Efficient Algorithms for Computing the Triplet and Quartet Distance Between Trees of Arbitrary Degree

Gerth Stølting Brodal[*,†] Rolf Fagerberg[‡] Thomas Mailund[§]
Christian N. S. Pedersen[*,§] Andreas Sand[*,§]

Abstract

The triplet and quartet distances are distance measures to compare two rooted and two unrooted trees, respectively. The leaves of the two trees should have the same set of n labels. The distances are defined by enumerating all subsets of three labels (triplets) and four labels (quartets), respectively, and counting how often the induced topologies in the two input trees are different. In this paper we present efficient algorithms for computing these distances. We show how to compute the triplet distance in time $O(n \log n)$ and the quartet distance in time $O(dn \log n)$, where d is the maximal degree of any node in the two trees. Within the same time bounds, our framework also allows us to compute the parameterized triplet and quartet distances, where a parameter is introduced to weight resolved (binary) topologies against unresolved (non-binary) topologies. The previous best algorithm for computing the triplet and parameterized triplet distances have $O(n^2)$ running time, while the previous best algorithms for computing the quartet distance include an $O(d^9 n \log n)$ time algorithm and an $O(n^{2.688})$ time algorithm, where the latter can also compute the parameterized quartet distance. Since $d \leq n$, our algorithms improve on all these algorithms.

1 Introduction

Trees are widely used in many scientific fields to represent relationships, in particular in biology where trees are used e.g. to represent species relationships, so called phylogenies, the relationship between genes in gene families, or for hierarchical clustering of high-throughput experimental data. When using trees to investigate relationships, different data or different computational reconstruction methods, can lead to slightly different trees on the same set of leaves.

Several distance measures exist to compare trees constructed from different data, or trees constructed with the same data but using different reconstruction methods. These include the Robinson-Foulds distance [8], the triplet distance [4], and the quartet distance [6], which all enumerate certain features of the trees they compare and count how often the features differ between the two trees. The Robinson-Foulds distance enumerates all edges in the trees and counts how many of the induced bipartitions differ between the two input trees. The triplet distance (for rooted trees) and quartet distance (for unrooted trees) enumerate all subsets of leaves of size three and four, respectively, and count how many of the induced topologies differ between the two input trees.

For trees with n leaves, the Robinson-Foulds distance can be computed in time $O(n)$ [5], which is optimal. The quartet distance can be computed in time $O(n \log n)$ for binary trees [2], in time $O(d^9\, n \log n)$ for trees where all nodes have degree less than d [9], and in time $O(n^{2.688})$ for trees [7] of arbitrary degree. See also Christiansen *et al.* [3] for a number of algorithms for general trees with different tradeoffs depending on the degree of inner nodes. For the triplet distance, $O(n^2)$ time algorithms exist for both binary and general trees [1, 4].

In this paper we present efficient algorithms for computing the triplet and quartet distance between two trees of arbitrary degree. Our algorithms are inspired by the ideas introduced in [2], extended to handle the additional complexity of trees of arbitrary degree that imply non-binary triplet and quartet topologies (so-called unresolved topologies). We show how to compute the triplet distance in time $O(n \log n)$ and the quartet distance in time $O(dn \log n)$, where d is the maximal degree of any node in the two trees. Since $d \leq n$, our algorithms improve on all previous algorithms for computing the triplet and quartet distance.

Bansal *et al.* [1] introduce the concept of parameterized triplet and quartet distances, where a scaling

[*]Department of Computer Science, Aarhus University, Denmark. `gerth@cs.au.dk`.

[†]MADALGO, Center for Massive Data Algorithms, a Center of the Danish National Research Foundation.

[‡]Department of Mathematics and Computer Science, University of Southern Denmark, Denmark. `rolf@imada.sdu.dk`.

[§]Bioinformatics Research Center, Aarhus University, Denmark. `{mailund,cstorm,asand}@birc.au.dk`.

parameter is used to weight resolved (binary) topologies against unresolved (non-binary) topologies. Their $O(n^2)$ time algorithm for computing the triplet distance between general trees also computes the parameterized triplet distance. The previous best algorithm for computing the parameterized quartet distance is the $O(n^{2.688})$ time algorithm in [7] for the quartet distance between general trees that can be extended to compute the parameterized quartet distance without loss of time. The algorithms presented in this paper can also compute the parameterized triplet and quartet distance and thus also improve on all previous algorithms for computing these measures.

2 Overview

The triplet and quartet distance measures are defined for two trees, T_1 and T_2, having the same set of leaves, i.e, they both have n leaves and these are labeled with the same set of leaf labels. The triplet distance is defined for rooted trees, while the quartet distance is defined for unrooted trees.

A triplet is a set $\{i, j, k\}$ of three leaf labels. This is the smallest number of leaves for which the subtree induced by these leaves can have different topologies in two rooted trees T_1 and T_2. The possible topologies, given by the lowest common ancestor (LCA) relationships between the three leaves, are shown in Fig. 1. The last case is not possible for binary trees, but it is for trees of arbitrary degrees, which is the subject of this paper. The triplet distance is defined as the number of triplets whose topology differ in the two trees. It can naïvely be computed by enumerating all $\binom{n}{3}$ sets of three leafs and for each comparing the induced topologies in the two trees.

A quartet is a set $\{i, j, k, \ell\}$ of four leaf labels. This is the smallest number of leaves for which the subtree induced by these leaves can have different topologies in two unrooted trees T_1 and T_2. The possible topologies, given by the relationships of the paths between pairs of leaves in the quartet, are shown in Fig. 2. The last case is not possible for binary trees, but it is for trees of arbitrary degrees. For the remaining three cases, the set $\{i, j, k, \ell\}$ is *split* into two sets of two leaves. We sometimes use the notation $ij|k\ell$ for splits, with this particular instance referring to the leftmost topology in Fig. 2. Similar to the triplet distance, the quartet distance is defined as the number of quartets whose topology differ in the two trees. It can naïvely be computed by enumerating all $\binom{n}{4}$ sets of four leafs and for each comparing the induced topologies in the two trees.

Based on terminology from phylogenetic trees, the rightmost case in each of the Fig. 1 and 2 is called *unresolved*, while the remaining are called *resolved*. Both for triplets and quartets, the possible topologies in the two trees T_1 and T_2 can be partitioned into the five cases shown in Fig. 3. Note that in the resolved-resolved case, the triplet/quartet topologies may either agree or differ in the two trees. In the resolved-unresolved and unresolved-resolved cases they always differ, and in the unresolved-unresolved case, they always agree.

