Optimal Subtree Prune and Regraft for Quartet Score in Sub-Quadratic Time

Shayesteh Arasti
Computer Science and Engineering Department, University of California, San Diego, CA, USA

Siavash Mirarab[1]
Electrical and Computer Engineering Department, University of California, San Diego, CA, USA

Abstract

Finding a tree with the minimum total distance to a given set of trees (the median tree) is increasingly needed in phylogenetics. Defining tree distance as the number of induced four-taxon unrooted (i.e., quartet) trees with different topologies, the median of a set of gene trees is a statistically consistent estimator of the species tree under several models of gene tree species tree discordance. Because of this, median trees defined with quartet distance are widely used in practice for species tree inference. Nevertheless, the problem is NP-Hard and the widely-used solutions are heuristics. In this paper, we pave the way for a new type of heuristic solution to this problem. We show that the optimal place to add a subtree of size m onto a tree with n leaves can be found in time that grows quasi-linearly with n and is nearly independent of m. This algorithm can be used to perform subtree prune and regraft (SPR) moves efficiently, which in turn enables the hill-climbing heuristic search for the optimal tree. In exploratory experiments, we show that our algorithm can improve the quartet score of trees obtained using the existing widely-used methods.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases Phylogenetics, Gene tree discordance, Quartet score, Quartet distance, Subtree prune and regraft, Tree search, ASTRAL

Digital Object Identifier 10.4230/LIPIcs.WABI.2023.4

Supplementary Material *Text (Figures)*: `https://doi.org/10.6084/m9.figshare.23582676`

Funding The work was supported by National Health Institute (NIH) grant R35GM142725.
Siavash Mirarab: was supported through National Science Foundation grant IIS 1845967.

1 Introduction

Phylogenetic reconstruction literature has now fully embraced Maddison's insight that gene trees and species trees differ [16], and a plethora of methods exist for accounting for such discordance when inferring a species tree [3]. A prominent biological cause of discordance is incomplete lineage sorting (ILS), a process that can never be fully discounted and is studied using the multi-species coalescent model (MSC) [23, 26]. Under the MSC model, the species tree is the parameter of a distribution that generates true gene trees disagreeing with the species tree due to ILS [20]. In developing methods that account for ILS, an easily obtained result [1] has proved invaluable: Under the MSC distribution parameterized by a four-taxon (i.e., quartet) species tree, the unrooted gene tree topology matching the species tree unrooted topology has a higher (marginal) probability than the two alternative topologies. This observation is what underlines many of the existing methods, including the popular methods ASTRAL [21], SVDQuartets [7], and others [14, 29]. As a result, the decades-old idea of inferring phylogenetic trees by dividing the dataset into quartets [9, 32, 31] has gained increased attention in recent years.

[1] corresponding author

A naive method dividing a set of n taxa into quartets will have one of two problems: it either has to scale with $\Omega(n^4)$, which is impractical for modern datasets, or it has to subsample quartets to an asymptotically smaller size. Neither is ideal, but also, neither is necessary. Take the simple question of computing the fraction of $\binom{n}{4}$ quartet topologies shared between two trees. Naively, we list all topologies and compare them, which requires $\Omega(n^4)$ time. However, a relatively straightforward algorithm [6] can compute the score using a post-traversal of one tree versus each node of the other tree in time that grows with $\Theta(n^2)$. But, we can do even better. As Brodal *et al.* have shown [4], the quartet score can be computed in $O(n \log^2(n))$ using a complex algorithm that uses a sophisticated data structure called Hierarchical Decomposition Tree (HDT) to represent trees. Thus, sub-quadratic (and hence, scalable) analysis using quartets is doable.

Beyond computing quartet distance, we can seek to minimize it. For an input set of k unrooted (gene) trees, the median quartet tree is the (species) tree that maximizes the number of input tree quartets shared with the output tree. Several theoretical and empirical results have shown that the median quartet tree is robust to not just ILS but also HGT [8] or duplication and loss [19, 15, 11], making the search for this tree a fruitful optimization objective. One of the most widely used solutions to this problem is ASTRAL [21], which approaches this NP-Hard [13] problem using a dynamic programming algorithm capable of precisely solving the problem; nevertheless, to achieve scalability, it restricts the search space to the trees that can be built from a predefined set $\mathcal{X}$ of clusters. With this construction, the running time of the latest version, ASTRAL-III [35], becomes $O(|\mathcal{X}|^{3/\log_3(27/4)}nk)$, which is $O((nk)^{2.726})$ because ASTRAL-III restricts $|\mathcal{X}| = O(nk)$; on typical datasets, the scaling seems to be close to n^2k^2 empirically. While ASTRAL is reasonably scalable, it does not take advantage of the HDT data structure proposed by Brodal *et al.* [4] (referred to as B13 henceforth). Moreover, unlike most methods used in practice, ASTRAL does not use a hill-climbing search algorithm and instead uses a dynamic programming approach pioneered by earlier work [10, 5, 33]. This leads to a question: Is the dynamic programming algorithm used in ASTRAL the most efficient and effective way to find the quartet distance median tree or would hill-climbing be more efficient?

Building a hill-climbing heuristic requires the ability to make tree rearrangements. Having computed the score of the current tree, T, computing the quartet score of Nearest Neighbour Interchange (NNI) moves around T is relatively simple and can be done efficiently given some pre-computations. However, NNIs are generally thought to be not enough for efficient search. All leading phylogenetic search tools use Subtree Prune and Regraft (SPR) rearrangements in addition. An SPR move on an *unrooted* query tree is defined on a node u and one of its three neighbors u'; it prunes the subtree pending from the $u \leftrightarrow u'$ edge (including that edge) and grafts back this subtree on another edge $v \leftrightarrow w$, creating two new edges $v \leftrightarrow u$ and $w \leftrightarrow u$. Knowing the quartet score of T, it is far from clear how to compute the score of the tree after the SPR move. Naively, the B13 algorithm can recompute the score in $O(n \log^2(n)k)$ after each move, leading to $O(n^2 \log^2(n)k)$ to find the optimal SPR move for each pruned edge $u \leftrightarrow u'$ and $O(n^3 \log^2(n)k)$ to test all possible SPR moves over a tree. This process of finding optimal SPR moves has to happen numerous times, rendering such an algorithm impractical. Recognizing this difficulty, all existing median quartet tree heuristics use alternatives such as ASTRAL's dynamic programming [21] or graph-based techniques used in weighted quartet max-cut [2].

Recent advances have opened up the way for a hill-climbing quartet optimization approach. Mai and Mirarab [17] showed how to prune and regraft single-taxon subtrees to their *optimal* place in $O(n \log^2(n)k)$ time. As we see, it is trivial to use that algorithm to prune and regraft

a subtree of size m in $O((n-m)m\log(n-m)\log(n)k) = O(n^2\log^2(n)k)$ time. In this paper, we show a key result: a subtree of size m can be pruned and regrafted onto the query tree with $O((n-m)\log(n-m)\log(n)k)$ time complexity, which is nearly independent of the size of the clade m (and is $O(n\log^2(n)k)$ in the worst case). This result allows us to complete a full round of optimal moves (i.e., testing all subtrees to find their optimal placement) in $O(n^2\log^2(n)k)$ time. This improved running time starts to make a hill-climbing strategy viable. After describing the algorithm, which we have implemented and made available (github.com/shayesteh99/Quartet_SPR), we present exploratory experiments showing that using SPR moves, the quartet score obtained by existing tools (ASTER and ASTRAL-III) can be further improved. These results provide us with all the necessary elements for developing a full-fledged hill-climbing quartet score optimizer, a task that future work can easily tackle.

2 Materials and Methods

2.1 Problem Definition and Notations

Let $T = (E, V)$ be a tree rooted at r_T. All leaves of T are labeled and denoted by $L(T)$. We use $v = p(u)$ to denote the parent of u and use $C(u)$ to refer to the subtree that includes vertex u after removing the edge (v, u) from the tree T. The *placement* of a subtree $C(w)$ on the edge $e = (v, u)$ of T is denoted by $\mathcal{P}_T(C(w), u)$ and is obtained by adding a degree-2 node p_u to $e = (v, u)$ to obtain (v, p_u) and (p_u, u), and connecting the subtree $C(w)$ to p_u by adding the edge (p_u, w). A *rooting* of T on $e = (v, u)$ is denoted by T_u and is obtained by dividing the edge $e = (v, u)$ into two edges (r, v) and (r, u) and reversing the direction of all edges in the path from v to r_T. Note T can be multifurcating and let d be the maximum degree among all nodes of T.

A *triplet* is a tree induced from any arbitrary set of three leaves in $L(T)$, and the least common ancestor of these three leaves is called its *anchor*. Similarly, a *rooted quartet* is a tree induced from any arbitrary set of four leaves in $L(T)$, and the *anchor* is the least common ancestor (LCA) of the four leaves. An *unrooted quartet* is the unrooted tree induced by the four leaves (unless otherwise specified, we use "quartet" to refer to an unrooted tree). A triplet on the set of leaves x, y, z is called a *matching* or *shared* triplet in trees T_1 and T_2, if $x, y, z \in L(T_1) \cap L(T_2)$ and it has the same topology in both trees. The triplet score of the trees T_1 and T_2 is defined as the number of matching triplets of T_1 and T_2 and is denoted by $S_T(T_1, T_2)$. Similarly, a quartet on the set of leaves w, x, y, z is called a matching or shared quartet in T_1 and T_2, if $w, x, y, z \in L(T_1) \cap L(T_2)$ and it has the same *unrooted* topology in both trees. The quartet score of the trees T_1 and T_2 is defined as the number of matching quartets between T_1 and T_2 and is denoted by $S_Q(T_1, T_2)$.

