TreeFormer: Single-view Plant Skeleton Estimation via Tree-constrained Graph Generation Supplementary Material

Xinpeng Liu[1] Hiroaki Santo[1] Yosuke Toda[2,3] Fumio Okura[1]
[1]Osaka University [2]Phytometrics [3]Nagoya University
{liu.xinpeng,santo.hiroaki,okura}@ist.osaka-u.ac.jp yosuke@phytometrics.jp

This supplementary material provides additional information, including details of our SFS layer (Sec. A), dataset details (Sec. B), implementation details of the baseline methods (Sec. C), performance analysis of our method (Sec. D), other design choices (Sec. E), and more visual results (Sec. F).

A. Details of SFS layer

A.1. Motivation

Our method infers a tree graph via MST as formulated in Eq. (1)–Eq. (4) in the main paper. Our task's goal is to optimize the graph generation network so that the **final output (*i.e.*, tree graph via MST) becomes similar to the ground-truth tree graph**. Since MST modifies the edge availability in *unconstrained* inferences, the unconstrained methods evaluating the unconstrained graph edges are indirect. Instead, our method **directly evaluates the quality of the final output tree** by mimicking MST. While experiments highlight our method's benefit, the following theoretical analysis also supports this intuition.

A.2. Derivation

This section details the derivation of Eq. (10) in the main paper. To make this material self-contained, we repeat several descriptions in the main paper.

As discussed in the main paper, we consider the edge probabilities $\hat{\mathbf{y}}_{(i,j)} = [\hat{y}^+_{(i,j)}, \hat{y}^-_{(i,j)}]^\top$ is usually computed through the softmax activation σ applied to the output feature vector of the final layer $\hat{\mathbf{f}}_{(i,j)} = [\hat{f}^+_{(i,j)}, \hat{f}^-_{(i,j)}]^\top$ as

$$\begin{aligned}\hat{\mathbf{y}}_{(i,j)} &= \sigma(\hat{\mathbf{f}}_{(i,j)}) \\ &= \left[\frac{\exp(\hat{f}^+_{(i,j)})}{\exp(\hat{f}^+_{(i,j)})+\exp(\hat{f}^-_{(i,j)})}, \frac{\exp(\hat{f}^-_{(i,j)})}{\exp(\hat{f}^+_{(i,j)})+\exp(\hat{f}^-_{(i,j)})}\right]^\top.\end{aligned} \tag{S1}$$

The set of unconstrained graph edges $\hat{E}$ are then obtained by comparing the edge existence probabilities as

$$\hat{E} = \{(i,j) \mid \hat{y}^+_{(i,j)} > \hat{y}^-_{(i,j)}\}, \tag{S2}$$

in which $\hat{E}$ records node pairs where the edge exists.

Suppose the projection function $\mathcal{P}$ converts the set of unconstrained edge probabilities $\{\hat{\mathbf{y}}_{(i,j)}\}$ to a set of constrained edges E. Let the difference of two sets be $E^+ = E - \hat{E}$ and $E^- = \hat{E} - E$, denoting the sets of edges newly added and removed by the projection. To mimic the discrete (and non-differentiable) inferences by $\mathcal{P}$ in the differentiable end-to-end learning, we modify the edge features corresponding to $E^+ \cup E^-$ in the differentiable forward process. Here, we want to get the edge probabilities that approximate the constrained edges E, which can be denoted as

$$\mathbf{y}_{(i,j)} = \begin{cases} [\ 1\ ,\ 0\ ]^\top & ((i,j) \in E^+) \\ [\ 0\ ,\ 1\ ]^\top & ((i,j) \in E^-) \\ \left[\hat{y}^+_{(i,j)}, \hat{y}^-_{(i,j)}\right]^\top & (\text{otherwise}). \end{cases} \tag{S3}$$

$$\sim \begin{cases} [1-\epsilon,\ \epsilon\ ]^\top & ((i,j) \in E^+) \\ [\ \epsilon\ , 1-\epsilon]^\top & ((i,j) \in E^-) \\ \left[\hat{y}^+_{(i,j)}, \hat{y}^-_{(i,j)}\right]^\top & (\text{otherwise}). \end{cases} \tag{S4}$$

When ϵ is small enough, the constrained output $\mathbf{y}_{(i,j)}$ perfectly mimics the output by the projection function $\mathcal{P}$. Our goal is to modify the feature vector $\hat{\mathbf{f}}_{(i,j)}$ so that it makes the probabilities as Eq. (S4) through the softmax activation.

In the SFS layer, we replace the features as

$$\begin{aligned} f^-_{(i,j)} &:= -\Lambda \qquad ((i,j) \in E^+) \\ f^+_{(i,j)} &:= -\Lambda \qquad ((i,j) \in E^-), \end{aligned} \tag{S5}$$

where Λ is assumed to be large enough. Given modified features $\mathbf{f}_{(i,j)} = [f^+_{(i,j)}, f^-_{(i,j)}]^\top$, the softmax activation σ

normalizes and converts them to edge probability $\mathbf{y}_{(i,j)}$ as

$$\mathbf{y}_{(i,j)} = \begin{cases} \sigma([\hat{f}^+_{(i,j)}, \ -\Lambda\ ]^\top) & ((i,j) \in E^+) \\ \sigma([\ -\Lambda\ , \hat{f}^-_{(i,j)}]^\top) & ((i,j) \in E^-) \\ \sigma([\hat{f}^+_{(i,j)}, \hat{f}^-_{(i,j)}]^\top) & (\text{otherwise}). \end{cases} \tag{S6}$$

Without loss of generality, we discuss the case in $(i,j) \in E^+$. Substituting Eq. (S6) into Eq. (S1) yields

$$\begin{aligned} \mathbf{y}_{(i,j)} &= \left[\frac{\exp(\hat{f}^+_{(i,j)})}{\exp(\hat{f}^+_{(i,j)})+\exp(-\Lambda)}, \frac{\exp(-\Lambda)}{\exp(\hat{f}^+_{(i,j)})+\exp(-\Lambda)}\right]^\top \\ &= \left[\frac{\exp(\hat{f}^+_{(i,j)})}{\exp(\hat{f}^+_{(i,j)})+\epsilon'}, \frac{\epsilon'}{\exp(\hat{f}^+_{(i,j)})+\epsilon'}\right]^\top \qquad ((i,j) \in E^+), \end{aligned}$$

where $\epsilon' = \exp(-\Lambda) \sim 0$ when Λ is large enough, leading to $\mathbf{y}_{(i,j)} = [1-\epsilon, \epsilon]^\top$ as in Eq. (S4) by denoting $\epsilon = \frac{\epsilon'}{\exp(\hat{f}^+_{(i,j)})+\epsilon'} \sim 0$.