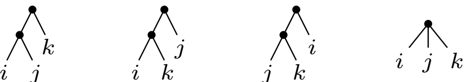


Figure 1: Triplet topologies.

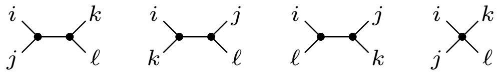


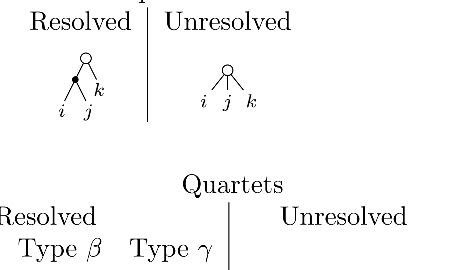
Figure 2: Quartet topologies.

		T_2 Resolved	T_2 Unresolved
T_1	Resolved	A: Agree B: Differ	C
	Unresolved	D	E

Figure 3: Cases for topologies in the two trees.

At the outermost level, our algorithms for calculating the triplet and quartet distances work by first finding the row and column sums in Fig. 3, i.e., finding the sums $A + B + C$, $D + E$, $A + B + D$, and $C + E$, where each capital letter designates the number of triplets/quartets falling in that category. As we show in Sec. 3, finding these sums can be done in $O(n)$ time by simple dynamic programming. Once all these are found, C, D and E can be found if A and B are known, or D, C and B can be found if A and E are known. The triplet or quartet distance can then be calculated as $B + C + D = (A + B + C) + (A + B + D) - 2A - B$. We will therefore concentrate on calculating A and B or E in the remainder of this paper.

The parameterized triplet and quartet distance is defined by Bansal *et al.* [1] as $B + p(C + D)$, for $0 \leq p \leq 1$, which makes it possible to weight the contribution of unresolved triplets/quartets to the total triplet/quartet distance. For $p = 0$, unresolved triplets/quartets do not contribute to the distance, i.e. unresolved triplets/quartets are ignored, while for $p = 1$, they contribute fully to the distance as is the case in the unparameterized triplet/quartet distance. The parameterized triplet/quartet distance can be calculated by $B + p((A + B + C) + (A + B + D) - 2A - 2B)$.



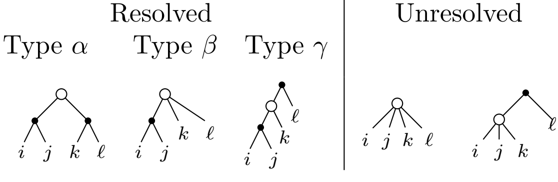


Figure 4: The anchor points (white nodes) of resolved and unresolved triplets and quartets. Note that edges in the figures correspond to disjoint paths in the underlying tree.

For rooted trees and triplets, we will show that A and E can be computed in time $O(n \log n)$. For unrooted trees and quartets, we will show that A can be computed in time $O(n \log n)$, and B can be computed in $O(dn \log n)$ time, where d is the maximal degree of any node in the two trees.

Our techniques for finding A and B, for both triplets and quartets, are very similar in nature, but quite technical. We therefore give a high-level introduction to these techniques, mainly formulated for finding A for triplets, in the rest of this section (some terminology introduced will be given for quartets too).

For a tree T, we assign each of the $\binom{n}{3}$ triplets, as well as each of the $\binom{n}{4}$ quartets, to a specific node in the tree. We say that the triplet/quartet is *anchored* at that node. For triplets, this anchor node is the lowest common ancestor in T of the three leaves. See Fig. 4, where anchor nodes are white. For quartets, the definition is slightly more involved: The quartet distance itself is defined for unrooted trees, but we will later need to root T in an (arbitrary) internal node. In the rooted tree, a quartet can appear in one of several forms. The possible forms, and our chosen anchor node for each, are shown in the bottom part of of Fig. 4.

For a node $v \in T$ we denote by τ_v the set of triplets anchored at v. Then $\{\tau_v \,|\, v \in T\}$ is a partition of the set $\mathcal{T}$ of triplets. Thus, $\{\tau_v \cap \tau_u \,|\, v \in T_1, u \in T_2\}$ is also a partition of $\mathcal{T}$. Our algorithm will find $\mathrm{A}(\mathcal{T}) = \sum_{v \in T_1} \sum_{u \in T_2} \mathrm{A}(\tau_v \cap \tau_u)$, where $\mathrm{A}(S)$ on a set S of triplets is the number of triplets being resolved in both T_1 and T_2, and having the same topology in both trees.

In the algorithm, we capture the resolved triplets in τ_v by a coloring of the leaves. For a node v, we say that the tree is colored according to v if leaves not in the subtree of v are colored with the color 0, all leaves in the ith subtree are colored i.

For two binary nodes $u \in T_1$ and $v \in T_2$ we can e.g. compute $\mathrm{A}(\tau_v \cap \tau_u)$ as follows. When colored according to v, then the resolved triplets in τ_v are exactly the triplets having two leaves colored 1 and one leaf colored 2, or two leaves colored 2 and one leaf colored 1. The number $\mathrm{A}(\tau_v \cap \tau_u)$ can be found as follows: let a and b be the two subtrees of u, let a_1 and b_1 be the number of leaves colored 1 in the two subtrees a and b, respectively, and a_2 and b_2 the two number of leaves colored 2. Then $\mathrm{A}(\tau_v \cap \tau_u) = \binom{a_2}{2} \cdot b_1 + \binom{b_2}{2} \cdot a_1 + \binom{b_1}{2} \cdot a_2 + \binom{a_1}{2} \cdot b_2$. We call these the number of resolved triplets of u compatible with the coloring. In later sections, we generalize such calculations to nodes of arbitrary degree.

Naïvely going through T_1 and for each node v coloring all leaves would take time $O(n)$ per node, for a total time of $O(n^2)$. We reduce the number of recolorings by a recursive coloring algorithm responsible for successively generating all colorings according to nodes in T_1. Going through T_2 for each coloring and counting the number of resolved triplets compatible with the coloring would also take time $O(n^2)$. We reduce this work by using a hierarchical decomposition tree (HDT) for T_2. A HDT is a balanced binary tree built on the nodes of T_2, with the nodes of the HDT corresponding to parts of T_2. In the HDT, the part of an inner node corresponds to the union of the parts of its two children, with the root corresponding to all of T_2. We show that we are able to decorate the nodes of the HDT with information quantifying the contribution of the node's part to the total count of the number of resolved triplets compatible with the coloring. The value at the root then contains $\mathrm{A}(\tau_v \cap \tau_u)$. In essence, the HDT performs the inner sum of $\mathrm{A}(\mathcal{T}) = \sum_{v \in T_1} \sum_{u \in T_2} \mathrm{A}(\tau_v \cap \tau_u)$, while the coloring algorithm performs the outer sum.

