A New Paradigm for Identifying Reconciliation-Scenario Altering Mutations Conferring Environmental Adaptation (Supplementary Materials)

Roni Zoller
Ben Gurion University of the Negev, Israel
ronizo@post.bgu.ac.il

Meirav Zehavi
Ben Gurion University of the Negev, Israel[1]
meiravze@bgu.ac.il

Michal Ziv-Ukelson
Ben Gurion University of the Negev, Israel[1]
michaluz@cs.bgu.ac.il

1 Additional Figures

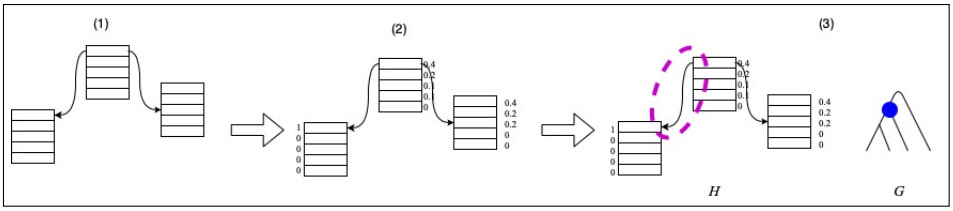

Figure S1 High-level overview of the RSAM-finder algorithm.

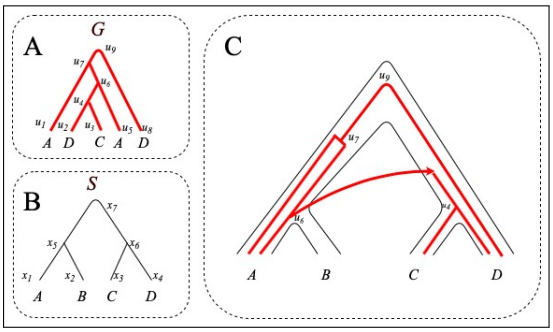

Figure S2 An example of a DLT scenario. (A) The Gene tree G. (B) The Species tree S. (C) A possible reconciliation scenario between G and S.

Fig. S2 demonstrates a DLT scenario. The species are written below the leaves of S. The (non-injective) mapping $\sigma : L(G) \to L(S)$ is implied by the labels of the leaves of G: $\sigma(u_1) = x_1$; $\sigma(u_2) = x_4$; $\sigma(u_3) = x_3$; $\sigma(u_5) = x_1$; $\sigma(u_8) = x_4$. In the DLT reconciliation of

[1] Corresponding authors.

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).
Editors: Katharina T. Huber and Dan Gusfield; Article No. 9; pp. 9:1–9:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

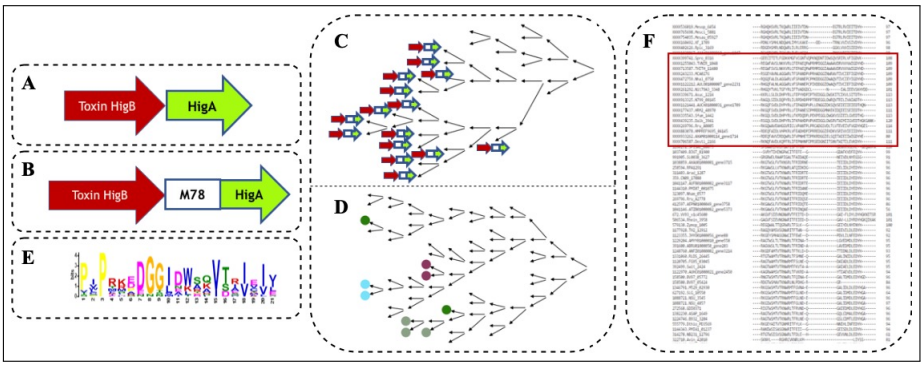

Figure S3 Application of RSAM-finder to the identification of two distinct chromosomal-invasion patterns of the higBA TA. (A) Two-component variant identified by Q_1, consisting of the toxin higB (red) and its antitoxin higA (green), and characterized by a fast duplication rate. (B) Three-component variant of higBA identified by Q_2, characterized by a slow duplication rate and far-reaching horizontal transfers. Predicted RSAM consists of a short insertion sequence in higB (shown in E) and the fusion of the antitoxin higA with an M78 peptidase domain. (C) Top-ranking subtree of the higB toxin gene tree identified by Q_2. Leaves corresponding to instances of the three-component toxin system are marked. (D) Top-ranking subtree of the higB gene tree identified by Q_1. Vertices marked with circles of the same color represent multiple copies of the gene in the same species. (E) A motif logo for the insertion sequence mentioned in B. (F) The multiple alignment of the higB protein sequences, illustrating the insertion sequence (highlighted by red rectangle).

G, S and σ (Fig. S2.C), the tubes illustrate the edges of S, and each edge of G is embedded inside the tube (edge of S) to which it is mapped by γ. Then, $\Sigma = \{u_9, u_4\}$, $\Delta = \{u_7\}$ and $\Theta = \{u_6\}$. Moreover, $\Xi = \{(u_6, u_4)\}$.

2 Pseudocode of Algorithms for Hypergraph Construction

In Fig. 1 in Section 1, the hypernode $(u_5, x_5, 1)$ is annotated with score 1 and event 'S'. To extract the best solution from the hypergraph, we begin with the first (i.e. top-ranking) slot in the root of the hypergraph (which is $(u_5, x_5, 1)$ in the figure), and then follow the incoming hyperedges in top-down order. In the figure, the best solution is (1) in part C. To extract it from the hypergraph in part B, we map u_5 to x_5 with cost 1 and a speciation event. Then, by first following the hyperedges incoming to $(u_5, x_5, 1)$, we derive the mapping of u_3 to x_1 and of u_4 to x_3, and by secondly following the hyperedges incoming to $(u_3, x_1, 1)$, we also derive the mapping of u_1 to x_1 and of u_2 to x_2.

The second best solution ((2) in part C) is extracted in the same manner—now, we start with the hypernode $(u_5, x_5, 2)$ rather than $(u_5, x_5, 1)$, and again follow incoming hyperedges in a top-down order until we reach the leaves. Similarly, we can extract all three non-nil solutions among the 4-best solutions (illustrated in part C). As before, the outer tubes illustrate the edges of S, and the edges of G are embedded inside based on the reconciliation.