A.3. Detailed analysis

We describe a detailed analysis of our reparameterization layer. As described in (S6), the unconstrained edge feature between i and j-th nodes $\hat{\mathbf{f}}_{(i,j)} = [\hat{f}^+_{(i,j)}, \hat{f}^-_{(i,j)}]^\top$ is converted to constrained prediction of the edge availability $\mathbf{y}_{(i,j)} = [y^+_{(i,j)}, y^-_{(i,j)}]^\top$ by selectively suppressing unwanted feature values.

When using the cross-entropy loss $\mathcal{L}_{\text{CE}}$ to evaluate the availability of the graph edges, the derivative to be back-propagated to the backbone graph generator is [1]

$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{\mathbf{f}}} = \begin{cases} [(1-\epsilon)-t^+, \quad 0 \quad\ ]^\top & ((i,j) \in E^+) \\ [\quad 0 \quad\ , (1-\epsilon)-t^-]^\top & ((i,j) \in E^-) \\ [\ y^+ - t^+ \ , \ y^- - t^- \ ]^\top & (\text{otherwise}), \end{cases} \tag{S7}$$

$$\sim \begin{cases} [\ 1-t^+\ , \quad 0 \quad\ ]^\top & ((i,j) \in E^+) \\ [\quad 0 \quad\ , \ 1-t^-\ ]^\top & ((i,j) \in E^-) \\ [y^+ - t^+, y^- - t^-]^\top & (\text{otherwise}), \end{cases} \tag{S8}$$

where $\mathbf{t} = [t^+, t^-]^\top$ denotes the ground truth edge existence and non-existence for the node pair (i,j). Our method modifies the computation graph of the network when the MST algorithm does not agree with the output of the graph generation model (*i.e.*, $(i,j) \in E^+ \cup E^-$), but in different ways for derivatives of each feature value $\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{f}^+}$ or $\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{f}^-}$.

Table S1 summarizes the case-by-case behavior, in which we can categorize the behaviors of the SFS layer into eight cases. Hereafter, we use $E^* \triangleq E^+ \cup E^-$.

Case $(i,j) \notin E^*$ *(Cases 1, 2, 5, 6)* When the graph algorithm (*i.e.*, MST) does not modify the edge availability (*i.e.*, cases 1, 2, 5, and 6 in the table), the behavior is the same as the usual cross-entropy loss for unconstrained edges.

[1] We omit the subscript (i,j) for simplicity.

Table S1. Case-by-case analyses of our reparameterization layer. For the columns of unconstrained features $\hat{\mathbf{f}}$, constrained prediction $\mathbf{y}$, and the ground truth edge availability $\mathbf{t}$, the table shows the index of the larger element. For example, the column $\hat{\mathbf{f}}$ will be $+$ when the edge feature for the positive edge availability is larger, *i.e.*, $\hat{f}^+ > \hat{f}^-$. The column (i,j) displays E^+ or E^- if the MST algorithm modifies the edge availability (in which the rows are also highlighted). For the remaining columns, $\uparrow$ and $\downarrow$ denote each value becoming (relatively) large or small, respectively.

Case	Feats & probs $\hat{\mathbf{f}}$	(i,j)	$\mathbf{y}$	GT $\mathbf{t}$	Loss $\mathcal{L}_{\text{CE}}$	Approx. derivatives $\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{f}^+}$	$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{f}^-}$	$\left\lvert\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{f}^+}\right\rvert$	$\left\lvert\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{f}^-}\right\rvert$	Descriptions
1	$+$		$+$	$[1,0]^\top$	$\downarrow$	$y^+ - 1$	y^-	$\downarrow$	$\downarrow$	Unmodified
2	$+$		$+$	$[0,1]^\top$	$\uparrow$	y^+	$y^- - 1$	$\uparrow$	$\uparrow$	Unmodified
3	$+$	E^-	$-$	$[1,0]^\top$	$\uparrow$	0	1	$\downarrow$	$\uparrow$	MST *incorrectly* modified
4	$+$	E^-	$-$	$[0,1]^\top$	$\downarrow$	0	0	$\downarrow$	$\downarrow$	MST *correctly* modified
5	$-$		$-$	$[1,0]^\top$	$\uparrow$	$y^+ - 1$	y^-	$\uparrow$	$\uparrow$	Unmodified
6	$-$		$-$	$[0,1]^\top$	$\downarrow$	y^+	$y^- - 1$	$\downarrow$	$\downarrow$	Unmodified
7	$-$	E^+	$+$	$[1,0]^\top$	$\downarrow$	0	0	$\downarrow$	$\downarrow$	MST *correctly* modified
8	$-$	E^+	$+$	$[0,1]^\top$	$\uparrow$	1	0	$\uparrow$	$\downarrow$	MST *incorrectly* modified

Case $(i,j) \in E^*$ & $\mathbf{y} \sim \mathbf{t}$ *(Cases 4, 7)* In these cases, the MST algorithm *correctly* suppresses the unwanted features, where the constrained prediction $\mathbf{y}$ becomes the approximation of the ground-truth edge availability $\mathbf{t}$. The loss value becomes small, and the derivative is $\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{\mathbf{f}}} \sim \mathbf{0}$. This is natural since our constrained graph generator produces correct predictions.

Case $(i,j) \in E^*$ & $\mathbf{y} \nsim \mathbf{t}$ *(Cases 3, 8)* In these cases, MST *incorrectly* modifies the edge availability, *i.e.*, the node pair (i,j) belongs to E^+ or E^-, but the constrained prediction $\mathbf{y}$ does not fit the ground truth $\mathbf{t}$. Here, we mathematically discuss the behavior in these cases. Without loss of generality, we focus on Case 3, where MST *incorrectly* removes an edge and compares the methods with and without tree-graph constraints using the SFS layer.

Case 3 (MST *incorrectly* removes an edge) The following discussions can be straightforwardly extended to Case 8, where MST *incorrectly* adds an edge.