The crux of the information decorating scheme is that the information in a node in the HDT can be calculated in time $O(1)$ from the information in its two children, hence updates of colors of leaves of T_2 are cheap to propagate through the tree. The exception is for the case of calculating the value B for quartets, where the propagation from children to parents in the HDT takes time $O(d)$, which accounts for the extra factor of d in the running time of our algorithm for the quartet distance. The information decorating scheme is quite involved, with many counting variables defined to assist in the propagation of values and of the main count of the number of resolved triplets compatible with the coloring. The main technical contribution of the

paper is this information decorating scheme, and the construction of the HDT. Both are generalizations from binary trees to trees of arbitrary degree of techniques from [2].

The rest of the paper is organized as follows: In Sec. 3 we show how to find the row and column sums of Fig. 3. In Sec. 4, we describe the construction of the HDT. In Sec. 5, we give the main algorithm and prove it has the claimed time complexity. In Sec. 6 and Sec. 7 we describe the information decoration scheme.

3 Counting triplets and quartets in a single tree

In this section we describe how to count the number of resolved and unresolved triplets and quartets in a single tree with n leaves in $O(n)$ time using straightforward dynamic programming. Algorithms achieving this was also presented in [1], but our algorithm is significantly simpler and very intuitive. The total number of triplets and quartets in a tree are $\binom{n}{3}$ and $\binom{n}{4}$. In the following we describe how to compute the number of unresolved triplets and quartets in a tree. The number of resolved triplets and quartets can be obtained by subtracting the number of unresolved from the total number of triplets and quartets, respectively.

For each node v of the tree we compute the number of leaves n^v in the subtree rooted at v in $O(n)$ time by a bottom up traversal of the tree. During the traversal we count for each node v how many resolved and unresolved triplets and quartets are anchored at v as described below. The sum over the answers from the individual nodes is the final answer.

To count the number of unresolved triplets and quartets rooted at a node v of degree d, we compute the values s_i^v, p_i^v, t_i^v and q_i^v. s_i^v is the total number of leaves in the first i subtrees below v (where $n^v = s_d^v$), and the counters p_i^v, t_i^v and q_i^v are the number of sets of two, three and four leaves, respectively, where the leaves in a set belong to different subtrees among the first i subtrees below v. Letting n_i^v denote the size of the subtree rooted at the ith child of v, these values can be computed by dynamic programming using

$$\begin{aligned} s_1^v &= n_1^v & s_i^v &= s_{i-1}^v + n_i^v \\ p_1^v &= 0 & p_i^v &= p_{i-1}^v + n_i^v \cdot s_{i-1}^v \\ t_1^v &= 0 & t_i^v &= t_{i-1}^v + n_i^v \cdot p_{i-1}^v \\ q_1^v &= 0 & q_i^v &= q_{i-1}^v + n_i^v \cdot t_{i-1}^v \end{aligned}$$

The number of unresolved triplets rooted at v is the value t_d^v. The number of unresolved quartets rooted at v is $q_d^v + t_d^v(n - n^v)$, where the first term counts the number of unresolved quartets where all four leaves are in distinct subtrees below v, and the second term counts the number of unresolved quartets where one of the four leaves is not in the subtree of v.

Since we for each edge to a child use $O(1)$ time, the total time for computing the triplet and quartet information for a single tree is $O(n)$.

4 Hierarchical Decomposition of Rooted Trees

In order to count the triplets/quartets in T_1 and T_2, we build a data structure called the hierarchical decomposition tree (HDT) on top of T_2. The HDT is a balanced binary tree where each node, or *component*, corresponds to a set of nodes in T_2.

In later sections, we describe how we decorate the nodes of the HDT with counting information. In this section, we first describe our construction of an HDT on any rooted tree T with n nodes of arbitrary degree. We then prove that the construction works in time $O(n)$, where n is the number of nodes in T, and that the resulting HDT is *locally balanced*. A rooted tree is locally balanced if for any node v in the tree, the height of the subtree rooted in v is $O(\log |v|)$, where $|v|$ is the number of leaves in the subtree. Lemma 2 in [2] shows that the union of k root-to-leaf paths in a locally balanced tree contains $O(k + k \log \frac{n}{k})$ nodes, which is a property of the HDT we need later.

The nodes of the HDT are one of the following four different types of components, each corresponding to a specific type of subsets of nodes in T_2. See also Fig. 5.

- **L:** a leaf in T,
- **I:** an internal node in T,
- **C:** a connected subset of the nodes of T,
- **G:** a set of subtrees with roots being siblings in T.

For a type **G** component we require that it is downward closed, meaning that in T no edge from a node inside the component to a child crosses the boundary of the component. For a type **C** component we require that at most two edges in T_2 crosses the boundary of the component—more precisely, at most one going upwards in T, and at most one going downwards in T.

We define five actions on components: two *transformations*, which change the type of a component, and three *compositions*, which merge components. The five actions are depicted in Fig. 6. Each composition merges exactly two components into one new. We will view the new component as the parent of the two old components. In this way, a series of compositions will generate a binary tree. This is the HDT. Intuitively, the HDT is a balanced binary tree, where the leaves correspond to the nodes of an unbalanced tree with nodes of arbitrary degree.

Given a tree T with n nodes, we construct the HDT bottom-up in a number of rounds. Before the first round, we first convert each leaf of T to an **L** component,

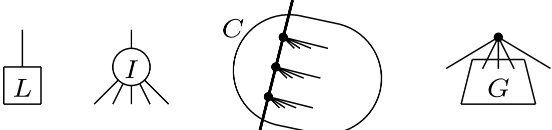


Figure 5: The four different types of components. **L** components contain a leaf from T_2, **I** components contain a single internal node from T_2, **C** components contain a connected set of nodes, and **G** components contain entire subtrees of siblings in T_2.