We define the Quartet Subtree Pruning and Regrafting (Q-SPR) problem as follows:

▶ **Definition 1** (Q-SPR Problem). *Given a rooted query tree $T' = (E, V)$, a set of arbitrarily rooted reference trees $\mathcal{R} = \{R_1, R_2, ..., R_k\}$, and $u \in V$, find*

$$\arg\max_{w} \sum_{i=1}^{k} S_Q(\mathcal{P}_T(C(u), w), R_i)$$

where T is the tree induced on $L(T') \setminus L(C(u))$.

We let $n = |T'|$ and $m = |C(u)|$, and hence $|T| = n - m$. We shorten the query subtree $C(u)$ to C when clear by context. Note that the solution to Q-SPR is the optimal position for regrafting subtree C after pruning it from the tree T', hence the name (see Figure 1 for a demonstration of the Q-SPR problem).

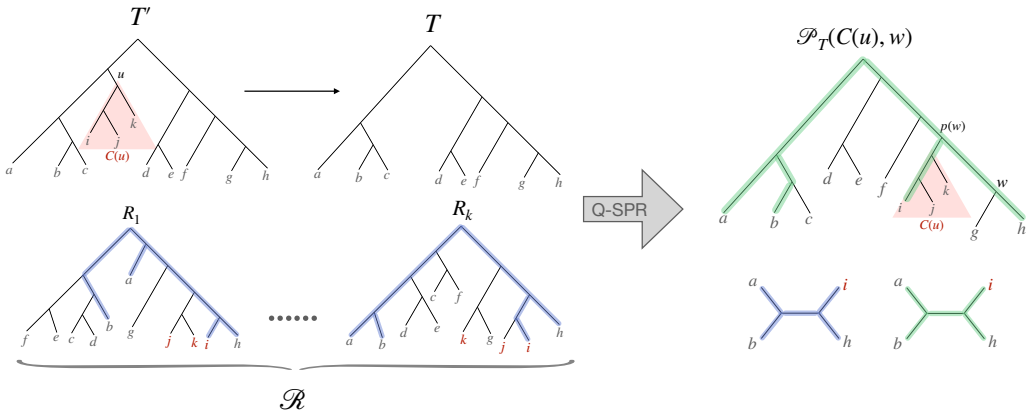


Figure 1 An overview of the Q-SPR problem. Given the query tree T', a set of reference trees $\mathcal{R}$, and a node u, the Q-SPR problem aims to find the optimal placement of $C(u)$ on T by maximizing the number of shared *unrooted* quartets between $\mathcal{P}_T(C(u), w)$ and the reference trees. An example of such quartets is highlighted in both the reference trees and the final placement.

The Q-SPR problem is an extension of the Maximum-matching Quartet Placement (MQP) problem, which is the special case of $m = 1$ and is defined as follows: Given an unrooted tree T, a single taxon q, and a set of unrooted reference trees $\mathcal{R}$, find the placement of taxon q on the tree T, denoted by T_q that maximizes $\sum_{i=1}^{k} S_Q(T_q, R_i)$. MQP is equivalent to the Maximum Triplet Rooting (MTR) problem, defined as follows: Given an unrooted tree T and a set of rooted reference trees $\mathcal{R}$, find a rooting of T on $u \in V(T)$ denoted by T_u that maximizes $\sum_{i=1}^{k} S_T(T_u, R_i)$. To see the equivalence of MQP and MTP [17], consider rooting the reference tree(s) on the query taxon q and then solving the MTR problem. The best placement of q would be at the root of the optimal rooting of the query tree T with respect to the reference tree(s).

2.2 Related work

While the Q-SPR problem is not studied in the past to our knowledge, MQP and MTR have been studied [25]. Most recently, tripVote [17] solved MTR and MQP by extending the B13 method for computing triplet scores. For a single reference tree R, tripVote computes the triplet score between the alternative rooting T_u of T and the rooted reference tree R using:

$$S_T(T_u, R) = S_T(T_{p(u)}, R) - \tau^i_{p(u)} - \tau^r_{p(u)} + \tau^o_u + \tau^r_u \tag{1}$$

where $\tau^i_{p(u)}$ is the number of shared triplets between T and R that are anchored at $p(u)$ in T, $\tau^r_{p(u)}$ is the number of shared triplets between $T_{p(u)}$ and R that are anchored at the root of $T_{p(u)}$, τ^o_u is the number of shared triplets between T_u and R that are anchored at $p(u)$ in T_u, and τ^r_u is the number of shared triplets between T_u and R that are anchored at the root of T_u. Clearly, given the three values τ^i_u, τ^o_u, and τ^r_u for every node $u \in V(T)$, the triplet score of rooting at every node can be computed in $O(1)$ using a pre-order traversal of T.

To compute these values efficiently, tripVote uses the coloring scheme of B13 for the leaf set in T and R. Tree T is traversed in a preorder manner and at each node, we recolor the leaves. Each degree d node u of T with children $c(u) = v_1, \ldots, v_{d-1}$ and parent $v_0 = p(u)$ creates d subtrees: $C(v_1), \ldots, C(v_{d-1})$, and $T - C(u)$. After recoloring based on u, each leaf in the $C(v_i)$ subtree would be assigned the color $i \in [1, d-1]$, and all the leaves in the $T \setminus C(u)$ subtree would have the color 0. The leaves of R are also assigned the same colors as T.

This coloring method enables tripVote to compute τ_u^i, τ_u^o, and τ_u^r in a top-down traversal of the nodes in T, where for each node the colors of the leaves in both trees are updated accordingly. After recoloring τ_u^i, τ_u^o, τ_u^r values can then be computed using predefined counters in the reference tree R in constant time. Colors are updated one leaf at a time by traversing R from that leaf to the root. Thus, each leaf recoloring and updating counters, if done naively, would require $O(H(R))$ operations, where $H(R)$ is the height of R. To prevent a quasi-quadratic total complexity, tripVote represents the reference trees using the HDT data structure used by B13. An HDT (Hierarchical Decomposition Tree) is a balanced binary tree data structure constructed from the nodes in the reference tree R within a time complexity of $O(n)$, where n represents the size of the reference tree. Each node in the HDT, also referred to as a component, consists of several nodes in the reference tree in a way that ensures local balance in the HDT, meaning that each component X in the HDT with m leaves has $O(\log(m))$ height. Thus, updating the HDT counters for each leaf recoloring will have $O(\log(n))$ complexity (see Table S1 for a comprehensive list of the components of the HDT).

tripVote uses two main counters in the HDT that are defined as follows (ρ^X was similar to B13 counters while π_j^X was new):

- ρ^X: The number of triplets that belong to the component X of the HDT and match the triplets anchored at the root of the alternative rooting of the query tree T_u.
- π_j^X: The number of triplets that belong to the component X of the HDT and match the triplets anchored at u in the alternative rooting of the query tree T_{v_j} where v_j corresponds to one of the child nodes of u ($j \in [1, d]$) or the parent node $p(u)$ ($j = 0$) in the query tree.

It immediately follows that HDT counters ρ^X and π_j^X are equivalent to the query tree parameters τ_u^i, τ_u^o, and τ_u^r at the root ($\mathfrak{R}$) of the HDT:

$$\tau_u^i = \pi_0^{\mathfrak{R}} \qquad \tau_u^r = \rho^{\mathfrak{R}} \qquad \tau_{v_j}^o = \pi_j^{\mathfrak{R}} \ .$$

tripVote, similar to B13, keeps $O(d^2)$ auxiliary counters for each HDT component to compute ρ^X and π_j^X efficiently using recursive functions of the colors of the leaves $j \in [0, d]$. Thus, for updating the color of each leaf, only the counters of its ancestors need to get updated, which can be done in $O(d^2 \log(n))$ using a bottom-up traversal of the HDT. Using a smaller-half trick, similar to B13, tripVote ensures $O(n \log(n))$ leaf recoloring in total. As a result, the total complexity of the algorithm is $O(d^2 n \log^2(n))$ for one reference tree and $O(kd^2 n \log^2(n))$ for k reference trees.

The tripVote algorithm can be used to solve the Q-SPR problem. It is easy to see that the optimal solution to Q-SPR can be obtained by placing each of the query taxa on the tree independently, noting the change in quartet score for each query for each branch, and choosing the branch that optimizes the total change at the end. This algorithm requires $O(kd^2(n-m)\log(n-m)\log(n)m)$ time, which is $O(kd^2n^2\log^2(n))$ for $m = \Theta(n)$ and becomes impractical for large trees.

2.3 Quartet SPR Placement Algorithm

We now propose an algorithm based on tripVote to solve the Q-SPR problem in a quasi-linear time. TripVote differentiates between the query taxon and the other taxa in the reference tree(s) by rooting the reference tree on the query taxon. However, since the Q-SPR problem has multiple query taxa that can be scattered in the reference tree(s), this approach is not viable. Therefore, we devise a quartet counting method that does not rely on a specific rooting of the reference tree(s). Instead of counting shared triplets among the query tree and the rooted version of the reference tree, we directly count the number of shared quartets

between the query tree and an arbitrary rooting of the reference tree. This method of counting presents new challenges as we have to keep track of the positions of the query taxa in the reference tree(s), and also introduce new counters for counting quartets. Below, we describe solving Q-SPR for one reference tree R; extending the approach to multiple reference trees follows trivially.

2.3.1 Counting Rooted Quartets

We select a new color, -1, for coloring the leaves of the query subtree C to be able to store their information in the counters. These leaves colored -1 are going to be present only in the reference tree, and unlike other leaves, their color is fixed throughout the algorithm.

To find the number of shared quartets between any potential placement of the subtree C on the query tree T and the reference tree R, we only need to consider the quartets that have a single taxon from $L(C)$ and three taxa from $L(T)$. This is true because the topology for quartets with zero or more than one taxon from $L(C)$ does not depend on the placement of C. We refer to the relevant quartets with one taxon from $L(C)$ as *solo* quartets. For each solo quartet, removing the sole taxon from C creates a triplet, which we will refer to as its *associated* triplet.