2.1 An Efficient Hypergraph Construction

An overview of our efficient algorithm can be found in Section 5.1. Here, we present the pseudocode for this algorithm.

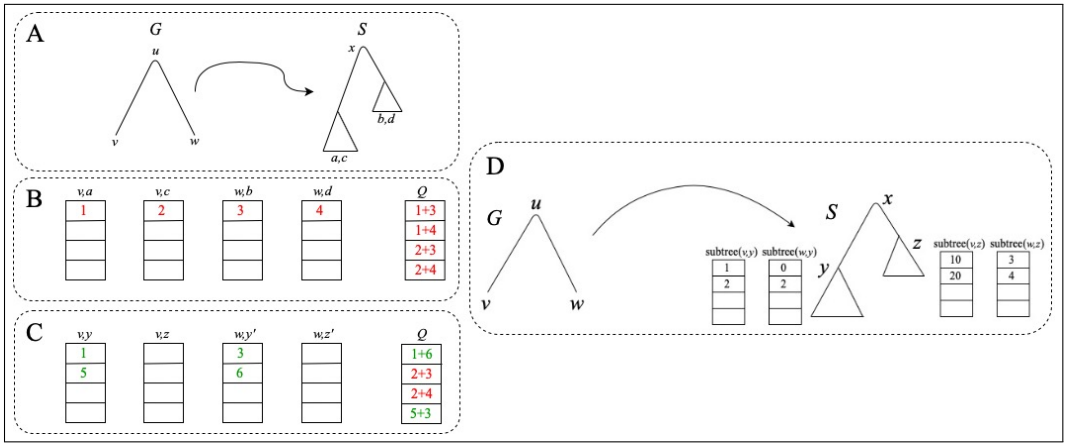

Figure S4 (A),(B) and (C) Demonstration of subroutine 1 of the naive algorithm. (Subroutines 2 and 3 are similar.) (D) The computation in line 21 of the efficient algorithm for hypergraph construction.

In this appendix we use the notation $i\mathsf{min}$, defined as follows: Let X and Y be sets, and consider a function $f : X \to Y$, and an index $i \in \{1, \ldots, |X|\}$. Then, $i\mathsf{min}_{x' \in X} f(x') \stackrel{\Delta}{=} f(x)$ where x is an element in X such that there are exactly i elements $x' \in X$ satisfying $f(x') \leq f(x)$. In case f is not an injective function, hence there multiple choices for x, we break ties arbitrarily.

Efficient DP Algorithm for Constructing the Hypergraph

1: **for** $u \in V(G),\ x \in V(S), 1 \leq i \leq k$ **do**
2: Set $c(u,x,i), p_\Sigma(u,x,i), p_\Delta(u,x,i), p_\Theta(u,x,i), \mathsf{subtree}(u,x,i)$ and $\mathsf{incomp}(u,x,i)$ to ∞
3: **end for**
4: **for** $u \in L(G)$ **do**
5: Add $(u, \sigma(u))$ to the supernode set of the hypergraph
6: $\mathsf{event}(u, \sigma(u), 1) \leftarrow \mathsf{leaf}$ $\mathsf{c}(u, \sigma(u), 1) \leftarrow 0$ $\mathsf{subtree}(u, \sigma(u), 1) \leftarrow 0$
7: **for** i from 2 to k **do**
8: $\mathsf{event}(u,x,i) \leftarrow \mathsf{NaN}$ $\mathsf{c}(u,x,i) \leftarrow \infty$
9: **end for**
10: **end for**
11: **for** $u \in I(G)$ with children v, w in postorder **do**
12: **for** $x \in V(S)$ in postorder **do**
13: **for** i from 1 to k **do**
14: **if** $x \in L(S)$ **then**
15: $p_\Sigma(u,x,i) = \infty$
16: $p_\Delta(u,x,i) \leftarrow c_\Delta + i\mathsf{min}_{1 \leq j,r \leq k}\{\mathsf{c}(v,x,j) + \mathsf{c}(w,x,r)\}$
17: **if** x is not the root of S **then**
18: $p_\Theta(u,x,i) \leftarrow c_\Theta + i\mathsf{min}_{1 \leq j,r \leq k} \begin{cases} \mathsf{subtree}(v,x,j) + \mathsf{incomp}(w,x,r), \\ \mathsf{subtree}(w,x,j) + \mathsf{incomp}(v,x,r) \end{cases}$
19: **end if**
20: **else** $x \in I(S)$ with children y, z
21: $p_\Sigma(u,x,i) \leftarrow i\mathsf{min}_{1 \leq j,r \leq k} \begin{cases} \mathsf{subtree}(v,y,j) + \mathsf{subtree}(w,z,r), \\ \mathsf{subtree}(w,y,j) + \mathsf{subtree}(v,z,r) \end{cases}$

22: $$p_{\Delta}(u,x,i) \leftarrow c_{\Delta} + i\mathsf{min}_{1\leq j,r\leq k} \begin{cases} \mathsf{subtree}(v,y,j) + \mathsf{subtree}(w,z,r), \\ \mathsf{subtree}(w,y,j) + \mathsf{subtree}(v,z,r), \\ \mathsf{subtree}(v,y,j) + \mathsf{subtree}(w,y,r), \\ \mathsf{subtree}(v,z,j) + \mathsf{subtree}(w,z,r), \\ \mathsf{c}(v,x,j) + \mathsf{c}(w,x,r), \\ \mathsf{c}(v,x,j) + \mathsf{subtree}(w,z,r), \\ \mathsf{c}(w,x,j) + \mathsf{subtree}(v,y,r), \\ \mathsf{c}(v,x,j) + \mathsf{subtree}(w,y,r), \\ \mathsf{c}(w,x,j) + \mathsf{subtree}(v,z,r) \end{cases}$$

23: **if** x is not the root of S **then**

24: $$p_{\Theta}(u,x,i) \leftarrow c_{\Theta} + i\mathsf{min}_{1\leq j,r\leq k} \begin{cases} \mathsf{subtree}(v,x,j) + \mathsf{incomp}(w,x,r), \\ \mathsf{subtree}(w,x,j) + \mathsf{incomp}(v,x,r) \end{cases}$$