Conditions:

- Unconstrained prediction (edge exists): $\hat{f}^+ > \hat{f}^-$,
- MST *removes* the edge: $(i,j) \in E^-$,
- GT edge availability (edge exists): $[t^+, t^-] = [1,0]$.

Unconstrained method (without SFS layer) The gradient at $\hat{f}^+$ by the unconstrained method is

$$\frac{\partial \mathcal{L}_{\text{unconst}}}{\partial \hat{f}^+} = \frac{\exp(\hat{f}^+)}{\exp(\hat{f}^+) + \exp(\hat{f}^-)} - 1. \tag{S9}$$

Since $\hat{f}^+ > \hat{f}^-$, it takes the value in the range of $(-0.5, 0)$. Similarly,

$$\frac{\partial \mathcal{L}_{\text{unconst}}}{\partial \hat{f}^-} = \frac{\exp(f^-)}{\exp(f^-) + \exp(f^+)} - 0, \tag{S10}$$

thus the gradient at $\hat{f}^-$ is inside $(0, 0.5)$. Thus, the gradient vector $\frac{\partial \mathcal{L}_{\text{unconst}}}{\partial \hat{\mathbf{f}}}$ is always shorter than $[-0.5, 0.5]^\top$ (corresponding to the special case $\hat{f}^+ = \hat{f}^-$).

Constrained method (Ours) As described in Eqs. (S7) and (S8), our method yields the gradient as

$$\frac{\partial \mathcal{L}_{\text{const}}}{\partial \hat{f}^+} = 0, \qquad \frac{\partial \mathcal{L}_{\text{const}}}{\partial \hat{f}^-} = 1 - \epsilon \sim 1,$$

i.e., $\frac{\partial \mathcal{L}_{\text{const}}}{\partial \hat{\mathbf{f}}} \sim [0, 1]^\top$.

Comparisons While both methods control the features to increase the edge availability, the relation of gradient vectors $\| \frac{\partial \mathcal{L}_{\text{const}}}{\partial \hat{\mathbf{f}}} \| > \| \frac{\partial \mathcal{L}_{\text{unconst}}}{\partial \hat{\mathbf{f}}} \|$ always holds, which means our method strongly penalizes the incorrect estimates by MST by directly comparing the final estimation (*i.e.*, tree graph) with the ground-truth edge availability, which highlights our key motivation—a **direct** control of the tree-constrained graph generation.

B. Dataset Details

We describe the details of the datasets used in our experiment.

Synthetic tree pattern dataset To prepare the synthetic dataset, we implement a generator of two-dimensional tree patterns based on the L-system [9], a formal language for describing the growth of the structural form. The L-system recursively applies rewriting rules to the current structure to simulate the growth of branching structures.

Figure S1 shows the initial structures and the rewriting rules we used. At the beginning of the tree generation, an initial sequence is randomly chosen from the pre-defined sequences marked with a purple frame in the figure. At each iteration during the tree generation, the leaf edges ("**A**" in the sequences) are replaced by a randomly chosen pattern from eight pre-defined ones. A simple example is shown in Fig. S2. We iterate the rewriting process a maximum of three times to generate a tree pattern. We also add randomness to the branch length and joint angles in our dataset. We randomly choose a branch length of scaling $[0.5, 2.5]$ and joint angles of $[10°, 35°]$.

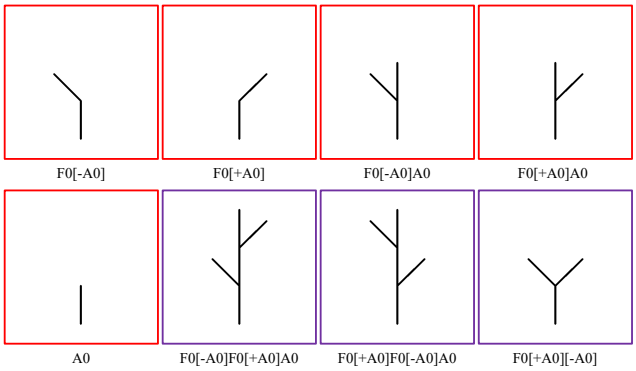
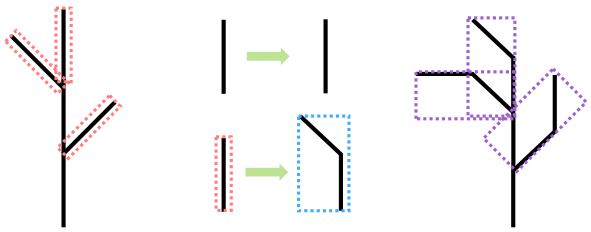


Figure S1. Atomic structures used for synthetic dataset generation. Three pre-defined initial structures are highlighted in purple. Eight pre-defined rewriting rules are used during the generation.



(a) Initial structure (b) Rewriting rule (c) After rewriting process

Figure S2. An example of the rewriting process. Suppose the initial structure is represented as **F0[+A0]F0[-A0]A0**. If a rewrite rule **F→F; A→F[-A]** is applied, *i.e.*, **F** remains unchanged and **A** becomes **F[-A]**, the result of the rewrite process is **F0[+F1[-A1]]F0[-F1[-A1]]F1[-A1]**. The digits in the sequences indicate the number of times the rewrite is applied.

Root dataset For the root dataset, the structure of the early-growing roots of Arabidopsis is manually annotated. The structures are annotated by placing points (*i.e.*, graph nodes) on the root path, where the distance between neighboring points may vary depending on the annotator and the images. We, therefore, resample the graph nodes with the same intervals. Starting from keypoints with the degree $\neq 2$ (*i.e.*, joints and leaf nodes), we sample nodes at intervals of 8 pixels along continuous branch segments.

For data augmentation, we apply flipping, rotation, cropping, noise, lighting, and scaling on the original images. Supposing the roots are almost aligned at seeding, we limit the range of rotation angles in $[-9°, +9°]$.

Grapevine dataset We use 3D2cut Single Guyot Dataset [4] containing manual annotations on branch structures. We perform data augmentation with rotation angles in $[-15°, +15°]$ in the same manner as [4]. This dataset also contains the classification of nodes (four classes) and edges (five classes) related to biological meanings. Since the existing two-stage method [4] estimates these

Table S2. Quantitative comparisons between our re-implementation of [4] and our two-stage baseline implementation.