Since the reference tree is traversed as a rooted tree, to count the number of shared solo quartets between the query tree and the reference tree, we need to consider the rooted version of the quartets. We define the anchor of a rooted quartet as the least common ancestor of all the four taxa in the quartet; we count each quartet when we arrive at its anchor. We adopt the recursion (1) from tripVote to computing the quartet score of a node $u \in V(T)$:

$$S_Q(\mathcal{P}_T(C,u),R) = S_Q(\mathcal{P}_T(C,p(u)),R) - \varphi^i_{p(u)} - \varphi^r_{p(u)} + \varphi^o_u + \varphi^r_u \tag{2}$$

where φ^i_u, φ^r_u, and φ^o_u are defined as follows:

- φ^i_u: The number of shared solo quartets between the placement $\mathcal{P}_T(C,u)$ and R, where the associated triplet is anchored at u in T.
- φ^r_u: The number of shared solo quartets between the placement $\mathcal{P}_T(C,u)$ and R, where the associated triplet is anchored at the root r_{T_u} in the alternative rooting T_u.
- φ^o_u: The number of shared solo quartets between the placement $\mathcal{P}_T(C,u)$ and R, where the associated triplet is anchored at $p(u)$ in the alternative rooting of T_u.

To compute these variables, we first need a definition:

▶ **Definition 2.** *A quartet q is said to belong to a HDT component X, if and only if all four leaves of q belong to X.*

We now update the definitions of the main counters ρ^X and π^X_j associated with the HDT:

- ρ^X: The number of solo quartets that belong to the component X of the HDT and match the solo quartets in the placement $\mathcal{P}_T(C,u)$ where the associated triplet is anchored at the root of the alternative rooting T_u.
- π^X_j: The number of solo quartets that belong to the component X of the HDT and match the solo quartets in the placement $\mathcal{P}_T(C,u)$ where the associated triplet is anchored at u in the alternative rooting of the query tree T_{v_j} where v_j corresponds to one of the child nodes of u ($j \in [1,d]$) or the parent node $p(u)$ ($j = 0$) in the query tree.

For a particular unrooted solo quartet q in the query tree T, we consider every possible rooted topology of q in the reference tree R in order to compute ρ^X and π^X_j for every component X in the HDT. Each rooted quartet is counted exactly once at the anchor of the quartet. Same as in tripVote, the counters ρ^X and π^X_j are associated with the query tree parameters φ^i_u, φ^r_u, and φ^o_u at the HDT root $\mathfrak{R}$:

$$\varphi_u^i = \pi_0^{\mathfrak{R}} \qquad \varphi_u^r = \rho^{\mathfrak{R}} \qquad \varphi_{v_j}^o = \pi_j^{\mathfrak{R}} \ .$$

The ρ^X and π_j^X counters can be computed recursively in a postorder traversal on the HDT components, in an efficient manner:

▶ **Lemma 3.** *The ρ^X and π_j^X counters of the HDT component X can be updated in $O(d^2)$ time assuming the counters for children of X are already calculated.*

Proof. The recursion equation for each component depends on the type of the component (see Table S1). For the I and L types, ρ^X and π_j^X are set to zero as they correspond to a single node in the reference tree R and do not contain any quartets. For a component X with two child components, a quartet belongs to X if it belongs to one of the child components of X or it has at least one leaf from each child component of X. Thus, for any component X of type $IG \to C$, $CC \to C$, or $GG \to G$, with child components X_1 and X_2, we define ρ^X and π_j^X as follows:

$$\begin{aligned} \rho^X &= \rho^{X_1} + \rho^{X_2} + \rho^X_{comb} \\ \pi_j^X &= \pi_j^{X_1} + \pi_j^{X_2} + \pi^X_{comb_j} \end{aligned}$$

where ρ^X_{comb} is the number of quartets that match the criteria for ρ^X and have at least one leaf from each of the child components of X, and $\pi^X_{comb_j}$ is the number of quartets that match the criteria for π_j^X and have at least one leaf from each of the child components of X. Thus, ρ^X_{comb} and $\pi^X_{comb_j}$ of a $IG \to C$ component is set to zero, as an I component in the HDT corresponds to a single internal node in the reference tree and does not contain any leaves. For any other HDT component X, ρ^X_{comb} and $\pi^X_{comb_j}$ are computed with the help of a set of auxiliary HDT counters defined by B13. The rooted quartets counted in ρ^X_{comb} must have either $((i,j),(-1,0))$ or $((0,0),(-1,i))$ unrooted topologies, where $i,j \in [1,d]$. The rooted quartets counted in $\pi^X_{comb_j}$ must have the unrooted topology $((i,i),(-1,k))$ where $i,k \in [0,d]$ and $i,k \neq j$. We provide the full set of equations for the rooted quartets with the topologies $((i,j),(-1,0))$ and $((0,0),(-1,i))$ for ρ^X_{comb} , and $((i,i),(-1,k))$ for $\pi^X_{comb_j}$ in Tables S2, S3, and S4, respectively, noting that iterating through all 62 cases would be cumbersome. Figure 2A demonstrates an example for how ρ^X is computed using the HDT counters and also how φ_u^r is computed using ρ^X.

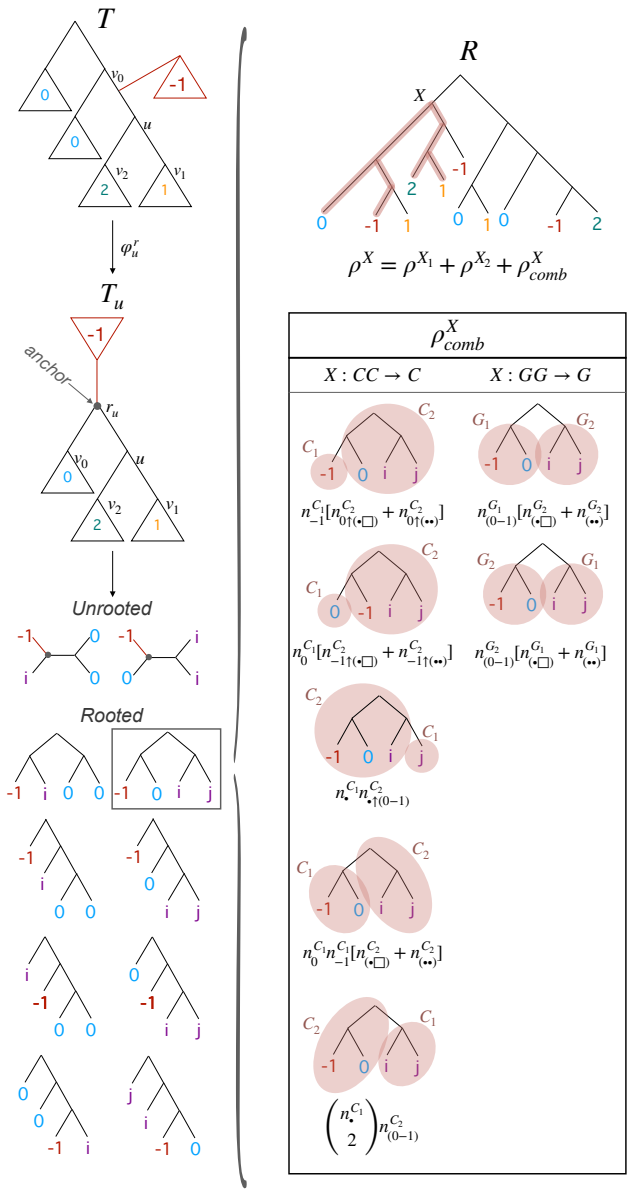
Because we only count resolved quartets and all quartets counted by our algorithm have exactly a single -1, we have $O(d^2)$ counters (i.e., of the form $(0,-1,i,j)$ for $i,j \in [1,d]$) per HDT component. From Tables S2–S4, it can be easily checked that the amortized cost of updating all the counters is constant per counter (most counters are constant while a constant number of counters need $O(d^2)$ time). ◀

To state our final results, we need one more result:

▶ **Lemma 4.** *In a top-down traversal of the query tree, a total of $(n-m)\log(n-m)$ leaf coloring steps are needed.*

Proof. The proof follows the B13 construction. There are $O(n-m)$ nodes in the tree. The smaller-half trick of B13 ensures that on each node, we avoid recoloring leaves in its largest child. This is because when we arrive at a node, the larger child already has the right color and does not need to be updated. Thus, over the entire tree, we need at most $O((n-m)\left(\frac{1}{2}+\frac{1}{2}+\ldots\right)) = O((n-m)\log(n-m))$ leaf recoloring steps. ◀

A) B)



function Q-SPR($T = (V, E), R$)
  $\mathcal{R} \leftarrow$ build-HDT(R)
  $T \leftarrow$ heaviest-child-first(T)
  color(root of T , 1)
  for $u \in$ pre-order (T) **do**
    if u is a leaf **then**
      color(u,0) and continue
    if u is the leftmost child **then**
      color(u,1)
    $v_1, v_2, \ldots, v_{d_u-1} \leftarrow c(u)$
    $\forall 2 \leq i < d_u$: color(v_i, i)
    update-counters (Tables S2–S4)
    $\varphi_u^i \leftarrow \pi_0^{\mathfrak{R}}$ and $\varphi_u^r \leftarrow \rho^{\mathfrak{R}}$
    $\forall 1 \leq i < d_u : \varphi^o_{v_i} \leftarrow \pi_i^{\mathfrak{R}}$
    $\forall 2 \leq i < d_u$: color(v_i, 0)
  for (v, u) in pre-order (T) **do**
    $S_Q(\mathcal{P}_T(C, u), R) =$
$S_Q(\mathcal{P}_T(C, v), R) - \varphi_v^i - \varphi_v^r + \varphi_u^o + \varphi_u^r$
  $u^* = \arg\max_u S_Q(\mathcal{P}_T(C, u), R)$
  return $\mathcal{P}_T(C, u^*)$

Figure 2 A) **An example of how φ_u^r is computed for the node u in the query tree.** φ_u^r is obtained by counting the number of matching quartets resulting from taking a triplet anchored at r_u in T_u and placing one query taxon at the anchor of the triplet. To count the number of matching unrooted quartets in the reference tree R, we consider every possible rooting of the quartet. By definition, φ_u^r can be obtained by updating $\rho^{\mathfrak{R}}$, which is obtained from the demonstrated recursive equation. ρ^X is computed for each HDT component X using the counters of its child components as shown in the table. The computations of ρ^X_{comb} correspond to the first row of the Table S2. B) **Quasi-linear-time algorithm to solve the Q-SPR problem.** `color(`u`,` i`)` colors all the leaves below u with i. d_u is the of degree u.