25: **end if**
26: **end if**
27: **end for**
28: Find k-best scenarios based on the lists $p_{\Sigma}(u,x)$, $p_{\Delta}(u,x)$ or $p_{\Theta}(u,x)$
29: Construce the coresponding hypernodes and connect hyperedges accordingly
30: $\mathsf{subtree}(u,x) \leftarrow k$-best list from $\mathsf{c}(u,x), \mathsf{subtree}(u,y)$ and $\mathsf{subtree}(u,z)$
31: **end for**
32: **for** $x \in I(S)$ with children y,z in preorder **do**
33: $\mathsf{incomp}(u,y) \leftarrow k$-best list from $\mathsf{incomp}(u,x), \mathsf{subtree}(u,z)$
34: $\mathsf{incomp}(u,z) \leftarrow k$-best list from $\mathsf{incomp}(u,x), \mathsf{subtree}(u,y)$
35: **end for**
36: **end for**

We proceed with a few clarifications of the pseudocode.

Initialization: Lines 1-10. We initialize all lists to contains only scores of ∞ (lines 1-3). Then, the lists associated with a matching between leaves that complies with σ—that is, supernodes of the form $(u,\sigma(u))$ for some $u \in V(G)$—are inserted into the hypergraphs, and their topmost items are updated with a leaf event, cost 0 and subtree 0 (because the cost of the best solutions mapping a gene to its species has the cost 0).

Division into First and Second Phases: Lines 11-36. For each vertex $u \in I(G)$ in postorder (line 11), we have two phases, on which we elaborate below. In the first phase (lines 12-31), we consider each vertex $x \in V(S)$ in postorder and perform most computations, and in the second phase (lines 32-35) we consider each vertex $x \in V(S)$ in postorder and compute the lists of incomp.

Recursive Formulas for p_{Σ}, p_{Δ} and p_{Θ}: Lines 13-27. In this part of the first phase, we find the k-best costs for mapping the subtree of G rooted in u to the subtree of S rooted in x for each possible event (speciation, duplication or horizontal transfer), based on computations done in previous iterations or the initialization. The recursive formulas for these computations are directly given in the pseudocode.

In Fig. S4.D, we demonstrate the construction of $p_{\Sigma}(u,x,i)$ for $i \in \{1,\dots k\}$. Given $u \in V(G)$ and $x \in V(S)$, we need to map the subtree of u to the subtree of x according to

the restrictions of a speciation event. Thus, we need to map the left and right children of u to some vertices in the subtrees of the left and right (or, alternatively, right and left) children of x, respectively. In the previous (naive) algorithm, this was done by simply always iterating over all the vertices in these subtrees and thereby finding the best scores. Here, we only need to look at two pairs of lists, $\{\mathsf{subtree}(v,y),\mathsf{subtree}(w,z)\}$ and $\{\mathsf{subtree}(v,z),\mathsf{subtree}(w,y)\}$, in order to find the best scores. Even though the best sum is $\mathsf{subtree}(v,y,1)+\mathsf{subtree}(w,y,1)=1$, it does not correspond a valid solution (since then v and w are mapped to the same subtree). Here, the best sum corresponding to a valid solution is $\mathsf{subtree}(v,y,1)+\mathsf{subtree}(w,z,1)=4$, where v is mapped to some y' in the subtree rooted in y, and similarly w is mapped to some z' in the subtree rooted in z.

Updating c and $\mathsf{subtree}$ in First Phase: Lines 29-30. First, in line 29, we immediately find k-best costs for mapping the subtree of u to the subtree of x (i.e. we compute $\mathsf{c}(u,x)$) by selecting k-best costs from the list that is the combination of $p_\Sigma(u,x)$, $p_\Delta(u,x)$ and $p_\Theta(u,x)$. Notice that in this line, we also add the appropriate vertices and hyperedges to the hypergraph, in the exact same way in which it was done in the naive algorithm. $\mathsf{event}(u,x,i)$ is defined by the source list ($p_\Sigma(u,x)$, $p_\Delta(u,x)$ or $p_\Theta(u,x)$) it came from. As before, if the combined list is shorter then k, we add hypernodes with $\mathsf{event}=\mathsf{Nan}$ and $\mathsf{cost}=\infty$. Secondly, in line 30, we find k-best costs for mapping the subtree of u to the subtree of some vertex x' in the subtree of x (i.e. we compute $\mathsf{subtree}(u,x)$) by selecting k-best costs from the list defined as follows. In case $x\in I(S)$, then this list is the combination of $\mathsf{c}(u,x)$, $\mathsf{subtree}(u,y)$ and $\mathsf{subtree}(u,z)$, where y and z are the children of x in S. Otherwise, this list is simply $\mathsf{c}(u,x)$.

Updating incomp in Second Phase: Lines 32-35. To compute the lists of the form $\mathsf{incomp}(u,\cdot)$, in the second phase we iterate over all vertices $x\in I(S)$ with children y and z in *preorder*. We note that now the traversal of S is in preorder rather than postorder because the computation of a list $\mathsf{incomp}(u,a)$ for a vertex $a\in V(S)$ that is not the root of S relies on having already computed the list $\mathsf{incomp}(u,b)$ where b is the parent of a in S. Specifically, for a vertex $x\in I(S)$ with children y and z, we compute the list $\mathsf{incomp}(u,y)$ by selecting k-best costs from the list that is the combination of $\mathsf{incomp}(u,x)$ and $\mathsf{subtree}(u,z)$, and symmetrically for $\mathsf{incomp}(u,z)$ (swapping the roles of y and z).

Time Complexity and Correctness. We now turn to prove Observation 2 and Lemma 1.