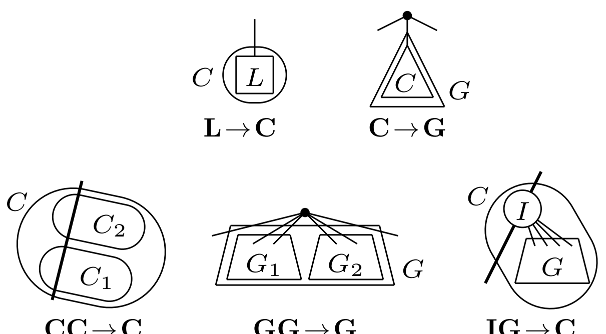


Figure 6: The two types of transformations (top) and the three types of compositions (bottom).

and each internal node to an **I** component. This gives a tree with I and L components as the nodes, and the edges of T as the edges. A round will transform one such tree into another smaller tree by performing a number of transformations and compositions of the types depicted in Fig. 6. Each round is working on the tree created by the previous round, and the process stops when the tree has size one. All trees are viewed as rooted in the component containing the root of T.

From the definitions of the transformations and compositions, the following are seen to be invariants of the HDT construction process: each component corresponds to (i.e., contains) a subset of the nodes of T; these subsets form a partition of the nodes of T; an **I** component corresponds to single internal nodes of T; a **C** component corresponds to a connected subset of nodes in T, with at most two edges connecting it to the rest of T, one from the root of the subset to its parent, and one (if the component is downwards open) from a leaf of the subset to a child; a **G** component is downwards closed, is a child of an **I** component, and corresponds to the subtrees in T rooted by a subset of the children of the node of this **I** component; each edge of the tree of components corresponds to some edge of the original T, except for edges between a **G** component and its parent **I** component, which may correspond to a set of edges in T (between the internal node corresponding to the **I** component and a subset of its children).

The compositions performed in a round will be non-overlapping, i.e., each composition merges two components existing at the start of the round. The new components formed by the compositions performed in a round will be internal nodes of the HDT, and the original components formed from the nodes of T before the first round will be the leaves of the HDT. Only compositions create internal nodes (components) of the HDT. The transformations are only changing the types of components, not merging components. They have no influence on the structure of the HDT, and are only included because they will ease the formulation of our later decoration of the nodes of the HDT with triplet/quartet counting information.

All work during the construction of the HDT is done via DFS traversals of the current tree of components, starting at the current trees root. Before the first round, we in a traversal first transform all **L** components into **C** components via the **L**→**C** transformation. Since no composition produces **L** components, such components will not appear again.

In each round, we in a traversal first transform all downwards closed **C** components being children of **I** components into **G** components via the **C** → **G** transformation. Now all children of **I** components are either **G** components or downwards open components (of type **C** or **I**). Call an **I** component *forking* if it has at least two downwards open children. In a second traversal, we perform the compositions of the round as follows. **G** components will be considered visited when visiting their parent **I** component. Thus, the traversal will traverse *paths* (possibly of length zero) consisting of non-forking **I** components and **C** components, as well as traverse forking **I** components. During the traversal, the following non-overlapping set of compositions are performed:

i) For an **I** component (forking or not) with $g \geq 2$ children being **G** components, pair siblings arbitrarily into $\lfloor g/2 \rfloor$ pairs and perform a composition of type **GG**→**G** on each pair.

ii) For a non-forking **I** component with only one child being a **G** component, perform one **IG** → **C** composition.

iii) For a maximal subpath of c consecutive **C** components, form $\lfloor c/2 \rfloor$ consecutive pairs (pairing topmost with second, third with fourth, etc.) and perform a **CC**→**C** composition on each pair.

Note that any **I** component having less than two

children must have exactly one **G** component as a child (when created before first round, all **I** components have at least two children, a number which can only decrease once some children have become **G** components, after which there will always be at least one **G** component child). Hence, all non-forking **I** components are covered by the cases above.

LEMMA 4.1. *Let T be a rooted tree with n nodes of arbitrary degree. The above construction of an HDT for T works in time $O(n)$, where n is the number of nodes in T. The resulting HDT is locally balanced: the subtree rooted at a node v has height $h \leq \lceil \log_{10/9} |v| \rceil$, where $|v|$ is the number of leaves in the subtree of v.*

Proof. We first consider the effect of a round on the entire current tree of components (which we may assume has size at least two, otherwise no round would take place). We will assign each component appearing at the start of the round to some composition performed, in such a way that no composition is assigned more than 9 components. Since each composition reduces the number of components by one, this shows that each round reduces the number of components by at least a factor of 9/8.

The two components taking part in a composition are assigned to that composition. In situation i), the **I** component and the possible non-paired **G** component is assigned to one of the compositions, giving at most four components assigned to any composition. In situation iii), there is possibly one non-paired **C** component to assign. If there is a non-forking **I** component at the upper end of the subpath of iii), we assign the non-paired **C** component to a composition taking place there (at least one is). So far, no composition has been assigned more than five components.

Left to assign is at most one **C** component per path (from the topmost subpath of **C** components), as well as the forking **I** components not covered by i), plus their possible single **G** component. Call a path with no forking **I** components at its lowest end a *leaf path*, and let m be the number of forking **I** components. Since the paths and the forking **I** components form a tree with all internal nodes at least binary, there are at least $m+1$ leaf paths. There are m non-leaf paths.

A leaf path has length at least two, since leaf paths start at forking **I** components by a downwards open component (except for the special case $m = 0$, where the tree is a single leaf path—this path has length at least two since the size of the tree is a least two). Hence, at least one composition takes place on each of the leaf paths. Distributing the components left to assign evenly among these compositions gives on average $1 + 3m/(m + 1) < 4$ per composition. In total, no composition is assigned more than nine components and all components have been assigned.

Hence, each round reduces the number of components by a factor of 9/8. It follows that the height of the HDT is $\lceil \log_{9/8} n \rceil$, where n is the number of nodes in T. Since each round can be performed in time linear in the current number of components, it also follows that the total construction time is $O(\sum_{i=0}^{\infty} n(8/9)^i) = O(n)$.

To prove that the HDT is locally balanced, we must, however, show that the subtree of *any* node v has logarithmic height. A node v in the HDT is a component of type **I**, **C**, or **G**. Any component corresponds to a subset S of the nodes of T. In the initialization phase, these nodes are converted to a set of components, after which point each round will reduce this set of components to a smaller set via the compositions performed. The height h of the subtree of v is the number of rounds performed until the set reaches size one. In any round until round h, the components being descendants of v forms a partition of S.