We now can state our main results.

▶ **Theorem 5.** *The Q-SPR problem can be optimally solved using the algorithm shown in Figure 2B in* $O(kd^2(n-m)\log(n-m)\log(n))$.

Proof. We perform top-down traversal of the query tree and at each node, we recolor *some* leaves and update the HDT counters (see Figure 2B). Each recoloring of the HDT according to a new node of the query tree requires recoloring the corresponding leaf components of the HDT and updating counters of all the ancestors of those leaf components. By construction, the HDT data structure of B13 has a height of $O(\log(n))$. By Lemma 3, each such update takes $O(d^2)$; thus, the total time for recoloring one leaf is $O(d^2\log(n))$. We need a total of $O((n-m)\log(n-m))$ recoloring steps by Lemma 4. Thus, we need $O(d^2(n-m)\log(n-m)\log(n))$ operations in total for recoloring and updating counters. This produces variables $\rho^{\mathfrak{R}}$ and $\pi_j^{\mathfrak{R}}$ at the root of the HDT, which then give us φ_u^i, φ_u^r, and φ_u^o for each edge. We need to repeat this for each reference tree, resulting in $O(kd^2(n-m)\log(n-m)\log(n))$. Then, the quartet score for each node u of the query tree can be trivially computed in a second preorder traversal of the query tree using Equation (2), which takes only $O((n-m)k)$ time. Ultimately, the optimal placement of C can be obtained by finding the branch that has the maximum quartet score. ◀

2.4 Tree search using quartet SPR moves

Using our method, we can perform heuristic tree searches to find the quartet median tree with respect to the reference tree(s) R using SPR moves. At each *round*, we have a current tree T' from the previous round. We draw (without replacement) a branch (v_0, u) uniformly at random from $E(T')$, remove the subtree $C(u)$ from the query tree, and obtain the pruned tree T. The optimal placement of $C(u)$ on T denoted as v^* is obtained by solving the Q-SPR problem. If $v^* \neq u$, we place $C(u)$ on v^* to obtain the improved tree $T^* = \mathcal{P}_T(C(u), v^*)$. If not, we repeat the process until edges of $E(T')$ are exhausted. We then start over the process (i.e., a new round), using T^* as the current tree. A tree is considered final if, in a round, the algorithm performs SPR on every branch of the tree and cannot find an improvement in the quartet score. Since the query tree T' has $2n-1$ branches, and we may have to find the optimal placement for all subtrees, the total running time of performing one round of SPR on every subtree of T' has $O(kd^2n^2\log^2(n))$ complexity. For the starting tree, any tree is valid. In particular, one can be made by inserting taxa into an empty tree with an arbitrary order using the Q-SPR problem. This simple algorithm can be further optimized in several ways, which we leave for discussions and future work.

2.5 Experimental setup

While we leave the development of a fully-featured software for quartet median tree search to future work, here, we present experiments that E1) demonstrate the running time of the tool and test its accurate implementation, and E2) show intriguing results in terms of the effectiveness of SPR moves and the optimality of existing tools. These experiments are based on an implementation of our method, called Q-SPR, publicly available at github.com/shayesteh99/Quartet_SPR. Our code builds on tripVote, which itself builds on the tqDist code [28]. TqDist is a library that calculates the triplet and quartet scores between two trees, utilizing the B13 algorithm.

2.5.1 E1: Running time comparison

We modified tripVote to find the optimal placement of a clade C on a query tree T with respect to the reference tree(s) R. To do so, we place each query taxon $t_i \in L(C)$ independently and obtain the quartet score $S_Q(\mathcal{P}_T(t_i, v), R)$ for every node $v \in V(T)$. To find the total quartet score for the placement of C on v, we aggregate the quartet score obtained for each taxon t_i as follows:

$$S_Q(\mathcal{P}_T(C, v), R) = \mathcal{C} + \sum_{t_i \in L(C)} S_T(T_v, R_{t_i})$$

where $\mathcal{C}$ is a constant (quartets with zero or more than one leaf from C), which we ignore. We can then simply pick the query with optimal placement. The running time of this algorithm is expected to grow linearly with $m = |L(C)|$.

For this experiment, we used an existing 10000-taxon simulated dataset [12] with 10 replicates with gene trees disagreeing with the species tree due to both ILS and Horizontal Gene Transfer (HGT), as simulated by SimPhy [18]. We used the true species tree as the query tree, and the estimated gene trees available from the original publication (estimated using FastTree-II) [24] as the reference set. We randomly selected $\{50, 100, 200, 500, 1000, 10000\}$ taxa for each replicate and pruned the input trees (the query tree and the reference trees) to only contain the selected taxa. We also subsampled the gene trees randomly to obtain $k = 100$ trees per replicate. For each of the replicates, we performed one round of SPR on *every* subtree of the query tree and measured the time each method took to find the optimal placement for each subtree. However, for trees of size 1000 and 10000, we restricted these analyses to subtrees of size at most 70 due to increased running time. In addition to the running times, we computed the quartet score $S_Q(\mathcal{P}_T(C, v), R)$ for every node $v \in V(T)$ and every subtree C, and compared the scores of Q-SPR to the scores of modified tripVote to ensure the correct implementation of Q-SPR.

2.5.2 E2: Improving ASTER trees

In this experiment, we tested the optimality of two leading median tree search heuristics currently in use. We tested ASTRAL-III and ASTER; the latter is a newer algorithm and is shown to be better than ASTRAL-III in the face of the missing data [34]. We used the species trees inferred from the gene trees using these methods as the starting tree to our Q-SPR hill-climbing search. If the initial tree is an optimal tree (even a locally optimal tree), then, no SPR move should improve the quartet score. We asked if this is true, and if not, how much improvement in quartet score, species tree topology, species tree branch length, and branch support can be obtained.

The experiment used an existing Simphy-simulated ILS-only 200-taxon dataset [22] that was modified by Zhang *et al.* [34] to randomly remove $\approx 5\%$ of taxa from each estimated gene tree. For the purpose of this experiment, we focused on four different model conditions in this dataset (tree height $\in \{5 \times 10^5, 2 \times 10^6\}$ corresponding to high and medium ILS, and $k \in \{50, 200\}$). Each model condition provides 50 replicates with a speciation rate of 10^{-6} and 50 replicates with a speciation rate of 10^{-7}, making a total of 100 replicates per condition. The model condition with high ILS and $k = 50$ is regarded as the most challenging model condition where Zhang *et al.* [34] observed the most pronounced differences between optimization methods (ASTRAL-III vs. ASTER) [34]. For each model condition, we ran Q-SPR on both the ASTRAL-III and the ASTER trees. To test the optimality of the starting trees, we report the number of rounds of improvement and the quartet score

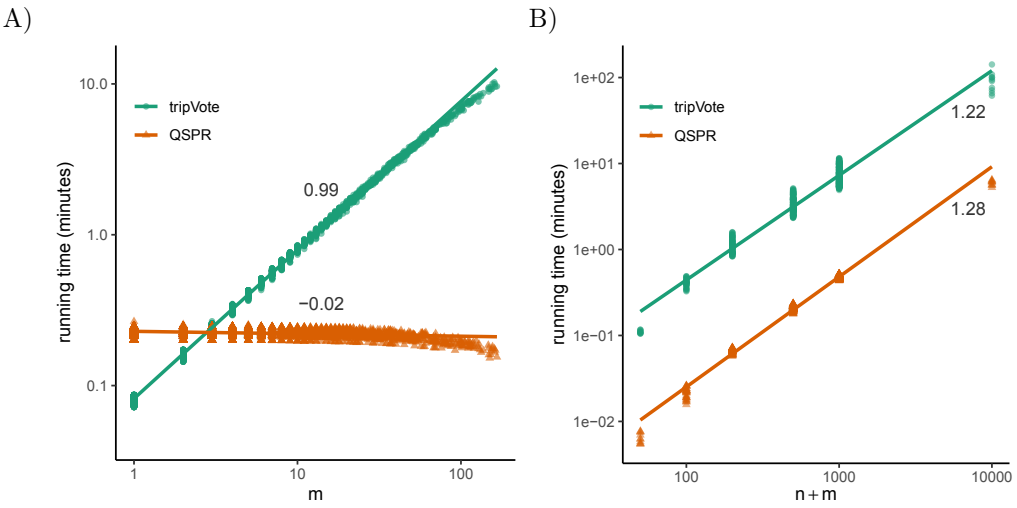


Figure 3 Results of E1. The running time is shown versus the increasing size of (A) the query subtree (m) or (B) the query tree (n). In (A), we fix $n = 500$ and in (B), we ensure $30 < m < 70$. Both axes are in log scale, making the line slope (annotated) an empirical estimate of the asymptotic running time degree. We refer to our adoption of tripVote for Q-SPR as tripVote.

improvement at the end of all rounds. We also report the total running time of Q-SPR for each experiment (not including starting tree inference). To measure accuracy, the Q-SPR and the starting trees were both compared against the known true species tree based on both normalized quartet distance and the normalized Robinson–Foulds (nRF) [27] metric. In addition, the local posterior probability (PP) [30] and the coalescent unit length for each internal branch was computed using ASTRAL-III. This method sets branch lengths to zero when the frequency of the species tree quartet topology is less than $1/3$ among gene tees (unexpected under MSC); note that localPP is $< 1/3$ under those conditions as well. We evaluated support and length for internal branches that matched or differed between the starting and final Q-SPR trees, with a particular focus on the unexpected cases with zero branch length or localPP $< 1/3$.