Method	SMD ↓	TOPO score ↑			MSE ↓	
		Prec.	Rec.	F1	Node confidence	Edge direction
Re-implementation of [4]	3.84×10^{-3}	0.459	0.365	0.406	6.95×10^{-3}	1.01×10^{-2}
Our implementation of two-stage method	$\mathbf{4.24 \times 10^{-4}}$	**0.677**	**0.589**	**0.630**	$\mathbf{1.19 \times 10^{-3}}$	$\mathbf{2.67 \times 10^{-3}}$

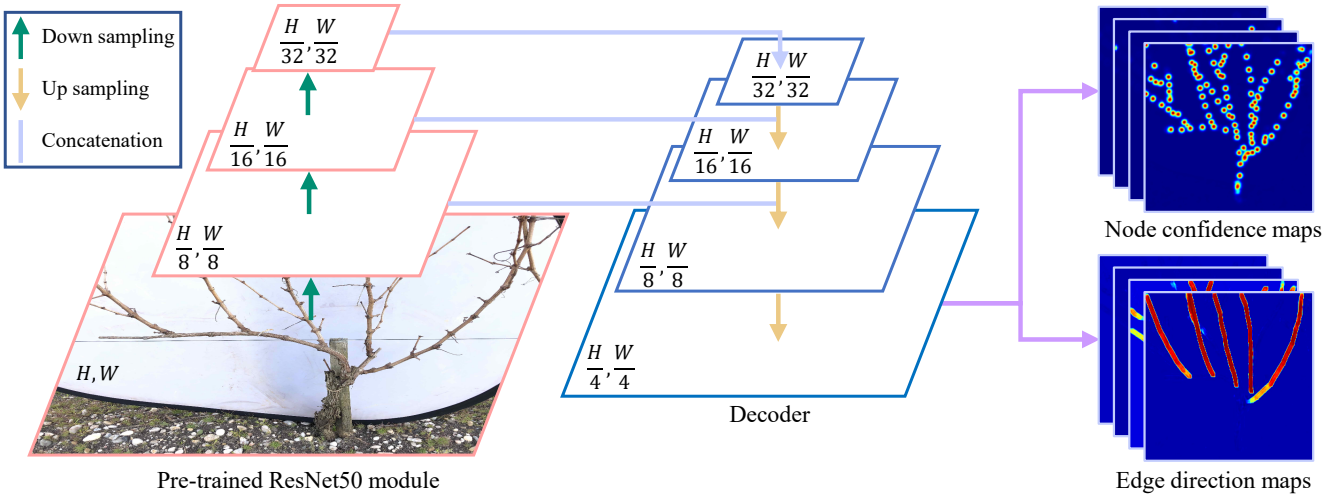


Figure S3. Network architecture for the first (skeletonization) stage of our two-stage baseline method.

categories, we follow the same setup for the two-stage baseline method (refer to the next section for detailed discussions). For the other methods, including our TreeFormer implementation, we use only the binary class information (*i.e.*, branch availability) for generalizability.

C. Details of Baseline Methods

We describe the implementation details for the baseline methods: The *two-stage* method and the method with the *test-time constraint*. Note the implementation for the other baseline, the *unconstrained* method, is identical to the original RelationFormer [11].

C.1. Two-stage baseline

Our experiment implements a two-stage baseline involving skeletonization and graph optimization. This baseline implementation is based on ViNet [4], a state-of-the-art plant skeleton estimation method. Since the implementation of [4] is not publicly available, we re-implement the method with reference to the descriptions in the paper. Through the re-implementation, we find room for improvement in the two-stage baseline method. Table S2 compares the performance of our two-stage implementation with a naive re-implementation of [4]. The SMD and TOPO scores are the same metrics used in the main paper, and we also compare the mean squared error (MSE) of the first-stage output of both methods. Our implementation achieves a better performance; thus, we use the improved version for our experiment. In the following, we describe the implementation details.

First stage: Skeletonization Similar to ViNet [4], the first stage of our implementation outputs the prediction of node and edge positions as single-channel confidence maps and two-channel vector fields (hereafter referred to as node confidence maps and edge direction maps, respectively). This step is similar to a widespread human pose estimation method *i.e.*, OpenPose [3], which jointly estimates the confidence of person keypoints and the Part Affinity Fields (*i.e.*, two-channel vector fields).

While ViNet [4] uses a sequence of residual blocks followed by the Stacked Hourglass Network [10] for this stage, we use a pre-trained ResNet50 [5] for image feature extraction. This is for a fair comparison to our TreeFormer implementation, which also uses ResNet50 as the backbone[2]. We implement an architecture like the Feature Pyramid Network (FPN) [8], illustrated in Fig. S3, to decode the node & edge maps from the image features. Figure S4 visually compares the estimated node & edge maps, showing a better accuracy by our two-stage implementation.

The original ViNet estimates multiple classes of nodes (four classes) and edges (five classes) as different maps for the grapevine dataset. Compared to just using binary

[2]ResNet50 is actually used as the node detection module in RelationFormer (that is based on Deformable DETR [12]), which is the basis of our TreeFormer, and we inherited its implementation.