If v is an **I** component, it corresponds to a single node of T, so h is zero. If v is a **C** component, it corresponds to a connected subset of nodes in T, with at most two edges connecting it to the rest of T, one from the root of the subset to its parent, and one (if the component is downwards open) from a leaf of the subset to a child. Similarly, in any round until round h, the components partitioning S form in the current tree of components a connected subset T' of nodes with at most one edge leaving the subset upwards and at most one edge leaving it downwards. During a round, the traversal algorithm performing compositions enters T' via the upward edge and leaves via the downward edge. This can only differ from running the algorithm on T' seen as a tree by itself (starting at the root of T') if the two runs would differ on the pairing of consecutive **C** components on paths containing the entering and leaving edge. However, this would merge components inside and outside of T', which cannot happen until after round h (as v would correspond to a different subset than S of nodes in T). Hence, the analysis above for T applies to T': each round reduces its number of components by a factor of at least 9/8.

Finally, if v is a **G** component, it corresponds to the subtrees in T rooted by a subset of the children of the node of this **I** component. In any round until round h, the components partitioning S form a subset T' in the current tree of components of nodes consisting of subtrees below the same **I** component. For a given round, let s be the number of subtrees in T', and let $g \leq s$ be the number of those consisting of a single **G** component (measured after the **C**$\rightarrow$**G** transformations

at the start of the round). The remaining $s - g$ subtrees have size at least two, since their root is a downwards open component. Let m be the total number of components in these subtrees. We have $s \geq 2$, otherwise v would already have been formed. In this round, the algorithm performing compositions operates on the $s - g$ subtrees not being $\mathbf{G}$ components as if the algorithm was run on each of these individually (starting from their roots). Hence, m is reduced by a factor of $9/8$ in the round (as the subtrees have sizes at least two). The algorithm reduces the g $\mathbf{G}$ components to $\lceil g/2 \rceil$, which is at least a reduction by a factor of $3/2$, if $g \geq 2$. So for $g \geq 2$, the reduction in number of components is at least a factor of $9/8$. If $g = 1$, we have $s \geq 1$, so at least one other tree exists, and contains at least two components. The total number of components is $m+1$, of which m is reduced at least by a factor of $9/8$. For $g = 1$ the total fraction of components removed is therefore at least $\lceil m/9 \rceil/(m+1)$, which is at least $1/10$. So for any value of g, the number of components is reduced by a factor of at least $10/9$.

Summing up, no matter what type of component v is, we have $h \leq \lceil \log_{10/9} |v| \rceil$. □

5 The Main Algorithm

In this section, we give the main algorithm, which recursively traverses the internal nodes of T_1. For each internal node v, it colors the leaves of T_2 according to v, and then queries the HDT of T_2 for the number of triplets/quartets in T_2 compatible with that coloring. The results of the queries are simply summed during the traversal of T_1. Correctness follows from the discussion in Sect. 2.

The construction and balance of the HDT was discussed in the previous section. In the next sections, we show how to decorate its nodes with counting information which allows the queries to be answered in O(1) time from the information in the root of the HDT. This requires the HDT counting information to be updated after each coloring during the traversal of T_1. Recall that colors are integers between zero and d, where d is the maximal degree of any node in the two trees.

For the triplet distance, T_1 is already rooted. For the quartet distance, we first choose an arbitrary internal node as the root. The traversal of T_1 is a DFS-type traversal starting at the root. The following invariant is maintained during the traversal:

> *When entering a node v, all leaves in the subtree of v have the color 1, and all leaves not in the subtree of v have the color 0. When exiting v, all leaves in T_1 have the color 0.*

Before the traversal, all leaves are colored 1 to initialize the invariant. The traversal is then performed by the recursive algorithm COUNT shown in Fig. 7, starting with an initial call COUNT(r), where r is the root of the tree. In COUNT, the size of a subtree is defined as its number of leaves. The sizes of all subtrees are found in linear time during a preprocessing (cf. Sec. 3). The subroutine of coloring by a given color all leaves in a subtree of T_1 is performed by a traversal of the subtree, and takes time linear in the size of the subtree. The corresponding leaves of T_2 are simultaneously given the same colors (leaves in T_1 and T_2 with the same labels are kept paired together by bidirectional pointers).

In the time analysis of COUNT, we cover the work of coloring the leaves in a subtree in T_1 (and corresponding leaves in T_2) by charging $O(1)$ work to a leaf in T_1 each time it is colored. Note that the recoloring work in an instance of COUNT(v) happens at leaves *not* in the largest child c_1. Hence, if a leaf is charged at instances COUNT(v) at ancestor nodes $v = v_1, v_2, v_3, \ldots$ in T_1 listed in order of increasing depth in the tree, then the size of the subtree of v_{j+1} is a most half that of v_j. Hence, a leaf is only charged $O(\log n)$ times, for a total charge of $O(n \log n)$ during the entire run of COUNT(r).

The remaining part of the time analysis is to consider the work done to update the HDT when the colors of leaves change. Our goal is to show that this work, when recoloring leaves in subtree c_i in an instance COUNT(v), can be covered by charging $O(1 + \log(|v|/|c_i|))$ to each leaf in the subtree of c_i. If this is possible, each leaf will be charged $O(\sum_j (1 + \log |v_j|/|v_{j+1}|))$, where $v_1, v_2, v_3, \ldots$ are the ancestor nodes incurring charge, listed in order of increasing depth. Since this sum is telescoping in the second part of the terms, and by the analysis above has at most $\log n$ terms, the sum is $O(\log n)$, leading to $O(n \log n)$ total work.

