ whose leaves include $T(x')$ or $T(y')$ must include both sets (since x', y' are siblings). Thus no node in $Q_x \cup Q_y$ has a twin in T'. Note that either Q_x or Q_y may be empty, but $Q_x \cup Q_y \neq \emptyset$ (otherwise x and y would be siblings, making u the twin of w and contradicting that w is dirty).

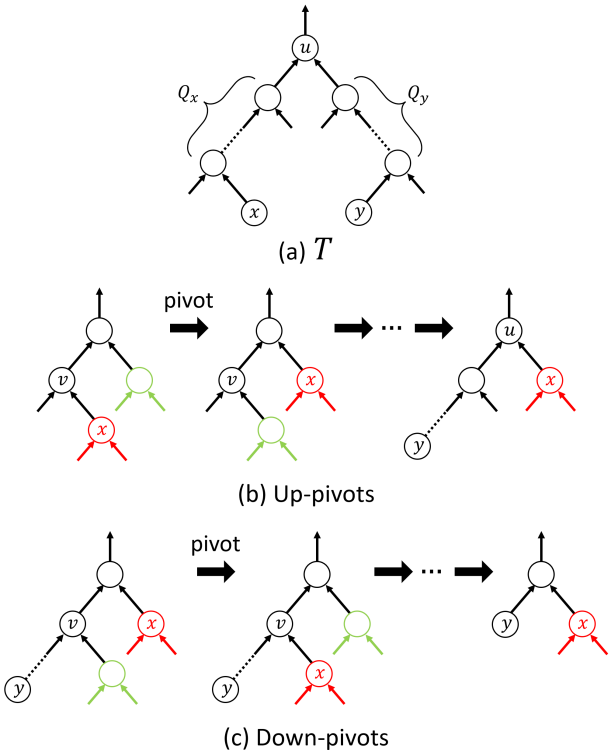

Fig. 17. Illustrations for the proof.

We will construct a sequence of pivots from T that consists of two sub-sequences:

- Up-pivots (Figure 17 (b)): iteratively pivoting x with its (unique) aunt, until x is a direct child of u.
- Down-pivots (Figure 17 (c)): iteratively pivoting x with one of its two nieces that is not an ancestor of y, until x is a sibling of y.

This sequence always ends with x and y being siblings. Their new parent is therefore the twin of w and hence is clean. Furthermore, the only node whose leaves (and hence clean/dirty status) changes is either the parent of x in an up-pivot (e.g., v in Figure 17 (b)) or an ancestor of y in a down-pivot (e.g., v in Figure 17 (c)). In either case, such a node is an ancestor of one of x and y, but not both, before the respective pivot, and hence it must be one of $Q_x \cup Q_y$. As a result, the leaves (and in turn the clean/dirty status) of the rest of the nodes, $T \setminus (Q_x \cup Q_y)$, remains unchanged after the sequence. As all of $Q_x \cup Q_y$ were originally dirty before the sequence, and at least one new clean node is

created (parent of x and y) as a result of the sequence, the sequence reduces the number of dirty nodes of T by at least one. Hence performing a finite number of such sequences will eventually remove all dirty nodes, producing T'.

□

APPENDIX E
PARTITION COSTS

We provide the definition of the partition cost function h used for the skeleton graphs in our experiments (Section 7.2). The definition on 2D skeleton graphs is identical to that in Estrada et al. [1], which we review first and then extend to 3D skeleton graphs.

Consider a 2D skeleton graph $G = \{V, E\}$, and a valid partition p of a node $v \in V$ with incident edges E_v. Suppose p consists of $m \geq 1$ instances of v, each associated with an edge group $g_i \subseteq E_v$ for $i = 1, \ldots, m$. Recall that, since p is valid, each group g_i must contain exactly one incoming edge and zero or more outgoing edges. We denote the set of out-going edges in group g_i as g_i^+ and the incoming edge as g_i^-. The partition cost of p is defined as

$$h_v(p) = -\sum_{i=1}^{m}(w_1 \log d_1(|g_i^+|) + w_2 \log d_2(g_i)), \quad (10)$$

where w_1, w_2 are balancing weights of two log-likelihood terms, d_1 and d_2. The first term, $d_1(c)$, evaluates the likelihood of a tree node having c outgoing edges. It is defined by a Poisson distribution with expected value (and variance) λ,

$$d_1(c) = \begin{cases} \frac{1}{1+\alpha}(\text{Pois}(c;\lambda) + \alpha) & \text{if } c = 1 \\ \frac{1}{1+\alpha}\text{Pois}(c;\lambda) & \text{otherwise} \end{cases} \quad (11)$$

where $\alpha >> \text{Pois}(1;\lambda)$ acts to encourage the continuation of a branch instead of splitting. The second term, $d_2(g)$ where g is an edge group of p, is the sum of the likelihood of the direction of each outgoing edge in g^+ with respect to the incoming edge g^-,

$$d_2(g) = \sum_{e_i \in g^+} \text{VMF}(e_i; (\mu_i(|g^+|, g^-) + F), \kappa). \quad (12)$$

Here VMF is the von Mises-Fisher distribution (restricted to a unit circle), κ is the concentration, F is a global force vector (e.g., gravity), and $\mu_i(c, v)$ is the expected direction of the i-th outgoing edge for a tree node with c outgoing edges and incoming edge direction v. In the implementation of [1] (as well as in our experiments), the parameters $(w_1, w_2, \lambda, \alpha, F, \kappa)$ were learned from a training set of 2D skeleton graphs, and the expected direction $\mu_i(c, v)$ is defined by sampling c vectors uniformly on a circle so that one of them is aligned with the incoming edge direction v.

Note that the Matlab code provided by the authors of [1] implemented a slightly different variant of the cost function than what is described in their paper. In particular, their code split the contribution of the force vector F in Equation 12 into a separate third additive term in the overall cost (Equation 10). This third term had no weight applied, and moreover, the weights w_1 and w_2 on the remaining terms were set by dividing by the number of split nodes (whose partitions have more than one instances). As the number of split nodes can vary with the solution, this implemented version of the cost can not be formulated as a sum of costs over nodes as defined in their paper (see Equations 7 and 8 in [1]), and also as required by our approach. To address this, we noted that the number of split nodes in an optimal solution is approximately equal to the number of graph cycles, and so, used this latter number (which is fixed and independent of the solution) instead to set the weights. We used this definition for experiments with our method, and also modified their code to use this definition when doing comparisons.

We also verified that this change to the cost has a negligible effect on the solutions returned by their method. Specifically, the average local and global similarities, to human labels, of the solutions from Estrada-100 were identical with and without the modification on the WIDE graphs, and very close for the RICE graphs—average local similarity was 0.9932 (modified) vs. 0.9935 (original), while average global similarity was 0.8369 (modified) vs. 0.8382 (original).

We apply the same cost definition (Equations 10, 11, 12) and parameters to the 3D skeleton graphs from our CORN dataset, after making two adjustments. First, we use the gravity direction of each root example as the force vector F. Second, to obtain the expected direction $\mu_i(c, g^-)$ for an edge group g, we compute the optimal alignment between an 1-to-c 3D branching template and g. To do so, we first align the incoming edge of the template with g^-. We then compute, for each permutation of the outgoing edges of the template, the rotation of those edges around g^- that aligns with the corresponding edges in g^+ with the least error (measured by the sum of dot products). This can be done using the technique of Singular Value Decomposition. Finally, we take the permutation that yields the least-error rotation.