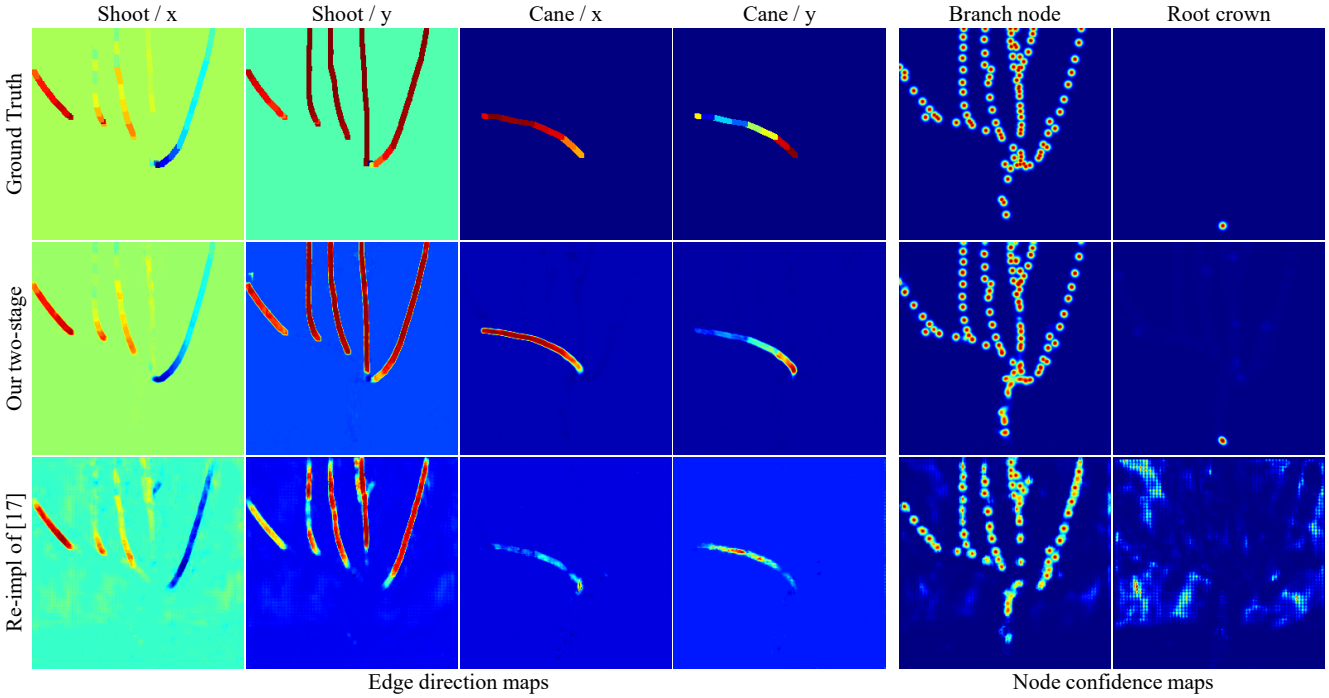


Figure S4. Visual comparisons between a re-implementation of [4] and our two-stage baseline implementation. Our implementation yields better node confidence and edge direction maps, which are the outputs of the first stage of these methods.

Table S3. Parameters used for the two-stage baseline method. d denotes the distance threshold for the local maximum value search (*i.e.*, non-maximum suppression) of node candidates. τ_m and τ_n are used as the thresholds for node detection from the confidence maps. For the detailed definitions, refer to the original paper [4].

Dataset	Image size (W, H) [px]	Map size (W, H) [px]	Node confidence diameter [px]	Edge direction width [px]	Node search distance d [px]	Thresholds for node detection τ_m	τ_n
Synthetic	512, 512	512, 512	4	5	9	0.97	0.5
Root	570, 190	570, 190	3	5	7	0.99	0.3
Grapevine	1008, 756	256, 256	3	10	25	0.97	0.5

classes (*i.e.*, a branch exists or not), our two-stage implementation also yields better estimation accuracies using the multiple classes (SMD in 4.2×10^{-4} with multi-class and 1.4×10^{-2} with binary classes). Therefore, we use the multi-class setup for our two-stage implementation of the grapevine dataset. For the other dataset, we use binary classification since we do not have specific class information.

Second stage: Graph algorithm Given the node confidence and edge direction maps, ViNet [4] first extracts the node positions, followed by the computation of the *resistivity* between each node pair, defined using the edge directions and the Euclidean distance between nodes. The final estimates of the graph structure are generated using the Dijkstra algorithm, where the tree structure is obtained by computing the shortest paths from all nodes to the detected root crown. The resistivity is used as the edge cost for the Dijkstra algorithm.

For the second stage, we follow the method in [4] except for the graph algorithm used; namely, we compute MST instead of the shortest paths given by the Dijkstra algorithm, since using MST reduces the SMD metric to 4.2×10^{-4}, compared to 5.9×10^{-4} using the Dijkstra algorithm for the grapevine dataset.

Detailed parameter settings The two-stage method involves heuristic parameters for node and edge detection. Therefore, we empirically select the best parameter sets for each dataset. Table S3 lists the detailed parameters. In particular, for the root dataset, we need to carefully tune some hyperparameters (namely, d, τ_m, and τ_n in the table) to yield reasonable estimates, where the configurations yielding the best SMD scores are reported in the main paper.

C.2. Test-time constraint baseline

For the test-time constraint baseline, we apply MST only in the inference phase, where the graph generator is trained using the same procedure as the unconstrained method. The MST used in this baseline method is identical to our proposed one.

D. Performance Analysis

In this section, we present a detailed analysis of the performance of our proposed method. The effectiveness of our method is evaluated through comprehensive experiments in different scenarios. Specifically, we compare our method with the Auto-regressive (AR) model, and we also analyze the performance when our method is applied solely during the training processes.

D.1. Comparison with auto-regressive (AR) method

While we implement the tree-graph constraint on the state-of-the-art non-autoregressive graph generator, RelationFormer [11], other choices of constrained graph generation are viable. Existing works aiming for tree-constrained graph generation, such as in molecule structure estimation [1,6,7], use auto-regressive (AR) graph generation. AR methods are a simpler choice for imposing the constraint since it is relatively straightforward to implement the tree-graph constraint in their graph development process. However, since the AR methods generate graph nodes and edges progressively, they are prone to breakdowns due to changes in the output order or errors during the generation. This tendency is particularly pronounced for relatively large graphs, including our setup.

To assess the potential of AR methods, we test the state-of-the-art transformer-based AR graph generator, Generative Graph Transformer (GGT) [2]. Table S4 compares our method and several variances of GGT on the synthetic tree pattern dataset. The results show that the accuracy of GGT falls short compared to our method, although the vanilla GGT (the top row) mostly outputs tree graphs (92 %) without explicitly imposing the tree-graph constraint. We identified that errors by GGT occurring at a particular step in the AR generation process continuously cause errors in the sequence of following generations. The GGT was initially designed for small datasets, specifically for graphs with $|V| \leq 10$. For our setup, where $|V| \geq 100$, generating these long sequences in a specific order presents a significant challenge.

D.2. Effectiveness of tree-constraint during training

To assess whether our SFS layer (positively) affects the training process itself or not, we evaluate our method *without* using the SFS layer and the MST algorithm during the inference phase, *i.e.*, introducing constraint only during the training phase (called *train-time constraint* hereafter). Table S5 summarizes the performances. Inducting the tree constraint during the training phase mostly outperforms the methods without constraints, meaning that the improvement by our method is based on network improvement by the loss propagated via the SFS layer. We also checked the change in the accuracy metric during training, and found our method consistently achieved better accuracy from the beginning of the training.