To achieve the goal, we will extend the above version of COUNT(v) by compressing T_2 and its HDT during the recursion such that both are always of size $O(|v|)$ when invoking COUNT(v). As will be seen in the next sections, the time for updating a node in the HDT from its children when a leaf changes color inside the component of the node (which is a leaf in its subtree in the HDT) will be $O(1 + x)$, where x is the number of colors different from 0 used inside the component. The total updating work in the first for-loop in COUNT can be performed by first marking the paths in the HDT from all leaves recolored, and then propagating the change of information in the nodes of the HDT in a bottom-up manner, passing each marked node only once. Since the only colors in use at the start of COUNT(v) by the invariant are 0 and 1, a non-constant value of x for a node in the HDT means that

```
COUNT(v)
    if v is a leaf
        Color v by the color 0.
    else
        Let c_1 be the child of v with largest subtree, and let c_2, ... c_k be its remaining children.
        for i = 2 to k
            Color the leaves in the subtree of c_i by the color i.
        // Leaves are now colored according to v
        Query the HDT for the number of triplets/quartets in T_2 compatible with the coloring.
        Add that number to the global count.
        for i = 2 to k
            Color the leaves in the subtree of c_i by the color 0.
        COUNT(c_1)
        for i = 2 to k
            Color the leaves in the subtree of c_i by the color 1.
            COUNT(c_i)
```

Figure 7: The main algorithm performing a recursive traversal of T_1

$\Theta(x)$ different colors affected leaves in its subtree, hence for $\Theta(x)$ different values of i a subtree c_i had leaves below the node. Let $k_i = |c_i|$. We know from Lemma 4.1 that the HDT is locally balanced, and from Lemma 2 in [2] that the union of k_i root-to-leaf paths in a locally balanced tree of size n contains $O(k_i + k_i \log \frac{n}{k_i})$ nodes. Since here, $n = O(|v|)$ via the compression done, it follows that charging $O(1 + \log(|v|/|c_i|))$ to each leaf in the subtree of c_i covers the cost, as x different such unions of paths pass the node updated in the HDT. This was the goal. The same analysis applies to the second of the for-loops.

We now outline how to perform the compression of T_2 and its HDT during the recursion in COUNT. This will be done by adding two extra subroutines, *contract* and *extract* to COUNT. Similar contract and extract operations appeared in [2].

The operation contract is performed just before the recursive call COUNT(c_1), if the size of c_1 is less than some fixed fraction of the size of (the current version of) T_2, and it generates a new, contracted copy of T_2, to be used for the call COUNT(c_1). The operation acts on T_2 after the leaves corresponding to c_1 have been designated as in incontractible. Using the compositions from Sec. 4 greedily as long as one applies, a tree of size $O(|c_1|)$ can be constructed which contains all the designated leaves with the same induced topologies for all subsets of these leaves. Using a queue of possible composition points in the tree during the contraction, this can be done in $O(|v|)$ time. The HDT is built from scratch for the new tree in time $O(|c_1|)$, using Lemma 4.1. Since only two colors will be in use when contracting, the decoration of the HDT also takes time $O(|c_1|)$. Contract operations can happen along a path in the recursion tree going from nodes towards their largest child c_1. Since the sizes of the contracted versions of T_2 generated along such a path are exponentially decreasing, the cost of the topmost contract operation on the path is proportional to the total cost of all contract operation on the path. New paths can start in the recursive calls COUNT(c_i) for $i \geq 2$, but the cost of those can be charged to the (extracted version of) c_i, as will follow from the analysis of the extract operation below.

The second operation will extract a tree of size $O(|c_i|)$ containing the leaves of the subtree c_i with the same induced topologies for all subsets of these leaves. This is done just after the query to the HDT, when these leaves all have color i, and is repeated for all $i \geq 2$. It is done by finding the union of k_i root-to-leaf paths in the HDT from the leaves of c_i to the root by traversing the paths one by one, stopping when a previous traversed path is met. During a top-down traversal of the union of the paths, all subtrees hanging to sides of this union of paths are converted to single nodes, with the counting information reset to represent the situation where all leaves in these subtrees have color 0. This takes time $O(k_i + k_i \log \frac{|v|}{k_i})$, and gives a tree of size $O(k_i + k_i \log \frac{|v|}{k_i})$. Running contract on this tree (with the $k_i = |c_i|$ leaves marked incontractible) produces a tree of size $O(|c_i|)$, for which a HDT is then built (with the designate leaves having the color 1, the rest the color 0), all of this in the same time bound. By the analysis above, charging $O(1 + \log(|v|/|c_i|)$ to each leaf in the subtree of c_i covers the cost. As mentioned above, these new trees (and associated HDTs)

are produced for each c_i for $i \geq 2$ just after the query in $\textsc{Count}(v)$. They are used in the last for-loop when calling recursively, i.e., they take the place of c_i in the call $\textsc{Count}(c_i)$. This insures the desired linear size constraint for these invocations.

6 Counting shared triplets and quartets

In this section we consider the basic idea how to count the number of shared triplets and quartets between two trees T_1 and T_2, where all triplets and quartets are anchored at a node v of degree d in T_1. We assume all leaves in the ith subtree below v have been colored i, and all leaves not in v's subtree are colored 0.

The resolved triplets anchored at v in T_1 are exactly all triplets of leaves, where all leaves have a nonzero color, and two of the leaves have the same color. The unresolved triplets anchored at v are exactly all triplets with three nonzero and distinct colored leaves. Given the coloring of the leaves in T_1 we must count the number of triplets in T_2 satisfying the same color constraints, since these are exactly the triplets anchored at v in T_1 which also appear in T_2.

Unfortunately, a resolved quartet in T_1 can appear in three different ways, type α, β, and γ (see Fig. 4), since we consider a rooted version of T_1. Furthermore in T_2 the same quartet may appear as a resolved quartet of any of the types α, β, and γ. This leaves us with nine cases to search for. Furthermore, due to the coloring of the leaves, we should also search for all relevant permutations of the colors in T_2. Figure 8 summarizes the different cases we need to search for in T_2 given a coloring of T_1. Note that searching resolved quartets appearing as type α in T_1 and type γ in T_2 can be reduced to the symmetric problem of searching for quartets appearing as type γ in T_1 and type α in T_2 by swapping the two trees. Thus the cases α-β, α-γ, and β-γ, omitted in Fig. 8, can be identified by swapping the two trees and searching for β-α, γ-α, and γ-β, respectively. We will not discuss all cases, but only consider the case γ-β, where a quartet appears as type γ in T_1 anchored at v and type β in T_2. We know that one leaf must have color 0, and three leaves must have nonzero color, of which exactly two have the same color. In T_2 the quartet is rooted at a node v', such that the four leaves belong to three subtrees below v', and exactly two leaves are in the same subtree below v'. Since the split given by the quartet in T_1 is the split $ii|0j$, where $i \neq j$, this coloring can only be realized by the two colorings in the γ-β entry in Fig. 8.

We count the shared triplets and quartets that are anchored at v in T_1, by considering a hierarchical decomposition of T_2. A triplet or quartet is *anchored* at a node $\bar{v}$ in a hierarchical decomposition of T_2, if the component of $\bar{v}$ contains all leaves of the triplet or quartet, but neither of the two children of $\bar{v}$ has this property. Triplets and quartets can only be anchored at nodes corresponding to $\mathbf{CC} \to \mathbf{C}$ and $\mathbf{GG} \to \mathbf{G}$ component compositions.