3 Results

3.1 Running time scaling

Comparing tripVote and Q-SPR, we ensured the two software produced identical results (quartet support of the placement clade) in all cases. The gene trees here are estimated and include polytomies. Our extensive empirical results provide validation that the complex set of recursively defined counters used by Q-SPR (Tables S2–S4) are correct.

Testing the impact of n for subclades of limited range ($30 < m < 70$) on the running time, we observe similar asymptotic behavior between tripVote and Q-SPR (Figure 3 B). The theoretical running time of both methods increases quasi-linearly with respect to the size of the query tree n, which matches our empirical estimate of the log-log slope to be only slightly above 1.0. Regardless of n, for the selected range of m, Q-SPR is always faster than tripVote by a factor of 10 to 31 (mean: 16), which is $\approx m/2.8$; this empirical observation matches the theory that Q-SPR is faster by a factor of $\Theta(m)$.

When we fix $n = 500$ and vary m, we observe the asymptotic advantage of Q-SPR over tripVote (Figure 3A). The running time of Q-SPR is nearly independent of the size of the query subtree (m), as expected, whereas the running time of tripVote increases linearly with m. Regardless of m, the running time of Q-SPR ranges between 9 and 16 seconds (mean: 13.5). In these analyses, Q-SPR is faster than tripVote in most conditions. The main exception is that tripVote is faster for subtrees of size one or two ($m \leq 2$); the likely reason is that tripVote has fewer counters to maintain than Q-SPR. At the other end, for $m > 100$ taxa, Q-SPR is on average 73 times faster than tripVote. The running time of Q-SPR has a slight downward trend. The reason is that increasing m decreases the size of the tree induced by removing the query subtree from the query tree $n-m$. In the SPR setting where n is fixed, as the subtree size increases, the size of the backbone tree $(n - m)$ decreases, which makes Q-SPR faster. The total running time of Q-SPR in this case is $O(k(n-m)\log(n-m)\log(n))$, while the overall running time for tripVote is $O(km(n - m)\log(n - m)\log(n))$.

3.2 Improving ASTER and ASTRAL trees

The quartet score of the starting tree with respect to the gene trees is guaranteed to remain fixed or improve after Q-SPR moves. We saw improvements in 310 out of 800 cases we tested, but there were wide differences depending on the model condition (Figure 4A). For example, 93% and 61% of runs starting from ASTRAL-III improved in quartet score after Q-SPR moves for 50 and 200 genes, respectively. In the other extreme, only 5% of low ILS cases starting from ASTER saw any improvements. The normalized quartet score improvements were as large as 6% in extreme cases (Figure S2) but were below 0.5% in 99% of cases. The mean improved score was 0.11% in the most difficult condition (50 genes, high ILS, starting from ASTRAL-III) and only 0.0001% in the easiest condition (200 genes, low ILS, starting from ASTER). Overall, 13.7% of branches across all trees changed.

The improvements in the optimization score tend to but do not always lead to improved species trees (Figure 4B,C). Overall, only 107 or 157 out of the 310 cases with improved quartet scores had improved accuracy compared to the true species trees in terms of nRF or quartet distance, respectively; in contrast, 137 or 149 cases had lowered accuracy (the rest had equal distance). However, while improving quartet score *can* reduce the accuracy, it never reduces it dramatically (e.g., never more than 7.5% nRF). In contrast, in *some cases*, accuracy improves dramatically after optimization (e.g., in seven cases delta nRF is 10% or more). Reassuringly, there does seem to be a positive correlation between improved score and improved accuracy and many cases with the largest quartet score improvement did have high improvements in accuracy (Figs. 4B and S2). Overall, Q-SPR optimization only slightly improved the nRF and the quartet distance between the estimated and true species trees (e.g., nRF improved by 0.1%).

Examining individual rounds of Q-SPR showed interesting patterns (Figure 4E). First, there is much variation from one replicate to another and from one condition to another, with the number of rounds ranging from 1 to 242 (mean 4.8). The mean number of rounds is as low as 1.5 for easy conditions and as high as 14 for difficult ones. Correspondingly, there is a high level of variation in terms of running times, requiring between 11 and 208 minutes on this 200-taxon dataset (Figure S1). Interestingly, some of the runs with many rounds of quartet score improvement had little or no impact on accuracy. Overall, it appears that improving the quartet score when it is already high leads to very little improvement in accuracy whereas improving the quartet score for challenging datasets with low accuracy can dramatically improve accuracy (Figure 4E).

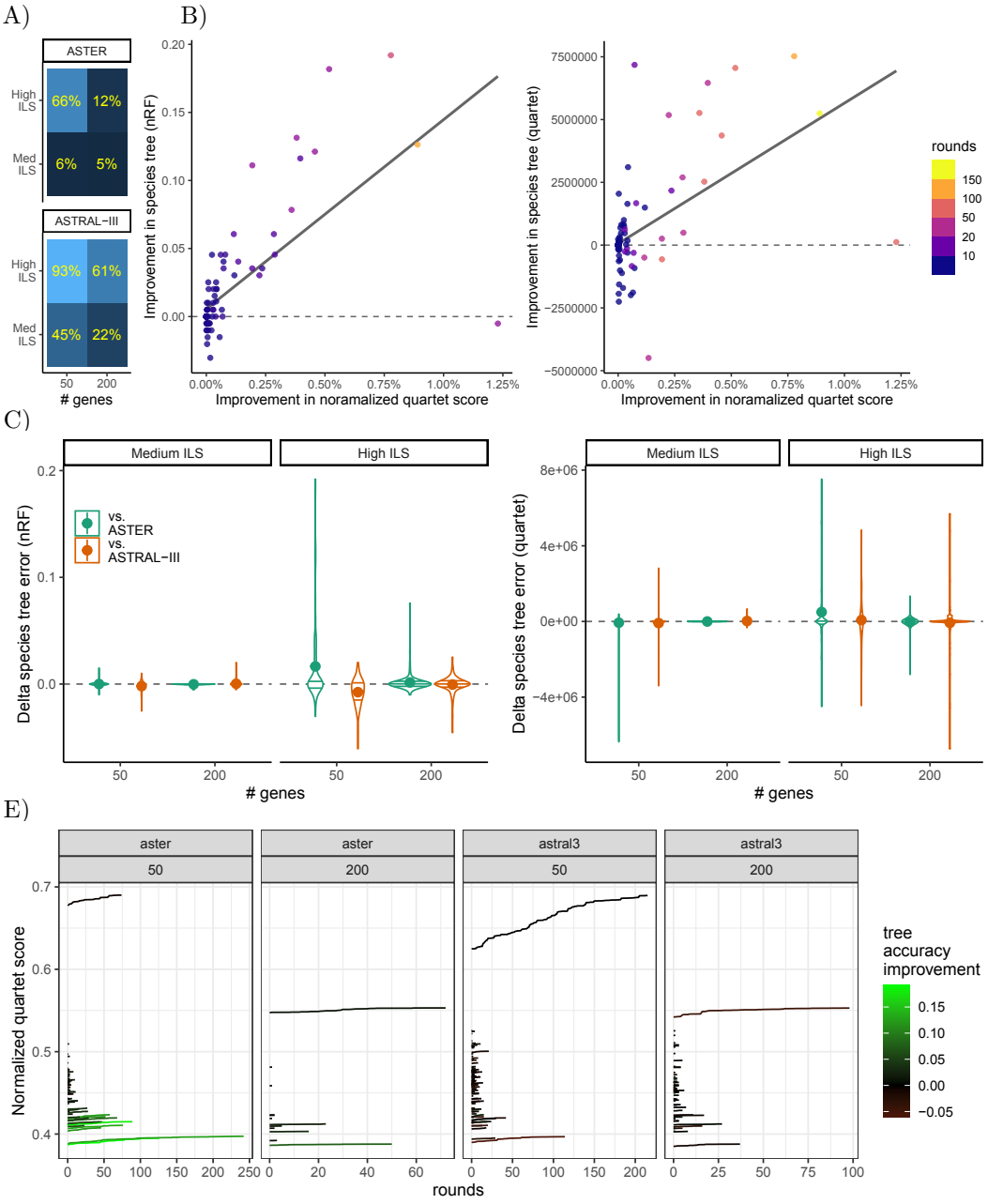


Figure 4 Results for E2. A) Percentage of the replicates with improved quartet score, starting from either ASTER or ASTRAL-III trees. B) Improvement in the quartet score of the Q-SPR algorithm above the ASTER tree (under the High ILS model condition and $k = 50$) with respect to the gene trees versus the improvement in the normalized RF or the quartet distance between the ASTER tree and the true species tree. The number of SPR rounds performed for each replicate is shown in colors. Restricted to high ILS, 50 genes, ASTER; see Figure S2 for all model conditions. C) The difference between the normalized RF and the quartet distance of the Q-SPR tree and the starting tree with respect to the true species tree for all tested conditions. Positive (negative) values indicate an improvement (reduction) in the accuracy of the starting tree. E) The normalized quartet score between the Q-SPR tree at the end of each SPR round and the gene trees under the High ILS (ILS rate $= 500k$) model conditions. The final improvement of the Q-SPR tree with respect to the true species tree is shown in colors.