Proof of Observation 2. For each pair of vertices $u\in V(G)$ and $x\in V(S)$, we construct a tuple of lists $(p_\Sigma(u,x),p_\Delta(u,x),p_\Theta(u,x),\mathsf{subtree}(u,x),\mathsf{incomp}(u,x))$. From the pseudocode, it is clear that the computation of each one of these lists is done in time $O(k)$. Thus, we have that the total running time is $O(m\cdot n\cdot k)$. ◀

Proof of Lemma 1. We can focus on proving that for every pair of vertices $u\in V(G)$ and $x\in V(S)$, and every index $i\in\{1,\ldots,k\}$, if there exists an i^{th} best DLT scenario mapping the subtree of G rooted in u to the subtree of S rooted in x, then the hypernode (u,x,i) is inserted into the hypergraph $\mathcal{H}$ under construction with association to this scenario. In this lemma, the proof of this claim is done in conjunction with the proof that for every pair of vertices $u\in V(G)$ and $x\in V(S)$, and every index $i\in\{1,\ldots,k\}$, the following equalities hold.

- $\mathsf{subtree}(u,x,i)$ is the i^{th} best cost of a DLT scenario mapping the subtree of G rooted in u to some subtree of S whose root is a vertex y that is a descendant of x.

- $\mathsf{incomp}(u, x, i)$ is the i^{th} best cost of a DLT scenario mapping the subtree of G rooted in u with some subtree of S whose root is a vertex y that is incomparable to x.

The proof is by induction on the order of computation.
In particular, $\mathsf{table}_1(u_1, x_1) < \mathsf{table}_2(u_2, x_2)$ where $\mathsf{table}_1, \mathsf{table}_2 \in \{\mathsf{c}, \mathsf{subtree}, \mathsf{incomp}\}$ if one of the following conditions holds:

- u_1 is visited before u_2 in the postorder traversal of G.
- $u_1 = u_2$, $\mathsf{table}_1, \mathsf{table}_2 \in \{\mathsf{c}, \mathsf{subtree}\}$, and x_1 is visited before x_2 in the *postorder* traversal of S.
- $u_1 = u_2$, $x_1 = x_2$, $\mathsf{table}_1 = \mathsf{c}$ and $\mathsf{table}_2 = \mathsf{subtree}$.
- $u_1 = u_2$, $\mathsf{table}_1 = \mathsf{subtree}$ and $\mathsf{table}_2 = \mathsf{incomp}$.
- $u_1 = u_2$, $\mathsf{table}_1 = \mathsf{table}_2 = \mathsf{incomp}$, and x_1 is visited before x_2 in the *preorder* traversal of S.

The basis of the induction comprises of the computation of hypernodes of the form (u, x, i) where $u \in L(G)$. To prove its correctness, consider such a hypernode (u, x, i). If $i = 1$, then the algorithm inserts the hypernode $(u, x, 1)$ (whose event is leaf) with score 0 and $\mathsf{subtree}$ value 0; otherwise, the algorithm does not insert the hypernode—more precisely, it inserts a place-holder (whose event is NaN) with score ∞ and $\mathsf{subtree}$ value ∞. In both cases, incomp value remains ∞ as in its creation. The correctness of these operations directly follows from the definitions of $\mathsf{subtree}$ and incomp, and the fact that the only possible DLT scenario in this case maps u to x, and the score of this match is 0.

For the inductive step, we consider some pair of vertices $u \in I(G)$ and $x \in V(S)$ along with a table $\mathsf{table} \in \{\mathsf{c}, \mathsf{subtree}, \mathsf{incomp}\}$, and prove that the values in table of the supernode (u, x) are computed correctly. For the inductive assumption, suppose that for every triple $(\mathsf{table}', u', x')$ ordered before (table, u, x), the values in table' of (u', x') have already been computed correctly.

First, consider the case where $\mathsf{table} = \mathsf{subtree}$. By the pseudocode, $\mathsf{subtree}(u, x)$ consists of the k-best scores from the lists $\mathsf{c}(u, x), \mathsf{subtree}(u, y)$ (if x is not a leaf) and $\mathsf{subtree}(u, z)$ (if x is not a leaf). Observe that these three lists have already been computed. Thus, by the inductive hypothesis, $\mathsf{c}(u, x)$ consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in u to the subtree of S rooted in x (where u and x are matched to each other), $\mathsf{subtree}(u, y)$ consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in u to the subtree of S rooted in some descendant of y (if x is not a leaf), and $\mathsf{subtree}(u, z)$ consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in u to the subtree of S rooted in some descendant of z (if x is not a leaf). Notice that a DLT scenario maps the subtree of G rooted in u to the subtree of S rooted in some descendant of x if and only if it is a DLT scenario that maps the subtree of G rooted in u to one of the following subtrees: *(i)* the subtree of S rooted in x (where u and x are matched); *(ii)* a subtree of S rooted in some descendant of y (if x is not a leaf); *(iii)* a subtree of S rooted in some descendant of z (if x is not a leaf). Thus, it follows that $\mathsf{subtree}(u, x)$ is computed correctly.

Second, consider the case where $\mathsf{table} = \mathsf{incomp}$. By the pseudocode, if x is the root of S, then $\mathsf{incomp}(u, x)$ does not contain any item (having score different from ∞) as in its creation, which is correct because in this case, there exists no vertex incomparable to x and hence we cannot map one of the children of u as required in the definition of the DLT scenarios that correspond to $\mathsf{incomp}(u, x)$. Therefore, now suppose that x is not the root of S, and let p denote the parent of x in S, and s denote the sibling of x in S (i.e. the other child of p in S). Then, by the pseudocode, $\mathsf{incomp}(u, x)$ consists of the k-best scores from the

lists $\mathsf{incomp}(u, p)$ and $\mathsf{subtree}(u, s)$. Observe that these two lists have already been computed. Thus, by the inductive hypothesis, $\mathsf{incomp}(u, p)$ consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in u to the subtree of S rooted in some vertex incomparable to p, and $\mathsf{subtree}(u, s)$ consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in u to the subtree of S rooted in some descendant of s. Notice that a DLT scenario maps the subtree of G rooted in u to a subtree of S rooted in some vertex incomparable to x if and only if it is a DLT scenario that maps the subtree of G rooted in u to one of the following subtrees: *(i)* a subtree of S rooted in some vertex incomparable to p; *(ii)* a subtree of S rooted in some descendant of s. Thus, it follows that $\mathsf{incomp}(u, x)$ is computed correctly.