We will count the number of shared triplets and quartets in a bottom up traversal of the hierarchical decomposition of T_2, where we in each node $\bar{v}$ compute a set of triplets and quartets anchored at $\bar{v}$ (Fig. 10 and 13) using a set of *counters* that we compute for each node of the hierarchical decomposition of T_2. The details are described in the following subsections. (Since dealing with quartets requires a significant number of cases, we have tried to be as systematic as possible in the figures. The cases could likely be reduced by combining some of the cases/counters, but to ensure that all cases were covered, we decided for a more systematic approach in the presentation.)

7 Counters for components

The counters we maintain for each component are shown in Fig. 11. The counters required for counting shared triplets are marked †. The majority of the counters are only used for counting shared resolved quartets.

We first explain some basic counters. For a $\mathbf{C}$ component C we let n_i^C denote the number of leaves with color i in component C. Here i can be any of the colors $1, \ldots, d$. The number of leaves colored zero in C is n_0^C. The number of leaves with a nonzero color is denoted $n_\bullet$. In the illustrations in Fig. 11 we use j to indicate a color that can take several values. Similarly we have counters for $\mathbf{G}$ components. Some counters are defined for both $\mathbf{C}$ and $\mathbf{G}$ components, and some are only defined for $\mathbf{C}$ components. Those defined for both $\mathbf{C}$ and $\mathbf{G}$ components have a superscript X in Fig. 11 and Fig. 12.

We now consider more advanced counters. Each of the remaining counters count the number of pairs or triplets in a component with some constraints on the coloring and relative placement in the components, This is captured by the subscripts. The color constraint is indicated by 0, $i \in \{1, \ldots, d\}$, $\bullet$ and $\square$. Here $\bullet$ denotes any color in $\{1, \ldots, d\}$ if i does not appear in the subscript, and otherwise any color in $\{1, \ldots, d\} \setminus \{i\}$, and $\square$ is any color in $\{1, \ldots, d\} \setminus \{i, \bullet\}$. Two colors in a subscript, e.g. $n_{i\bullet}^G$, counts the number of pairs of leaves in G with non-zero colors i and j, for any $j \neq i$, where the leaves are in two distinct subtrees of the super root (i.e. the LCA of the two leaves in T_2 is the super root of G). For a C component the corresponding counter $n_{i\bullet}^C$ counts the same type of pairs of leaves, but where the LCA of the two leaves is a node on the external path (i.e. the path connecting the two external edges) and

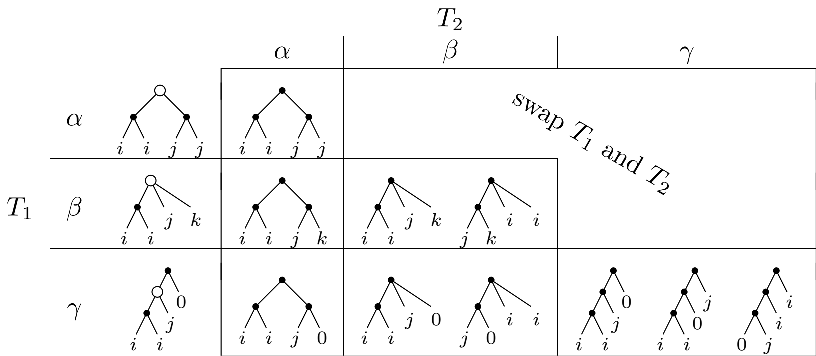


Figure 8: Shared resolved quartets – the different cases that should be counted

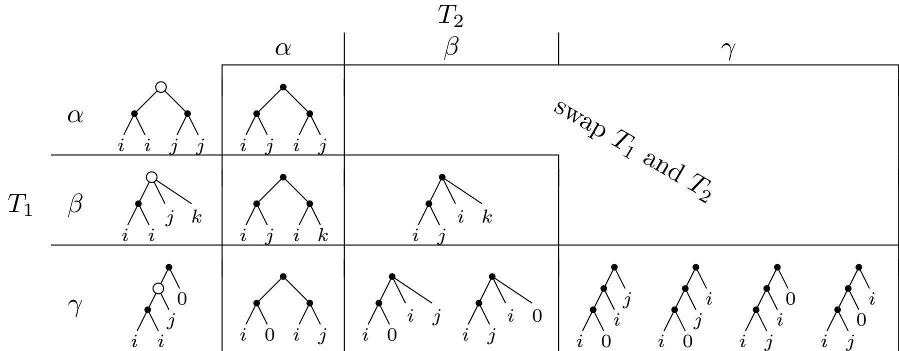


Figure 9: Quartets resolved differently in the two trees – the different cases that should be counted

both leaves are not contained in the subtree rooted at the child subtree on the external path.

Parenthesis around the subscripts, e.g. $n^C_{(\bullet\square)}$ and $n^G_{(\bullet\square)}$, require the LCA of the leaves not to be a node on the external path or the super root, respectively. E.g. for the pairs counted by $n^C_{(\bullet\square)}$ the colors are required to be two distinct colors in $\{1,\ldots,d\}$. Going one step further, e.g., we let $n^G_{(i(0\bullet))}$ denote the number of resolved triplets in G where all three leaves are in the same subtree below the super root (outer parenthesis in the subscript), one of the leaves should be colored i, one colored zero, and the last leaf should have any color in $\{1,\ldots,d\}\setminus\{i\}$ ($\bullet$ in the subscript), and the LCA of the last two leaves should be lower than the LCA with i (the inner parenthesis). Dropping the outer parenthesis, i.e. $n^G_{i(0\bullet)}$, requires the triplets to be rooted at the super root of G.

For C components we use $\uparrow$ to require that the leaves should branch out from different nodes on the external path. E.g. for $n^C_{i\uparrow\bullet\uparrow 0}$ we count triplets with leaves colored i, $\{1,\ldots,d\}\setminus\{i\}$, and 0, where the leaves should branch out from three different nodes on the external path, where the left-to-right order in the subscript equals bottom-up order on the external path. Parenthesis and $\uparrow$ can also be combined, e.g. $n^C_{i\uparrow(\bullet\square)}$ (see Fig. 11).