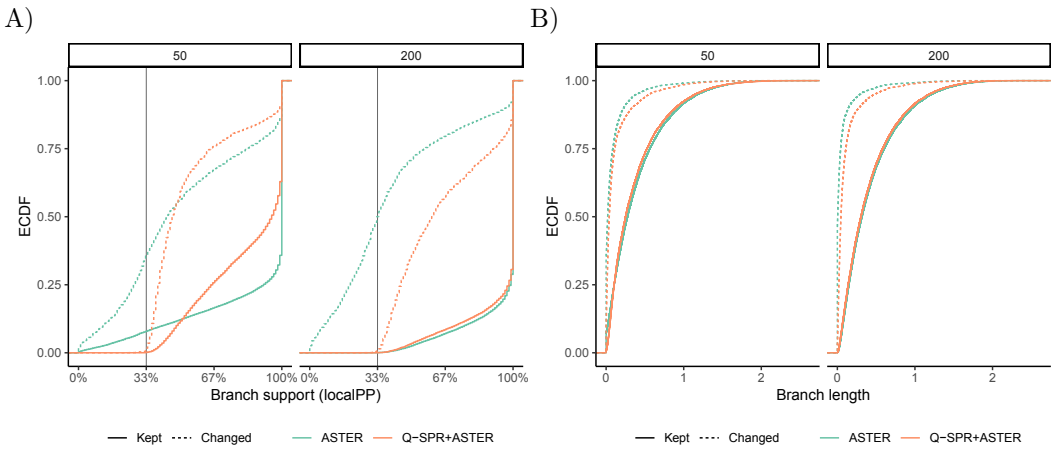


Figure 5 The results of the analysis on the branch lengths and branch supports of the improved tree using Q-SPR and the starting ASTER tree. A) The branch supports for the improved tree (ASTER + Q-SPR) and the starting tree (ASTER). The non-matching branches refer to the branches that are either removed from the ASTER tree or added to the improved tree. The branch support threshold 1/3 is shown by the dotted horizontal line. B) The empirical cumulative density function (ECDF) of the branch lengths of the improved and the starting tree for both the non-matching and matching branches. Shown is only ASTER starting tree with high ILS; see Figure S3 for full results.

Comparing the branch supports of ASTER and Q-SPR trees shows that fewer branches have unexpectedly low support (Figure 5A). Around 10% of branches before Q-SPR had support below $\frac{1}{3}$ and branch length 0. Species tree branches with these properties are problematic as they would have to have less frequency than alternative topologies in the gene trees (inconsistent with MSC). In contrast, only 0.2% of the branches after Q-SPR had support below $\frac{1}{3}$ or 0 branch length. While most of the branches changed by the Q-SPR had low support, some had high support (e.g., 16% had localPP > 0.95). Interestingly, it appears that compared to the starting trees, Q-SPR (indirectly) trades off some reduction in the number of high support branches with having fewer very low support branches.

4 Discussion

We introduce a quasi-linear algorithm for placing a query subtree on a tree. Our algorithm takes as input a set of reference trees that include all or some of the leaves in the query subtree and maximizes the total quartet score between the output tree and the reference tree(s). We demonstrated that the running time of this method improves on the state-of-the-art (tripVote) and is indeed nearly independent of the size of the query subtree. Our algorithm, called Q-SPR, can perform quartet-based subtree prune and regraft (SPR) on a given tree with respect to the reference tree(s). This efficient algorithm for SPR moves provided us with a way to enhance species tree inference starting from the output of methods such as ASTER and ASTRAL, as we demonstrated in our experiments. We showed that for both methods, the quartet score could be often improved. These improvements tended to eliminate super low support branches with zero estimated length (i.e., those that do not match MSC expectations). As such, they would make the resulting trees, and in particular support values on those trees, more interpretable. However, the impact on the overall topological accuracy was small to moderate, especially given enough genes. While improving the quartet score

generally leads to a more precise median tree for an infinite number of gene trees, this is often not the case when dealing with a finite set of gene trees. Thus, imperfect correlations between the quartet score and accuracy mean that further improving the quartet score beyond what heuristics such as ASTER and ASTRAL-III achieve has limited practical value. However, our results indicate that perhaps a faster and better heuristic method can be designed relying solely on hill-climbing using SPRs. This will be the subject of future work.

Our approach can be expanded in many ways. Having an efficient SPR move available, we can now explore the best possible way to apply SPR moves on a starting tree to obtain efficient hill climbing. Our current approach, testing all possible pruning locations at random with no attempt at guessing the best move, can likely be improved. More ambitiously, our theory can be extended to ask if some of the calculations performed for the optimal SPR move of a clade can be reused for adjacent clades, hence reducing running time. Moreover, the much faster NNI moves (which are a special case of SPR) should perhaps also be tried. The choice of the starting tree will also need to be explored. We can, for example, use the widely-used step-wise addition strategy, which incidentally, is also made possible by our algorithm (and tripVote). These improvements need to all be combined with high-performance computing features such as parallelism. Thus, while all the main algorithmic ingredients are ready, a fully-featured heuristic hill-climbing implementation requires further engineering.

References

1 E S Allman, James H. Degnan, and J A Rhodes. Identifying the rooted species tree from the distribution of unrooted gene trees under the coalescent. *J. Math. Biol.*, 62:833–862, 2011.

2 Eliran Avni, Reuven Cohen, and Sagi Snir. Weighted Quartets Phylogenetics. *Systematic Biology*, 64(2):233–242, March 2015. doi:10.1093/sysbio/syu087.

3 Paul D Blischak, Jeremy M Brown, Zhen Cao, Alison Cloutier, Kerry Cobb, Alexandria A DiGiacomo, Deren AR Eaton, Scott V Edwards, Kyle A Gallivan, and Daniel J Gates. *Species Tree Inference: A Guide to Methods and Applications*. Princeton University Press, 2023.

4 Gerth Stølting Brodal, Rolf Fagerberg, Thomas Mailund, Christian N. S. Pedersen, and Andreas Sand. Efficient Algorithms for Computing the Triplet and Quartet Distance Between Trees of Arbitrary Degree. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1814–1832, Philadelphia, PA, January 2013. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973105.130.

5 David Bryant and Mike Steel. Constructing Optimal Trees from Quartets. *Journal of Algorithms*, 38(1):237–259, January 2001. doi:10.1006/jagm.2000.1133.

6 David Bryant, John Tsang, Paul E Kearney, and Ming Li. Computing the quartet distance between evolutionary trees. In *Algorithms and Computation. ISAAC 2001*, volume 9(11) of *LNCS*, pages 285–286. Citeseer, 2000.

7 Julia Chifman and Laura S Kubatko. Quartet Inference from SNP Data Under the Coalescent Model. *Bioinformatics*, 30(23):3317–3324, August 2014. Publisher: Oxford Univ Press. doi:10.1093/bioinformatics/btu530.

8 Ruth Davidson, Pranjal Vachaspati, Siavash Mirarab, and Tandy Warnow. Phylogenomic species tree estimation in the presence of incomplete lineage sorting and horizontal gene transfer. *BMC Genomics*, 16(Suppl 10):S1, 2015. doi:10.1186/1471-2164-16-S10-S1.

9 G. F. Estabrook, F. R. McMorris, and C. A. Meacham. Comparison of Undirected Phylogenetic Trees Based on Subtrees of Four Evolutionary Units. *Systematic Biology*, 34(2):193–200, June 1985. doi:10.2307/sysbio/34.2.193.

10 M. T. Hallett and Jens Lagergren. New algorithms for the duplication-loss model. In *Proceedings of the fourth annual international conference on Computational molecular biology - RECOMB '00*, pages 138–146, New York, New York, USA, 2000. ACM Press. doi:10.1145/332306.332359.

11 Max Hill, Brandon Legried, and Sebastien Roch. Species tree estimation under joint modeling of coalescence and duplication: sample complexity of quartet methods. *arXiv*, 2020. arXiv:2007.06697.

12 Yueyu Jiang, Metin Balaban, Qiyun Zhu, and Siavash Mirarab. DEPP: Deep Learning Enables Extending Species Trees using Single Genes. *Systematic Biology*, page 2021.01.22.427808, April 2022. doi:10.1093/sysbio/syac031.

13 Manuel Lafond and Celine Scornavacca. On the Weighted Quartet Consensus problem. *Theoretical Computer Science*, 769:1–17, May 2019. arXiv:1610.00505 Genre: Data Structures and Algorithms. doi:10.1016/j.tcs.2018.10.005.

14 Bret R. Larget, Satish K. Kotha, Colin N. Dewey, and Cécile Ané. BUCKy: Gene tree/species tree reconciliation with Bayesian concordance analysis. *Bioinformatics*, 26(22):2910–2911, November 2010. arXiv:0912.4472 Publisher: Department of Statistics, University of Wisconsin-Madison, WI 53706, USA. ISBN: 03036812. doi:10.1093/bioinformatics/btq539.

15 Brandon Legried, Erin K Molloy, Tandy Warnow, and Sébastien Roch. Polynomial-Time Statistical Estimation of Species Trees Under Gene Duplication and Loss. *Journal of Computational Biology*, 28(5):452–468, May 2021. doi:10.1089/cmb.2020.0424.

16 Wayne P. Maddison. Gene Trees in Species Trees. *Systematic Biology*, 46(3):523–536, September 1997. doi:10.2307/2413694.

17 Uyen Mai and Siavash Mirarab. Completing gene trees without species trees in sub-quadratic time. *Bioinformatics*, 38(6):1532–1541, March 2022. doi:10.1093/bioinformatics/btab875.

18 Diego Mallo, Leonardo De Oliveira Martins, and David Posada. SimPhy : Phylogenomic Simulation of Gene, Locus, and Species Trees. *Systematic Biology*, 65(2):334–344, March 2016. doi:10.1093/sysbio/syv082.

19 Alexey Markin and Oliver Eulenstein. Quartet-based inference is statistically consistent under the unified duplication-loss-coalescence model. *Bioinformatics*, page btab414, May 2021. doi:10.1093/bioinformatics/btab414.