Third, consider the (last) case where $\mathsf{table} = \mathsf{c}$. By line 29 of the pseudocode, $\mathsf{c}(u, x)$ consists of the k-best scores from the lists $p_\Sigma(u, x)$, $p_\Delta(u, x)$ and $p_\Theta(u, x)$. Thus, to prove the correctness of the computation of $\mathsf{c}(u, x)$, it suffices to prove that the following statement holds: $p_\Sigma(u, x)$, $p_\Delta(u, x)$ and $p_\Theta(u, x)$ consist of k-best DLT scenarios mapping the subtree of G rooted in u to the subtree of S rooted in x under the constraint that the event corresponding to the matching of u and x is speciation, duplication and horizontal transfer, respectively.

Towards the proof of the statement, consider the list $p_\Sigma(u, x)$. If $x \in L(S)$, then because $u \in I(G)$, there does not exist a DLT scenario mapping the subtree of G rooted in u to the subtree of S rooted in x under the constraint that the event corresponding to the matching of u and x is speciation, and hence the assignment of ∞ to every element $p_\Sigma(u, x, i)$ of the list is correct. Now, suppose that $x \in I(S)$. Then, by the pseudocode, $p_\Sigma(u, x)$ consists of the k-best scores present in the following multisets: $\{\mathsf{subtree}(v, y, j) + \mathsf{subtree}(w, z, j) \mid 1 \leq j, r \leq k\}$ and $\{\mathsf{subtree}(w, y, j) + \mathsf{subtree}(v, z, j) \mid 1 \leq j, r \leq k\}$. Observe that the lists $\mathsf{subtree}(v, y), \mathsf{subtree}(w, z), \mathsf{subtree}(w, y)$ and $\mathsf{subtree}(v, z)$ have already been computed. Thus, by the inductive hypothesis, $\mathsf{subtree}(v, y)$ (resp., $\mathsf{subtree}(w, z), \mathsf{subtree}(w, y)$ and $\mathsf{subtree}(v, z)$) consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in v (resp., w, w and v) to a subtree of S rooted in some descendant of y (resp., z, y and z). Notice that a DLT scenario maps the subtree of G rooted in u to a subtree of S rooted in x under the constraint that the event corresponding to the matching of u and x is speciation if and only if it is a DLT scenario that matches u and x, maps the subtree of G rooted in v to a subtree of S rooted in a descendant of one child (y or z) of x, and the subtree of G rooted in w to a subtree of S rooted in a descendant of the other child of x. Thus, it follows that $p_\Sigma(u, x)$ is computed correctly.

Now, consider the list $p_\Delta(u, x)$. In case $x \in I(S)$, let y and z denote its children. By the pseudocode, $p_\Delta(u, x)$ consists of the k-best scores obtained by adding c_Δ to the costs present in the following multisets, where only the first one is relevant in case $x \in L(S)$: $\{\mathsf{c}(v, x, j) + \mathsf{c}(w, x, r) \mid 1 \leq j, r \leq k\}$, $\{\mathsf{c}(v, x, j) + \mathsf{subtree}(w, z, r) \mid 1 \leq j, r \leq k\}$, $\{\mathsf{c}(w, x, j) + \mathsf{subtree}(v, y, r) \mid 1 \leq j, r \leq k\}$, $\{\mathsf{c}(v, x, j) + \mathsf{subtree}(w, y, r) \mid 1 \leq j, r \leq k\}$, $\{\mathsf{c}(w, x, j) + \mathsf{subtree}(v, z, r) \mid 1 \leq j, r \leq k\}$, $\{\mathsf{subtree}(v, y, j) + \mathsf{subtree}(w, z, r) \mid 1 \leq j, r \leq k\}$, $\{\mathsf{subtree}(w, y, j) + \mathsf{subtree}(v, z, r) \mid 1 \leq j, r \leq k\}$, $\{\mathsf{subtree}(v, y, j) + \mathsf{subtree}(w, y, r) \mid 1 \leq j, r \leq k\}$ and $\{\mathsf{subtree}(v, z, j) + \mathsf{subtree}(w, z, r) \mid 1 \leq j, r \leq k\}$. Observe that the lists $\mathsf{c}(v, x), \mathsf{c}(w, x), \mathsf{subtree}(w, y), \mathsf{subtree}(v, z), \mathsf{subtree}(w, z)$ and $\mathsf{subtree}(v, y)$ have already been computed. Thus, by the inductive hypothesis, we have that *(i)* $\mathsf{c}(v, x)$ (resp., $\mathsf{c}(w, x)$) consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in v (resp., w) to the subtree of S rooted in x, and *(ii)* $\mathsf{subtree}(v, y)$ (resp., $\mathsf{subtree}(w, z), \mathsf{subtree}(w, y)$ and $\mathsf{subtree}(v, z)$) consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in v (resp., w, w and v) to the subtree of S rooted in some descendant of y (resp., z, y and z). Notice that a DLT scenario maps the subtree of G rooted in u to a subtree of S

rooted in x under the constraint that the event corresponding to the matching of u and x is duplication if and only if it is a DLT scenario that matches u and x, maps the subtree of G rooted in v to a subtree of S rooted in a descendant of x (which can be x itself), and the subtree of G rooted in w to a subtree of S rooted in a descendant of x (which can be x itself). Thus, it follows that $p_{\Delta}(u,x)$ is computed correctly.