Table S4. Comparisons of different graph generation models (*i.e.*, RelationFormer [11] and GGT [2]) on the synthetic dataset.

Method	SMD ↓	TOPO score ↑			Tree rate
		Prec.	Rec.	F1	[%]
GGT [2]	2.71×10^{-3}	0.635	0.537	0.582	92.06
GGT w/ test-time constraint	4.13×10^{-3}	0.620	0.545	0.580	98.10
GGT w/ SFS layer	2.80×10^{-3}	0.652	0.584	0.616	99.63
RelationFormer [11] w/ SFS layer (Ours)	4.78×10^{-6}	0.986	0.968	0.977	100.0

Table S5. Quantitative results with additional baseline, *train-time constraint*.

Dataset	Method	SFS	SMD ↓	TOPO score ↑			Tree rate
				Prec.	Rec.	F1	[%]
Synthetic	Unconstrained		1.43×10^{-5}	0.978	0.929	0.953	36.2
	Test-time constraint		6.26×10^{-6}	0.977	0.953	0.965	**100.0**
	Train-time constraint	✓	8.44×10^{-6}	**0.987**	0.954	0.970	56.5
	Ours	✓	$\mathbf{4.78 \times 10^{-6}}$	0.986	**0.968**	**0.977**	**100.0**
Root	Unconstrained		1.19×10^{-4}	0.831	0.633	0.719	35.9
	Test-time constraint		1.52×10^{-4}	0.829	0.771	0.799	**100.0**
	Train-time constraint	✓	$\mathbf{7.81 \times 10^{-5}}$	0.853	0.619	0.718	37.2
	Ours	✓	8.82×10^{-5}	**0.861**	**0.807**	**0.833**	**100.0**
Grapevine	Unconstrained		1.45×10^{-4}	0.963	0.559	0.708	0.0
	Test-time constraint		1.47×10^{-4}	0.896	0.840	0.867	**100.0**
	Train-time constraint	✓	1.30×10^{-4}	**0.965**	0.566	0.713	0.0
	Ours	✓	$\mathbf{1.03 \times 10^{-4}}$	0.899	**0.843**	**0.870**	**100.0**

E. Other Design Choices

The experiments in the main paper already provide some ablation studies, namely, comparisons of our method with 1) graph generation without constraint (*unconstrained*), and 2) a method without using MST in the training loop (*test-time constraint*). Here, we delve further into the potential design choices of our TreeFormer model.

E.1. Other graph generators

Although the proposed module, the SFS layer, can be easily integrated into graph generators other than RelationFormer [11], we found that no methods but our TreeFormer implementation achieve satisfactory results. Here, we discuss results by the implementation of our method to the AR graph generator, GGT [2], which achieves the second-best accuracy for multiple datasets following the state-of-the-art RelationFormer.

Table S4 in the last section compares the GGT with and without the tree-graph constraint. Compared to the GGT with test-time MST, using our SFS layer on top of GGT improves both SMD and TOPO scores[3], although the ac-

[3]GGT w/ SFS layer does not achieve 100 [%] tree rate because it some-

Table S6. Ablation for Λ.

Λ	SMD ↓	TOPO score ↑		
		Prec.	Rec.	F1
2 ($\exp(-\Lambda) = 1.4 \times 10^{-1}$)	1.51×10^{-4}	0.871	0.803	0.836
5 ($\exp(-\Lambda) = 6.7 \times 10^{-3}$)	1.27×10^{-4}	0.866	0.799	0.831
10 ($\exp(-\Lambda) = 4.5 \times 10^{-5}$)	$\mathbf{1.03 \times 10^{-4}}$	**0.899**	**0.843**	**0.870**
100 ($\exp(-\Lambda) = 3.7 \times 10^{-44}$)	1.07×10^{-4}	0.886	0.830	0.857

curacies are insufficient in practice due to the drawback of AR-based generation processes discussed above. Using the newer RelationFormer model significantly improves the estimation accuracy, which implies that our SFS layer will benefit from the future development of graph generation models.

E.2. Using node distances for edge cost in MST

Although our proposed method uses the edge non-existence probabilities $\{\hat{y}^{-}_{(i,j)}\}$ as the edge cost for the MST algorithm, inspired by the two-stage method that uses node distance for the edge cost computation, we multiply the Euclidean distance between nodes by our original edge cost.

As a result, SMD with modified edge cost does not improve accuracy (it achieves the same SMD as our method in the Grapevine dataset). A possible reason for this is that the graph generator itself can take the node distance into account when estimating graph edges. Therefore, we simply use the edge non-existence probabilities $\{\hat{y}^{-}_{(i,j)}\}$ as the edge cost for our method.

E.3. Ablation for Λ

An important hyperparameter in our method is Λ, which controls the level of suppression for unwanted features. Here, we report an ablation study for this parameter using the grapevine dataset. Table S6 shows that our choice ($\Lambda = 10$) achieves better, while the changes in Λ do not significantly affect the overall accuracy as long as $\exp(-\Lambda)$ is close enough to zero. This result indicates that our method is robust to the hyperparameter setting.

F. Additional Visual Results

We finally show additional visual results. Figures S5 and S6 show the additional results for synthetic and root datasets, respectively. Figures S7 and S8 show the results for the grapevine dataset. Figure S9 show the results for out-of-domain testing.

These results consistently demonstrate the high-fidelity estimation of plant skeletons by our TreeFormer, which uses the SFS layer that incorporates the constraints while training graph generation models.

times fails to generate any graphs.