20 Siavash Mirarab, Luay Nakhleh, and Tandy Warnow. Multispecies Coalescent: Theory and Applications in Phylogenetics. *Annual Review of Ecology, Evolution, and Systematics*, 52(1):247–268, November 2021. doi:10.1146/annurev-ecolsys-012121-095340.

21 Siavash Mirarab, Rezwana Reaz, Md. Shamsuzzoha Bayzid, Théo Zimmermann, M. S. Swenson, and Tandy Warnow. ASTRAL: genome-scale coalescent-based species tree estimation. *Bioinformatics*, 30(17):i541–i548, September 2014. doi:10.1093/bioinformatics/btu462.

22 Siavash Mirarab and Tandy Warnow. ASTRAL-II: Coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. *Bioinformatics*, 31(12):i44–i52, June 2015. doi:10.1093/bioinformatics/btv234.

23 P Pamilo and M Nei. Relationships between gene trees and species trees. *Molecular biology and evolution*, 5(5):568–583, 1988. ISBN: 0737-4038 (Print). URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=3193878.

24 Morgan N. Price, Paramvir S. Dehal, and Adam P. Arkin. FastTree-2 – Approximately Maximum-Likelihood Trees for Large Alignments. *PLoS ONE*, 5(3):e9490, March 2010. Publisher: Public Library of Science. doi:10.1371/journal.pone.0009490.

25 Maryam Rabiee and Siavash Mirarab. INSTRAL: Discordance-Aware Phylogenetic Placement Using Quartet Scores. *Systematic Biology*, 69(2):384–391, August 2020. doi:10.1093/sysbio/syz045.

26 Bruce Rannala and Ziheng Yang. Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci. *Genetics*, 164(4):1645–1656, 2003. Publisher: Department of Medical Genetics, University of Alberta, Edmonton, Alberta T6G 2H7, Canada.

27 DF Robinson and LR Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981. URL: http://www.sciencedirect.com/science/article/pii/0025556481900432.

28 Andreas Sand, Morten K. Holt, Jens Johansen, Gerth Stølting Brodal, Thomas Mailund, and Christian N. S. Pedersen. tqDist: a library for computing the quartet and triplet distances between binary or general trees. *Bioinformatics*, 30(14):2079–2080, July 2014. doi:10.1093/bioinformatics/btu157.

29 Erfan Sayyari and Siavash Mirarab. Anchoring quartet-based phylogenetic distances and applications to species tree reconstruction. *BMC Genomics*, 17(S10):101–113, November 2016. doi:10.1186/s12864-016-3098-z.

30 Erfan Sayyari and Siavash Mirarab. Fast Coalescent-Based Computation of Local Branch Support from Quartet Frequencies. *Molecular Biology and Evolution*, 33(7):1654–1668, July 2016. doi:10.1093/molbev/msw079.

31 Sagi Snir, Tandy Warnow, and Satish Rao. Short Quartet Puzzling: A New Quartet-Based Phylogeny Reconstruction Algorithm. *Journal of Computational Biology*, 15(1):91–103, January 2008. doi:10.1089/cmb.2007.0103.

32 Michael Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116, January 1992. doi:10.1007/BF02618470.

33 Cuong Than and Luay Nakhleh. Species Tree Inference by Minimizing Deep Coalescences. *PLoS Computational Biology*, 5(9):e1000501, September 2009. doi:10.1371/journal.pcbi.1000501.

34 Chao Zhang and Siavash Mirarab. Weighting by Gene Tree Uncertainty Improves Accuracy of Quartet-based Species Trees. *Molecular Biology and Evolution*, 39(12):msac215, October 2022. doi:10.1093/molbev/msac215.

35 Chao Zhang, Maryam Rabiee, Erfan Sayyari, and Siavash Mirarab. ASTRAL-III: polynomial time species tree reconstruction from partially resolved gene trees. *BMC Bioinformatics*, 19(S6):153, May 2018. doi:10.1186/s12859-018-2129-y.

S1 Supplementary Tables

Table S1 HDT components types. Each component in the HDT corresponds to a part of the reference tree R, with C and G being the super-types in the HDT.

Type	Subtype	Description
I		corresponds to an internal node in the reference tree R. Every I component is a leaf in the HDT
G		corresponds to a set of subtrees in the reference tree R where the roots of the subtrees are siblings. A G component can be one of two types:
	L	corresponds to a leaf node in the reference tree R. Every L component is a leaf in the HDT
	$GG \rightarrow G$	a component of the HDT that is the parent of two other G components
C		corresponds to a connected subset of nodes in the reference tree R where the root of one subset is a descendant of the root of the other subset. A C component can be one of two types:
	$IG \rightarrow C$	a component of the HDT that is the parent of an I and a G component
	$CC \rightarrow C$	a component of the HDT that is the parent of two C components

Table S2 Equations for computing the HDT counter ρ^{X}_{comb} for component types $CC \to C$ and $GG \to G$. Each row shows one rooted topology of the unrooted quartet $((i,j),(-1,0))$. ρ^{X}_{comb} for each component type is the sum over the equations of all the rooted topologies.

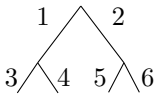
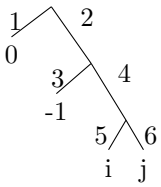
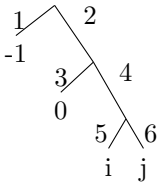
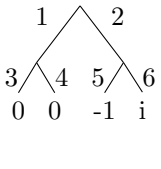
Rooted Quartet		$CC \to C$		$GG \to G$
	1 2 3 4 5, 6	$n_0^{C_1} n_{-1}^{C_1} [n_{(\bullet\square)}^{C_2} + n_{(\bullet\bullet)}^{C_2}]$ $+\binom{n_\bullet^{C_1}}{2} n_{(0-1)}^{C_2}$ $+n_0^{C_1} [n_{-1\uparrow(\bullet\square)}^{C_2} + n_{-1\uparrow(\bullet\bullet)}^{C_2}]$ $+n_{-1}^{C_1} [n_{0\uparrow(\bullet\square)}^{C_2} + n_{0\uparrow(\bullet\bullet)}^{C_2}]$ $+n_\bullet^{C_1} n_{\bullet\uparrow(0-1)}^{C_2}$	1 2	$n_{(0-1)}^{G_1} [n_{(\bullet\square)}^{G_2} + n_{(\bullet\bullet)}^{G_2}]$ $+n_{(0-1)}^{G_2} [n_{(\bullet\square)}^{G_1} + n_{(\bullet\bullet)}^{G_1}]$
	1 2 3 4 5, 6	$+n_0^{C_1} [n_{(-1(\bullet\square))}^{C_2} + n_{(-1(\bullet\bullet))}^{C_2}]$ $+[n_{[-1(\bullet\square)]}^{C_1} + n_{[-1(\bullet\bullet)]}^{C_1}] n_0^{C_2}$ $+n_{-1}^{C_1} [n_{(\bullet\square)\uparrow 0}^{C_2} + n_{(\bullet\bullet)\uparrow 0}^{C_2}]$ $+\binom{n_\bullet^{C_1}}{2} n_{-1\uparrow 0}^{C_2}$ $+n_\bullet^{C_1} n_{\bullet\uparrow -1\uparrow 0}^{C_2}$	1 2	$+n_0^{G_1} [n_{[-1(\bullet\square)]}^{G_2} + n_{[-1(\bullet\bullet)]}^{G_2}]$ $+n_0^{G_2} [n_{[-1(\bullet\square)]}^{G_1} + n_{[-1(\bullet\bullet)]}^{G_1}]$
	1 2 3 4 5, 6	$+n_{-1}^{C_1} [n_{(0(\bullet\square))}^{C_2} + n_{(0(\bullet\bullet))}^{C_2}]$ $+[n_{[0(\bullet\square)]}^{C_1} + n_{[0(\bullet\bullet)]}^{C_1}] n_{-1}^{C_2}$ $+n_0^{C_1} [n_{(\bullet\square)\uparrow -1}^{C_2} + n_{(\bullet\bullet)\uparrow -1}^{C_2}]$ $+\binom{n_\bullet^{C_1}}{2} n_{0\uparrow -1}^{C_2}$ $+n_\bullet^{C_1} n_{\bullet\uparrow 0\uparrow -1}^{C_2}$	1 2	$+n_{-1}^{G_1} [n_{[0(\bullet\square)]}^{G_2} + n_{[0(\bullet\bullet)]}^{G_2}]$ $+n_{-1}^{G_2} [n_{[0(\bullet\square)]}^{G_1} + n_{[0(\bullet\bullet)]}^{G_1}]$
	1 2 3 4 5 6	$+n_\bullet^{C_1} n_{(\bullet(0-1))}^{C_2}$ $+n_{[\bullet(0-1)]}^{C_1} n_\bullet^{C_2}$ $+n_\bullet^{C_1} n_{(0-1)\uparrow\bullet}^{C_2}$ $+n_0^{c_1} n_{-1}^{C_1} [n_{\bullet\uparrow\square}^{C_2} + n_{\bullet\uparrow\bullet}^{C_2}]$ $+n_0^{C_1} [n_{-1\uparrow\bullet\uparrow\square}^{C_2} + n_{-1\uparrow\bullet\uparrow\bullet}^{C_2}]$ $+n_{-1}^{C_1} [n_{0\uparrow\bullet\uparrow\square}^{C_2} + n_{0\uparrow\bullet\uparrow\bullet}^{C_2}]$	1 2	$+n_\bullet^{G_1} n_{[\bullet(0-1)]}^{G_2}$ $+n_\bullet^{G_2} n_{[\bullet(0-1)]}^{G_1}$

Table S3 Equations for computing the HDT counter ρ^X_{comb} for component types $CC \to C$ and $GG \to G$ (Cont.) Each row shows one rooted topology of the unrooted quartet $((0,0),(-1,i))$. ρ^X_{comb} for each component type is the sum over the equations of all the rooted topologies in Table S2 and S3.