Lastly, consider the list $p_{\Theta}(u,x)$. If x is the root of S, then there does not exist a DLT scenario mapping the subtree of G rooted in u to the subtree of S rooted in x under the constraint that the event corresponding to the matching of u and x is horizontal transfer (because there is no vertex incomparable to x to whom one of the children of u should be mapped), and hence it is correct that each element $p_{\Theta}(u,x,i)$ remains with the assignment of ∞ as it was created. Now, suppose that x is not the root of S. Then, by the pseudocode, $p_{\Theta}(u,x)$ consists of the k-best scores present in the following multisets: $\{\mathsf{subtree}(v,x,j)+\mathsf{incomp}(w,x,r) \mid 1 \leq j,r \leq k\}$ and $\{\mathsf{subtree}(w,x,j)+\mathsf{incomp}(v,x,r) \mid 1 \leq j,r \leq k\}$. Observe that the lists $\mathsf{subtree}(v,x), \mathsf{subtree}(w,x), \mathsf{incomp}(w,x)$ and $\mathsf{incomp}(v,x)$ have already been computed. Thus, by the inductive hypothesis, we have that *(i)* $\mathsf{subtree}(v,x)$ (resp., $\mathsf{subtree}(w,x)$) consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in v (resp., w) to a subtree of S rooted in some descendant of x (which can be x itself), and *(ii)* $\mathsf{incomp}(v,x)$ (resp., $\mathsf{incomp}(w,x)$) consists of the (scores of the) k-best DLT scenarios mapping the subtree of G rooted in v (resp., w) to a subtree of S rooted in some vertex incomparable to x. Notice that a DLT scenario maps the subtree of G rooted in u to a subtree of S rooted in x under the constraint that the event corresponding to the matching of u and x is horizontal transfer if and only if it is a DLT scenario that matches u and x, maps the subtree of G rooted in one of the children of u (v or w) to a subtree of S rooted in a descendant of of x (which can be x itself), and the subtree of G rooted in the other child of u to a subtree of S rooted in a vertex incomparable to x. Thus, it follows that $p_{\Sigma}(u,x)$ is computed correctly. ◀

3 Assigning Probabilities

Proof of Lemma 3. We will verify a stronger property than the one in the statement of the lemma: For every vertex u in the Gene tree G, it holds that

$$\sum_{x,i:(u,x,i)\in V(\mathcal{H})} \mathsf{w}(u,x,i) = 1.$$

Before we verify this property, observe that when u is a leaf, then $\mathsf{c}(u,x,1) = 0$ for the unique vertex x that is compatible with u, and $\mathsf{c}(u,x,i) = \infty$ (which means that $\mathsf{D}(u,x,i) = \emptyset$ and hence $\mathsf{w}(u,x,i) = 0$) for any other pair (x,i). Thus, the stronger property implies the correctness of the weaker statement regarding leaves.

To prove the (stronger) property above, we use induction. In the basis, u is the root of the Gene tree G. Then, we have that $\sum_{x,i:(u,x,i)\in V(\mathcal{H})} \mathsf{w}(u,x,i) = \sum_{i\in\{1,2,\ldots,k\}} \mathsf{w}(r,i) = 1$, and therefore the property holds. Now, suppose that u is not the root of G, and that the property holds for each of its ancestors. Let v be the parent of u in $\mathcal{H}$. Then, we have that

$$\begin{aligned}\sum_{x,i:(u,x,i)\in V(\mathcal{H})} \mathsf{w}(u,x,i) &= \sum_{x,i:(u,x,i)\in V(\mathcal{H})} \sum_{y,j:(v,y,j)\in \mathsf{D}(u,x,i)} \mathsf{w}(v,y,j) \\ &= \sum_{y,i:(v,y,i)\in V(\mathcal{H})} \mathsf{w}(v,y,i) = 1.\end{aligned}$$

Here, the first equality followed directly from the definition of weights. The second equality followed from the fact that each hypernode (v,y,i) (for any y and i) that has positive weight

is derived from exactly one hypernode (u, x, j) (for some specific x and j). (However, each hypernode (u, x, j) can be used to derive several hypernodes (v, y, i).) The last equality followed from the inductive hypothesis. This completes the proof. ◀

4 Deferred Text

Fig. 1 demonstrates the second stage of the $((\{\mathsf{HT}\}, \mathsf{red}, \mathsf{True}), (\{\mathsf{S}, \mathsf{D}, \mathsf{HT}\}, \mathsf{black}, \mathsf{False})$. The figure is an example of one possible reconciliation, and here we seek a vertex of the Gene tree the has a one child with lot of red-to-red horizontal transfers below it, and the other with a lot of black events. For convenience, we marked a vertex of the species tree, but it actually is the gene tree vertex which was mapped to it in the reconciliation.

4.1 Stage 3.1: Coloring the Species and Gene Trees (Optional)

Our framework can be used for searching *different* evolutionary patterns. *Some* sought patterns might involve a phenotype of the species. Therefore, we assume that (as a part of the input), each leaf of the Species tree S *might* have a color (red or black) to represent its phenotype. In this paper, we use colors as *habitat identifiers*. More precisely, we say that a leaf $\ell \in L(S)$ has an *environmental habitat* (soil, water, plants, etc.) if $\mathsf{black}(\ell) = 1$ and $\mathsf{red}(\ell) = 0$, and a *living habitat* (human, animal) if $\mathsf{black}(\ell) = 0$ and $\mathsf{red}(\ell) = 1$.

The goal here is to assign to each vertex $x \in V(S)$ a color $\mathsf{red}(x), \mathsf{black}(x) \in \mathbb{N}_0$ as explained below.[2]

Computational of the Assignment. Recall that as part of the input, we are given a function $\mathsf{colors} : L(S) \to \Upsilon$ where $\Upsilon = \{\mathsf{red}, \mathsf{black}\}$. Initially, for each leaf $x \in L(S)$, we define $\mathsf{red}(x) = 1$ and $\mathsf{black}(x) = 0$ if x is red (i.e. $\mathsf{colors}(x) = \mathsf{red}$), and $\mathsf{red}(x) = 0$ and $\mathsf{black}(x) = 1$ otherwise. Now, in postorder, for each internal vertex $x \in V(S)$ with children y and z, compute: $\mathsf{red}(x) = \mathsf{red}(y) + \mathsf{red}(z)$, and $\mathsf{black}(x) = \mathsf{black}(y) + \mathsf{black}(z)$.

Additionally, we use $\sigma : L(G) \to L(S)$ to color the vertices of the Gene tree G as follows. Initially, for each leaf $u \in L(G)$, we define $\mathsf{red}(u) = 1$ and $\mathsf{black}(u) = 0$ if $\mathsf{colors}(\sigma(u)) = \mathsf{red}$, and $\mathsf{red}(u) = 0$ and $\mathsf{black}(u) = 1$ otherwise. Now, in postorder, for each internal vertex $x \in V(S)$ with children y and z, compute: $\mathsf{red}(x) = \mathsf{red}(y) + \mathsf{red}(z)$, and $\mathsf{black}(x) = \mathsf{black}(y) + \mathsf{black}(z)$.