References

[1] Sungsoo Ahn, Binghong Chen, Tianzhe Wang, and Le Song. Spanning tree-based graph generation for molecules. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2022. 6

[2] Davide Belli and Thomas Kipf. Image-conditioned graph generation for road network extraction. In *Proceedings of NeurIPS Workshop on Graph Representation Learning*, 2019. 6

[3] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2D pose estimation using part affinity fields. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 4

[4] Theophile Gentilhomme, Michael Villamizar, Jerome Corre, and Jean-Marc Odobez. Towards smart pruning: ViNet, a deep-learning approach for grapevine structure estimation. *Computers and Electronics in Agriculture*, 207:107736, 2023. 3, 4, 5

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 4

[6] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 2323–2332, 2018. 6

[7] Wengong Jin, Regina Barzilay, and T. Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *Proceedings of International Conference on Machine Learning (ICML)*, 2020. 6

[8] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017. 4

[9] Aristid Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968. 3

[10] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 11–16, 2016. 4

[11] Suprosanna Shit, Rajat Koner, Bastian Wittmann, Johannes Paetzold, Ivan Ezhov, Hongwei Li, Jiazhen Pan, Sahand Sharifzadeh, Georgios Kaissis, Volker Tresp, et al. Relationformer: A unified framework for image-to-graph generation. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 422–439, 2022. 4, 6

[12] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable transformers for end-to-end object detection. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021. 4

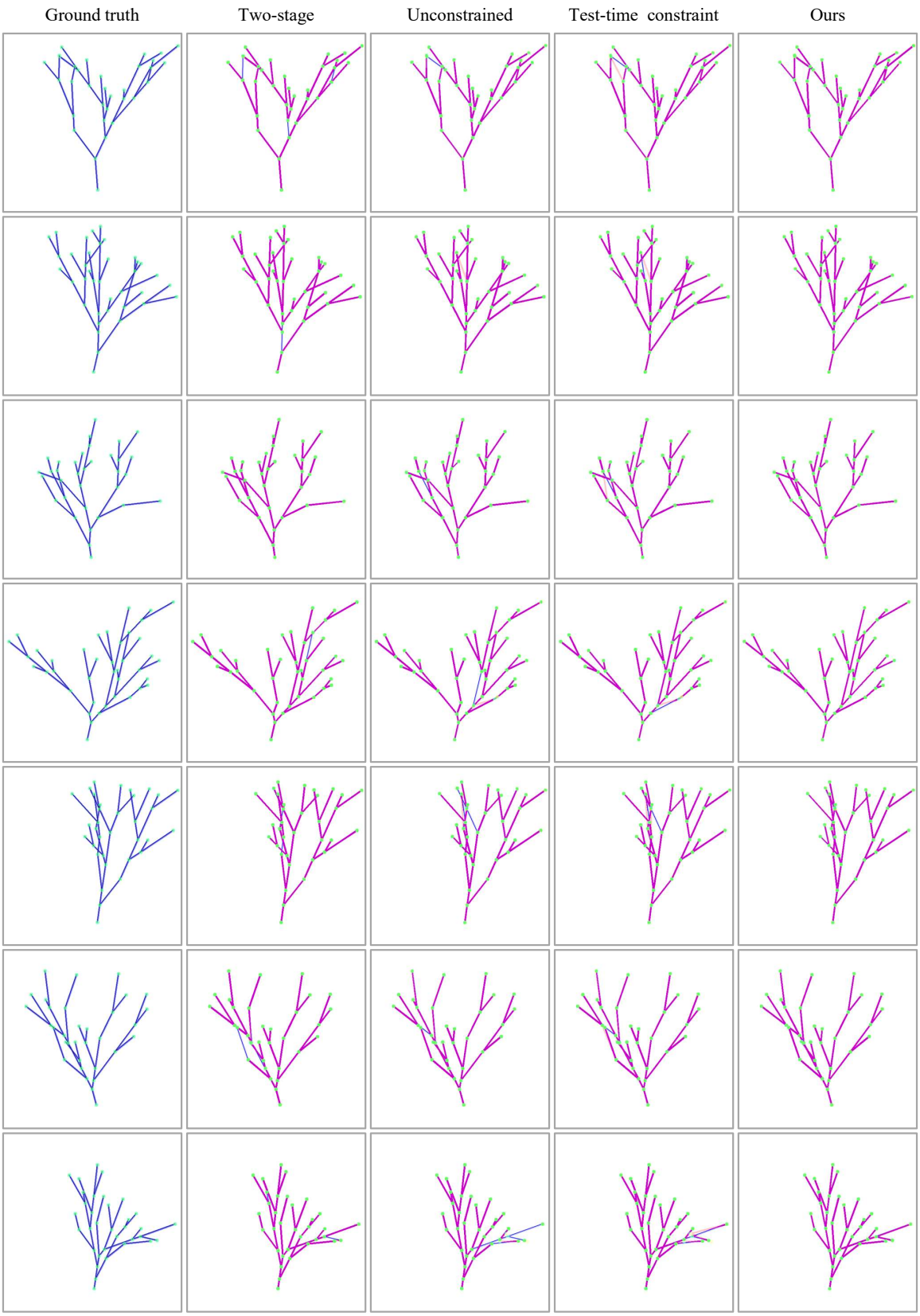


Figure S5. Additional results for the synthetic branch pattern dataset.