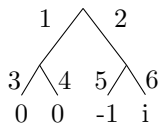
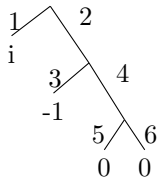
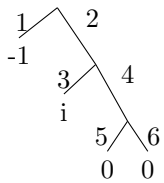
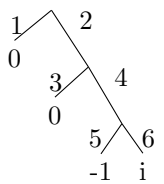
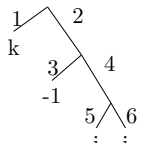
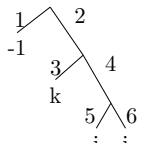
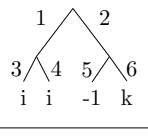
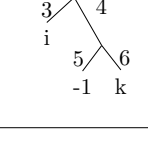
Rooted Quartet		$CC \to C$		$GG \to G$
	1 2 3, 4 5 6	$+\binom{n_0^{C_1}}{2} n^{C_2}_{(-1\bullet)}$ $+n^{C_1}_{-1} n^{C_1}_{\bullet} n^{C_2}_{(00)}$ $+n^{C_1}_{0} n^{C_2}_{0\uparrow(-1\bullet)}$ $+n^{C_1}_{-1} n^{C_2}_{\bullet\uparrow(00)}$ $+n^{C_1}_{\bullet} n^{C_2}_{-1\uparrow(00)}$	1 2	$+n^{G_1}_{(00)} n^{G_2}_{(-1\bullet)}$ $+n^{G_2}_{(00)} n^{G_1}_{(-1\bullet)}$
	1 2 3 4 5, 6	$+n^{C_1}_{\bullet} n^{C_2}_{(-1(00))}$ $+n^{C_1}_{[-1(00)]} n^{C_2}_{\bullet}$ $+n^{C_1}_{-1} n^{C_2}_{(00)\uparrow\bullet}$ $+\binom{n_0^{C_1}}{2} n^{C_2}_{-1\uparrow\bullet}$ $+n^{C_1}_{0} n^{C_2}_{0\uparrow-1\uparrow\bullet}$	1 2	$+n^{G_1}_{\bullet} n^{G_2}_{[-1(00)]}$ $+n^{G_2}_{\bullet} n^{G_1}_{[-1(00)]}$
	1 2 3 4 5, 6	$+n^{C_1}_{-1} n^{C_2}_{(\bullet(00))}$ $+n^{C_1}_{[\bullet(00)]} n^{C_2}_{-1}$ $+n^{C_1}_{\bullet} n^{C_2}_{(00)\uparrow-1}$ $+\binom{n_0^{C_1}}{2} n^{C_2}_{\bullet\uparrow-1}$ $+n^{C_1}_{0} n^{C_2}_{0\uparrow\bullet\uparrow-1}$	1 2	$+n^{G_1}_{-1} n^{G_2}_{[\bullet(00)]}$ $+n^{G_2}_{-1} n^{G_1}_{[\bullet(00)]}$
	1 2 3 4 5 6	$+n^{C_1}_{0} n^{C_2}_{(0(-1\bullet))}$ $+n^{C_1}_{[0(-1\bullet)]} n^{C_2}_{0}$ $+n^{C_1}_{0} n^{C_2}_{(\bullet-1)\uparrow0}$ $+n^{c_1}_{\bullet} n^{C_1}_{-1} n^{C_2}_{0\uparrow0}$ $+n^{C_1}_{-1} n^{C_2}_{\bullet\uparrow0\uparrow0}$ $+n^{C_1}_{\bullet} n^{C_2}_{-1\uparrow0\uparrow0}$	1 2	$+n^{G_1}_{0} n^{G_2}_{[0(-1\bullet)]}$ $+n^{G_2}_{0} n^{G_1}_{[0(-1\bullet)]}$

Table S4 Equations for computing the HDT counter $\pi^X_{comb_j}$ for component types $CC \to C$ and $GG \to G$ Each row shows one rooted topology of the unrooted quartet $((i,i),(-1,k))$. $\pi^X_{comb_j}$ for each component type is the sum over the equations of all the rooted topologies.

Rooted Quartet		$CC \to C$		$GG \to G$
	1 2 3 4 5, 6	$\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{(-1(\bullet\bullet))} - n^{C_2}_{(-1(ii))} - n^{C_2}_{(-1(jj))})$ $+\sum_{i\neq j} n^{C_1}_{[-1(ii)]}(n^{C_2}_{\bullet} - n_i^{C_2} - n_j^{C_2})$ $+n_{-1}^{C_1}(n^{C_2}_{(\bullet\bullet)\uparrow\square} - n^{C_2}_{(jj)\uparrow\bullet} - n^{C_2}_{(\bullet\bullet)\uparrow j})$ $+\sum_{i\neq j} \binom{n_i^{C_1}}{2}(n^{C_2}_{-1\uparrow\bullet} - n^{C_2}_{-1\uparrow i} - n^{C_2}_{-1\uparrow j})$ $+\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{i\uparrow -1\uparrow\bullet} - n^{C_2}_{i\uparrow -1\uparrow j})$	1 2	$\sum_{i\neq j} n^{G_2}_{[-1(ii)]}(n^{G_1}_{\bullet} - n_i^{G_1} - n_j^{G_1})$ $+\sum_{i\neq j} n^{G_1}_{[-1(ii)]}(n^{G_2}_{\bullet} - n_i^{G_2} - n_j^{G_2})$
	1 2 3 4 5, 6	$+n_{-1}^{C_1}(n^{C_2}_{(\bullet(\square\square))} - n^{C_2}_{(j(\bullet\bullet))} - n^{C_2}_{(\bullet(jj))})$ $+(n^{C_1}_{[\bullet(\square\square)]} - n^{C_1}_{[j(\bullet\bullet)]} - n^{C_1}_{[\bullet(jj)]})n_{-1}^{C_2}$ $+\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{(\bullet\bullet)\uparrow -1} - n^{C_2}_{(ii)\uparrow -1} - n^{C_2}_{(jj)\uparrow -1})$ $+\sum_{i\neq j} \binom{n_i^{C_1}}{2}(n^{C_2}_{\bullet\uparrow -1} - n^{C_2}_{i\uparrow -1} - n^{C_2}_{j\uparrow -1})$ $+\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{i\uparrow\bullet\uparrow -1} - n^{C_2}_{i\uparrow j\uparrow -1})$	1 2	$+n_{-1}^{G_1}(n^{G_2}_{[\bullet(\square\square)]} - n^{G_2}_{[j(\bullet\bullet)]} - n^{G_2}_{[\bullet(jj)]})$ $+n_{-1}^{G_2}(n^{G_1}_{[\bullet(\square\square)]} - n^{G_1}_{[j(\bullet\bullet)]} - n^{G_1}_{[\bullet(jj)]})$
	1 2 3, 4 5 6	$+\sum_{i\neq j} \binom{n_i^{C_1}}{2}(n^{C_2}_{(-1\bullet)} - n^{C_2}_{(-1i)} - n^{C_2}_{(-1j)})$ $+\sum_{i\neq j} n_i^{C_1} n_{-1}^{C_1}(n^{C_2}_{(\bullet\bullet)} - n^{C_2}_{(ii)} - n^{C_2}_{(jj)})$ $+\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{i\uparrow(-1\bullet)} - n^{C_2}_{i\uparrow(-1j)})$ $+n_{-1}^{C_1}(n^{C_2}_{\bullet\uparrow(\square\square)} - n^{C_2}_{j\uparrow(\bullet\bullet)} - n^{C_2}_{\bullet\uparrow(jj)})$ $+\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{-1\uparrow(\bullet\bullet)} - n^{C_2}_{-1\uparrow(ii)} - n^{C_2}_{-1\uparrow(jj)})$	1 2	$+\sum_{i\neq j} n^{G_1}_{(ii)}[n^{G_2}_{(-1\bullet)} - n^{G_2}_{(-1i)} - n^{G_2}_{(-1j)}]$ $+\sum_{i\neq j} n^{G_2}_{(ii)}[n^{G_1}_{(-1\bullet)} - n^{G_1}_{(-1i)} - n^{G_1}_{(-1j)}]$
	1 2 3 4 5 6	$+\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{(i(-1\bullet))} - n^{C_2}_{(i(-1j))})$ $+\sum_{i\neq j} (n^{C_1}_{[i(-1\bullet)]} - n^{C_1}_{[i(-1j)]})n_i^{C_2}$ $+\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{(\bullet -1)\uparrow i} - n^{C_2}_{(j-1)\uparrow i})$ $+\sum_{i\neq j} n_i^{C_1} n_{-1}^{C_1}(n^{C_2}_{\bullet\uparrow\bullet} - n^{C_2}_{j\uparrow j} - n^{C_2}_{i\uparrow i})$ $+n_{-1}^{C_1}(n^{C_2}_{\bullet\uparrow\square\uparrow\square} - n^{C_2}_{j\uparrow\bullet\uparrow\bullet} - n^{C_2}_{\bullet\uparrow j\uparrow j})$ $+\sum_{i\neq j} n_i^{C_1}(n^{C_2}_{-1\uparrow\bullet\uparrow\bullet} - n^{C_2}_{-1\uparrow i\uparrow i} - n^{C_2}_{-1\uparrow j\uparrow j})$	1 2	$+\sum_{i\neq j} n_i^{G_1}(n^{G_2}_{[i(-1\bullet)]} - n^{G_2}_{[i(-1j)]})$ $+\sum_{i\neq j} n_i^{G_2}(n^{G_1}_{[i(-1\bullet)]} - n^{G_1}_{[i(-1j)]})$