4.1.1 Colored Pattern

When providing a pattern P with colors, we need to define the sought pattern. First, we will need some definitions,

Definitions. For each node $(u, x, i) \in V(\mathcal{H})$ define $\mathsf{red}(u, x, i) = \mathsf{red}(x)$, $\mathsf{black}(u, x, i) = \mathsf{black}(x)$, $\mathsf{totalL}(u, x, i) = \mathsf{red}(x) + \mathsf{black}(x)$. We define $\Pr[\text{red}] = \frac{\text{number of red leaves in } S}{|L(S)|}$, and, similarly $\Pr[\text{black}] = \frac{\text{number of black leaves in } S}{|L(S)|}$.

For a node $(u, x, i) \in V(\mathcal{H})$, we say the it is "mostly red" if it holds that

$$\sum_{r=\mathsf{red}(u,x,i)}^{\mathsf{totalL}(u,x,i)} \binom{\mathsf{totalL}(u, x, i)}{r} \cdot \Pr[\text{red}]^r \cdot \Pr[\text{black}]^{\mathsf{totalL}(u,x,i)-r} \leq p,$$

[2] The standard notation $\mathbb{N}_0$ denotes $\mathbb{N} \cup \{0\}$ where $\mathbb{N}$ is the set of natural numbers.

and is “mostly black" if it holds that

$$\sum_{r=\mathsf{red}(u,x,i)}^{\mathsf{totalL}(u,x,i)} \binom{\mathsf{totalL}(u,x,i)}{r} \cdot \Pr[\text{red}]^r \cdot \Pr[\text{black}]^{\mathsf{totalL}(u,x,i)-r} \geq 1-p,$$

where p is a predefined probability threshold. Intuitively, the formulas above are measuring how much a node in the hypergraph is a mapping between a Gene tree vertex $u \in V(G)$ and a Species tree vertex $x \in V(S)$, such that x is rooting a homogeneous (with respect to the colors) subtree. As the value p is smaller, we restrict the subtrees we define as “mostly red" (resp. “mostly black") to be more homogeneous.

For each $\mathsf{EV} \in \{\mathsf{S}, \mathsf{D}, \mathsf{HT}\}$, $\mathsf{color} \in \{\mathsf{red}, \mathsf{black}\}$ and $\mathsf{distance} \in \{\mathsf{True}, \mathsf{False}\}$, the pattern $(\mathsf{EV}, \mathsf{colors}, \mathsf{distance})$ is a hypernode (u, x, i) with children (v, y, j) and (w, z, r) (by children we mean that $(u, x, i) \in \mathsf{D}(v, y, j)$ and $(u, x, i) \in \mathsf{D}(w, z, r)$, as defined on Section 5.2), such that both (v, y, j) and (w, z, r) are “mostly color".

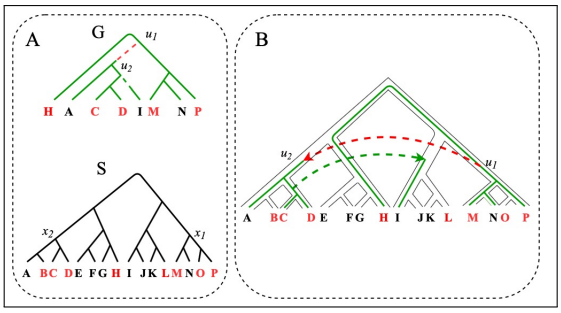

Figure S5 An illustration of $(\mathsf{HT}, \mathsf{red}, False)$ pattern

For example, Figure S5 demonstrates the pattern $(\mathsf{HT}, \mathsf{red}, False)$ inside a reconciliation (part B). The colors of the species are the colors of the text, this is $\mathsf{color}(B) = \mathsf{color}(C) = \mathsf{color}(D) = \mathsf{color}(H) = \mathsf{color}(L) = \mathsf{color}(M) = \mathsf{color}(O) = \mathsf{color}(P) = \mathsf{red}$ and $\mathsf{color}(A) = \mathsf{color}(E) = \mathsf{color}(F) = \mathsf{color}(G) = \mathsf{color}(I) = \mathsf{color}(J) = \mathsf{color}(K) = \mathsf{color}(N) = \mathsf{black}$. We illustrate one optimal solution for the input (part A). Note that this is not the hypergraph, see Figure S2 for more details about the representation. The bold red horizontal transfer edge is an edge that satisfies our requirements: a calculation for being “mostly red" for $(u_1, x_1, 1)$ is $\sum_{r=3}^{4} \binom{4}{r} \cdot \left(\frac{1}{2}\right)^r \cdot \left(\frac{1}{2}\right)^{4-r} = 0.3125 < 0.5$ and for $(u_2, x_2, 1)$ is the same. Thus, for $p = 0.5$ the hypernode representing the match between u_1 and x_1 will be considered as a pattern satisfying our requirements. On the other hand, the black bold horizontal transfer is not satisfying the requirements, and will not be reported as interesting. In all of our results, we use $p = 0.05$.

4.2 Stage 3.2

For each defined pattern $\mathsf{P} = (\mathsf{EV}, \mathsf{color}, \mathsf{distance})$, let $\mathsf{counter}_P : V(G) \to \mathbb{R}^+$ be a counter, initialized by 0. For each $u \in I(G)$ let v, w be its right and left children. Let $\mathcal{I}_u = \{(u, x, i) \in V(\mathcal{H}) : (u, x, i)$ is marked as interesting with respect to $P\}$. This is, for each vertex $x \in V(S)$ and $i \in \{1, \ldots, k\}$ such that $(u, x, i) \in V(\mathcal{H})$ was marked as interesting in stage 1 with respect to pattern P, $(u, x, i) \in \mathcal{I}_u$. Let

$$\mathsf{counter}_P(u) = \mathsf{counter}_P(v) + \mathsf{counter}_P(w) + \sum_{(u,x,i)\in\mathcal{I}_u} \mathsf{w}(u, x, i)$$

where $\mathsf{w}(u, x, i)$ are the probabilities assigned in Section 5.2. Intuitively, for each vertex $u \in V(G)$ we calculate its probability to be interesting, with respect to the patterns we defined. In order to avoid a bias, we normalise each value by the number of edges in the subtree root in the vertex times k, which is an upper bound on the number of possible patterns in all the solutions. That is, for each vertex $u \in V(G)$ and pattern P, let $\mathsf{counter}_P(u) = \frac{\mathsf{counter}_P(u)}{|E(G_u)| \cdot k}$.