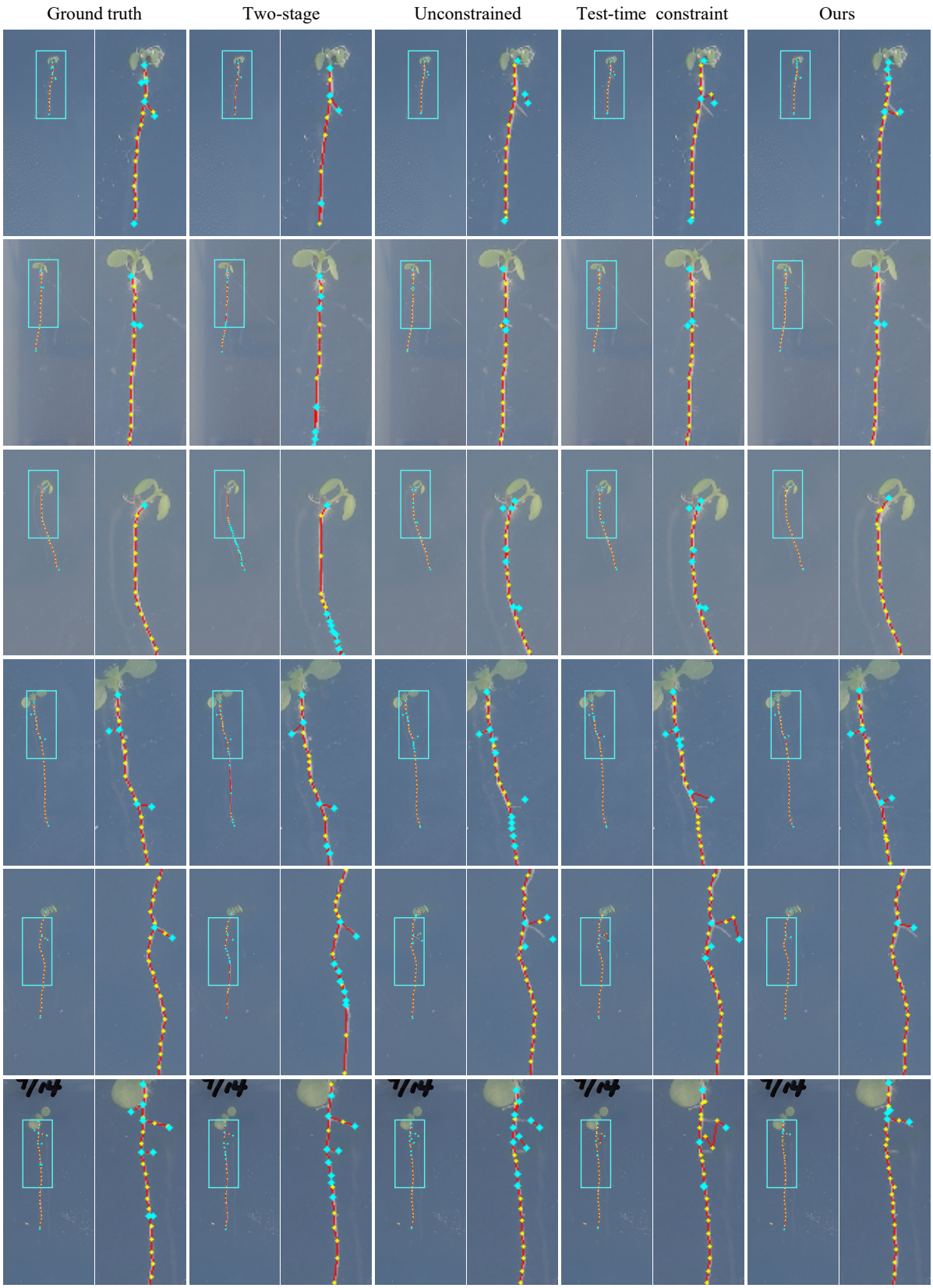


Figure S6. Additional results for the root dataset.

Figure S7. Additional results for the grapevine dataset.

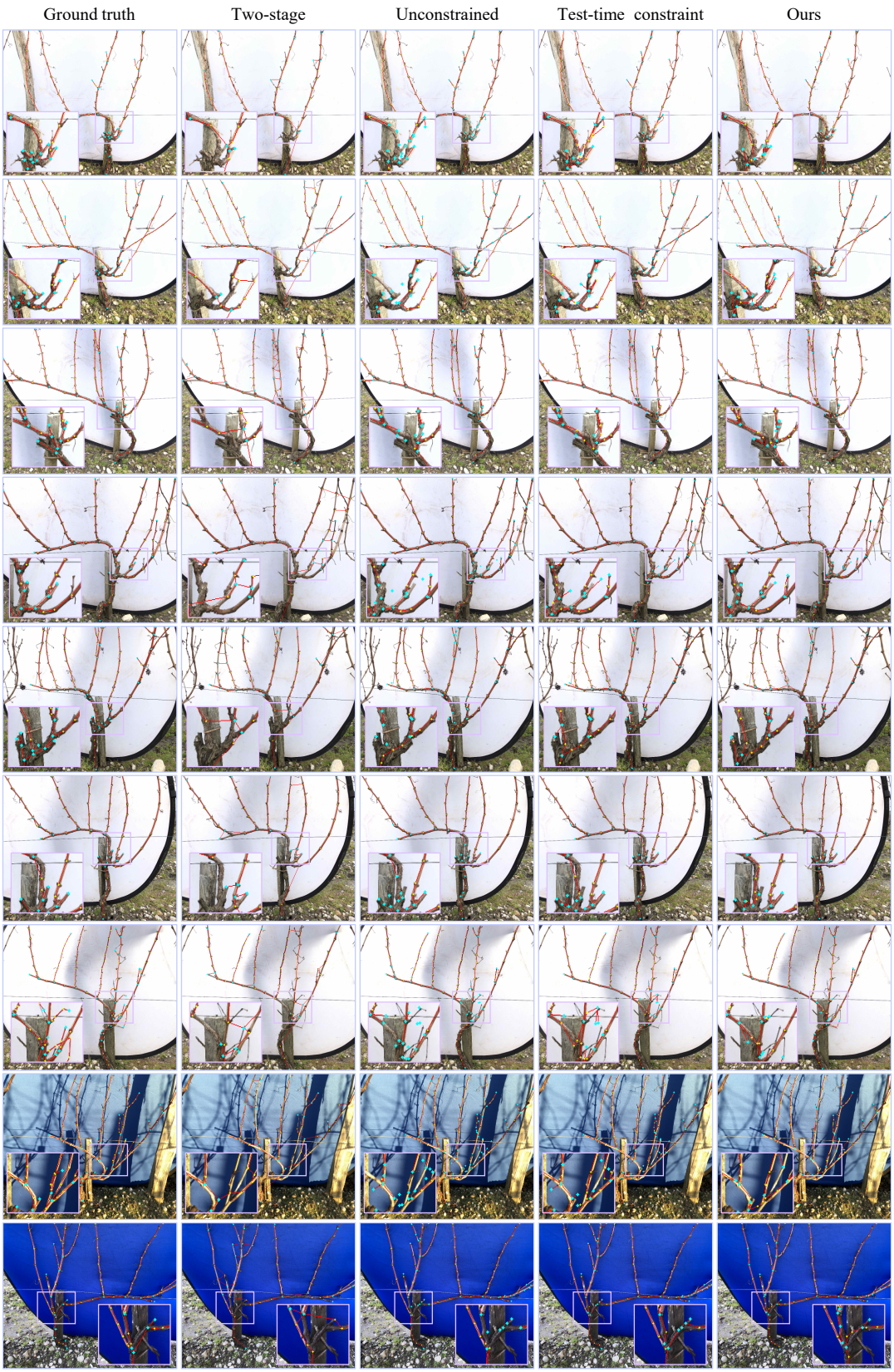


Figure S8. Additional results for the grapevine dataset (cont'd).

Figure S9. Additional results for the out-of-domain test dataset.