5 Methods

5.1 Simulator and Algorithm Implementation

.

In order to further test our approach, we generate a random binary tree. The generation of a random binary tree is done in a top down manner: A list of vertices is maintained, initialized by the root. In each step, we choose randomly the leftmost or the rightmost item of the list, pop it, add two vertices (its children) to the right of the list. In order to unbalance the tree, we choose randomly 25% of the vertices, and connect the subtree rooted in them to a random leaf. First we duplicate this random tree and call one copy G and the other S. The function $\sigma : L(G) \to L(S)$ is the matching between each leaf in G to its copy in S, and the function $\mathsf{color} : L(S) \to \{\mathsf{red}, \mathsf{black}\}$ is a random binary function. All of the simulations use the pattern $(\{\mathsf{HT}\}, \mathsf{red}, \mathsf{True})$, in which we seek a subtree that is enriched in red-to-red horizontal transfer events. (For more details, see Section 5.3). In order to create the pattern in the random trees, we pick a random vertex $u \in V(G)$, and change the function $\sigma : L(G) \to L(S)$ for vertices $w \in G_u$ in a way that will create a Horizontal transfer event. Assume we pick a vertex $w \in G_u$, and we want to make it a Horizontal transfer event. Let $L(S_x)$ denote the copy of $L(G_w)$ (the leaves of the subtree rooted in w) in the Species tree, thus the function σ maps each leaf of G_w to its copy in the leaves of S_x. In order to create a horizontal transfer in w, we need to find a vertex $y \in V(S)$ such that y and x are incomparable, and change the mapping of the leaves of G_w to the leaves of S_y. This creates a Horizontal Transfer in the DLT-reconciliation with high probability. Recall that in addition, we want to make those planted HT events red-to-red events, thus we search for "mostly red" vertices when choosing which vertex we want to make interesting and where the horizontal transfers will be mapped to.

5.2 Noise

We tested our tool on the same random data with an additional noise factor on horizontal transfers and colors. Each noise level represents the number of random horizontal transfer edges and random colors of the species. 0% means no horizontal transfer edges were added except those of the planted pattern, and no change of the function $\mathsf{color} : L(S) \to \{\mathsf{red}, \mathsf{black}\}$ was made, and 100% means that all of the edges were randomly changed into horizontal transfer and all of the species colors were randomly picked again. The horizontal transfers modifications were applied by changing the mapping between a Gene tree leave and a corresponding Species tree leave, i.e. σ. The colors noise was applied by changing the colors of the leaf of S randomly.

Fig. S6.A demonstrates the advantage of our approach across different noise levels. It is a demonstration on a random phylogenetic simulation, with a planted pattern. First, we constructed a random pylogenetic Species and Gene trees with 600 leaves and one planted pattern (marked in purple in Fig. S6.A). (see Appendix S5.1 for more details). For each noise level between 0% to 20% we constructed the hypergraph. For all experiments, we used

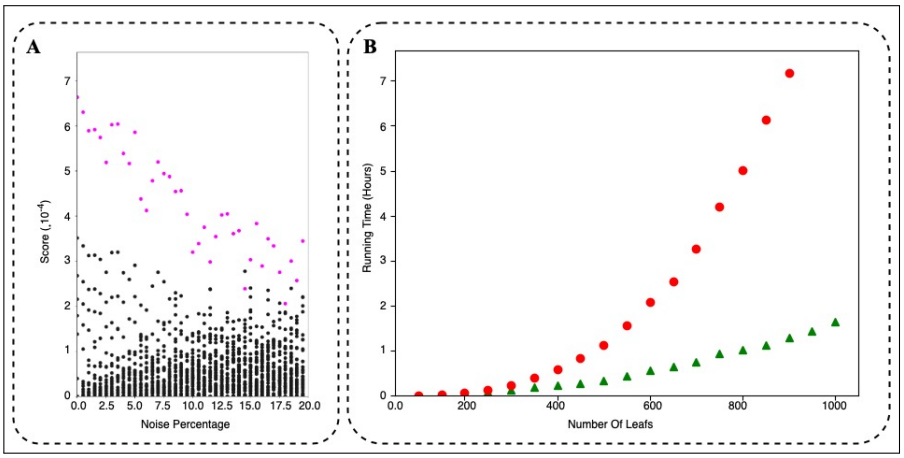

Figure S6 (A) The scores of the vertices in difference noise level on the input. Purple vertex is a pre-planted vertex, obeys the sought pattern (B) Running time of the naive and efficient algorithms. Green triangles encodes measurement for the efficient version and red circles corresponds to the naive version.

$k = 100$, minimum size of subtree to be 0.1% of all edges and $c_\Delta = c_\Theta = 1$. Each noise level is an average of 50 random choices for the same amount of random horizontal transfers and color changes. The scores are as defined in Section 5.3.

We found that the score of the planted vertex is higher than that of any other vertex at lower noise levels, and it decreases as the noise level increases. Note that the additional noise increases the false positive founding. That is, when adding more noise, the vertices that are computed to be interesting are not necessarily those we seek. These findings support the claim that our method is able to find a pattern within a noisy data. The results indicate a high level of reliability when testing the engine on real biological data, which is often very noisy.

5.3 Running Time

In orders to demonstrate the practicality of the theoretical improvements presented in Section 5.1, we compare the running time of the naive algorithm and the efficient version, by constructing hypergraphs for different inputs. We picked random binary trees with 100 to 1000 leaves. For each number of leaves, we randomly picked 10 trees, and constructed the corresponding hypergraphs in both methods, the naive and the efficient.

Fig. S6.B summarizes the running time test results. The green triangles are an average of the time measured for the efficient version of the algorithm, and the red circles are an average of the time measured for the naive version of the algorithm.

We found out that as we increase the number of leaves, the differences between the naive and the efficient algorithms becomes significant.