Finally we use squared parenthesis to indicate that we do not require anything special with respect to the super root or nodes on the external path. E.g. $n^C_{[0(ii)]}$ counts all triplets consisting of three leaves in C colored 0, i, and i, and where the LCA of the two i leaves is lower than the LCA with the leaf colored 0. We do not care if the triplets are rooted on the external path or contains edges from the external path.

Figure 11 lists and illustrates all the up to 53 different types of counters for a component we need to maintain. Note that 38 of the types involves i in the subscript, i.e. each of these covers d different counters, one for each possible value of $i=\{1,\ldots,d\}$. In total we need to maintain up to $15+38d$ counters for a component.

7.1 Counters for shared resolved triplets Figure 11 illustrates the counters for each each node $\bar{v}$ of the hierarchical decomposition of T_2, and Fig. 12 shows (in a very condensed form) how to compute these, given the counters for the children of $\bar{v}$. For a **CC**$\rightarrow$**C** composition it is trivial that $n^C_i = n^{C_1}_i + n^{C_2}_i$, i.e. the number of leaves colored i in C is the sum of the number of leaves colored i in C_1 and C_2.

A nontrivial case is $n^X_{i\bullet}$. For a $\mathbf{CC}\to\mathbf{C}$ composition we have $n^C_{i\bullet} = n^{C_1}_{i\bullet} + n^{C_2}_{i\bullet}$, since we require that the two leaves colored i and j, $j \neq i$, branch out from the same node on the external path. For a $\mathbf{GG}\to\mathbf{G}$ composition the equation becomes

$$n^G_{i\bullet} = n^{G_1}_{i\bullet} + n^{G_2}_{i\bullet} + n^{G_1}_i(n^{G_2}_\bullet - n^{G_2}_i) + n^{G_1}_i(n^{G_2}_\bullet - n^{G_2}_i) ,$$

where the four terms come from: (1) that the pair is in G_1, (2) in G_2, (3) that the i leaf is in G_1 and the non-i leaf is in G_2, or (4) the symmetric case of (3) with G_1 and G_2 swapped. Since these four terms are common for many $\mathbf{GG}\to\mathbf{G}$ compositions, we restate this as

$$n^G_{i\bullet} = n^{G_1}_{i\bullet} + n^{G_2}_{i\bullet} + g_{i\bullet}(G_1, G_2) + g_{i\bullet}(G_2, G_1)$$

$$g_{i\bullet}(G', G'') = n^{G'}_i(n^{G''}_\bullet - n^{G''}_i) ,$$

such that we only have to define $g_{i\bullet}$ for the $\mathbf{GG}\to\mathbf{G}$ composition. In the table we use "$-$" as a placeholder for the subscripts in a row, when defining generic expressions applying to several rows.

Counters involving $\uparrow$ in the subscript only apply to $\mathbf{C}$ components. E.g. consider $n^C_{i\uparrow0\uparrow\bullet}$ which can be computed as

$$\begin{aligned} n^C_{i\uparrow0\uparrow\bullet} &= n^{C_1}_{i\uparrow0\uparrow\bullet} + n^{C_2}_{i\uparrow0\uparrow\bullet} \\ &+ \left(n^{C_1}_i(n^{C_2}_{0\uparrow\bullet} - n^{C_2}_{0\uparrow i}) + n^{C_1}_{i\uparrow0}(n^{C_2}_\bullet - n^{C_2}_i)\right) , \end{aligned}$$

where the first two terms are common to many counters, and the last counter is referred to as $f_{i\uparrow0\uparrow\bullet}$ in Fig. 12. Here $f_{i\uparrow0\uparrow\bullet}$ can be computed as the number of triplets where either 1) i branches out in C_1, and 0 and non-i branch out in C_2, or 2) where i and 0 branch out in C_1 and only non-i branches out in C_2.

We leave it to the reader to do the tedious work of checking all the calculations in Fig. 12. It should be noted that we only need to keep track of counters that either (1) have no i in the subscript, or (2) with an i in the subscript and at least one leaf in the component is colored i (these counters are always zero if no leaf is colored i in the component). If there are x different colors from $\{1, \ldots, d\}$ used to color the leaves in a component, then up to $15 + 38x$ counters should be computed. Eight of the sums is a sum over x terms (contained in the box of Fig. 12), and the remaining $7 + 38x$ sums only contain a constant number of terms. It follows that all counters at a node in the hierarchical composition can be computed in $O(1 + x)$ time.

7.2 Counting shared triplets Having defined the counters, we can now describe how to compute the number of shared resolved and unresolved triplets anchored at a node in the hierarchical decomposition of T_2. Figure 10 lists the required computations.

The number of shared resolved triplets anchored in a $\mathbf{GG}\to\mathbf{G}$ composition is given by the equation

$$\sum_{i=1}^{d} n^{G_1}_{(ii)}\left(n^{G_2}_\bullet - n^{G_2}_i\right) + \sum_{i=1}^{d} n^{G_2}_{(ii)}\left(n^{G_1}_\bullet - n^{G_1}_i\right) ,$$

since a triplet anchored at v in T_1 and at the super root of G in T_2 must have two equally colored leaves (color i) in one subtree below the super root of G and another colored leaf in another subtree, where one subtree is in G_1 and the other in G_2. Since most sums have two symmetric terms with G_1 and G_2 swapped, we have introduced the function $\sigma[f(G', G'')](G_1, G_2) = f(G_1, G_2) + f(G_2, G_1)$, such that e.g. the above sum can be rewritten as

$$\sigma\left[\sum_{i=1}^{d} n^{G'}_{(ii)}\left(n^{G''}_\bullet - n^{G''}_i\right)\right] .$$

The number of shared resolved triplets anchored in a $\mathbf{CC}\to\mathbf{C}$ composition involves different cases. A resolved triplet anchored at v in T_1 has two leaves colored i and one colored j, for some colors i and j with $i \neq j$. The edge between C_1 and C_2 in T_2 can split the triplet in four different ways – one for each edge of the triplet. To make sure we count all cases we have labeled the edges of the triplet with numbers 1-4, and for each sum counting resolved shared triplets anchored in the $\mathbf{CC}\to\mathbf{C}$ node, we list the splits of the triplet which are covered. The three sums in Fig. 10 count the number of triplets, where C_1 contains one i, both is, and j, respectively (note that C_1 cannot contain one i and j without containing both is).

For a node in the hierarchical decomposition of T_2 we can compute the number of shared resolved triplets anchored at this node in $O(1 + x)$ time, where x is the number of distinct colors from $\{1, \ldots, d\}$ used to color the leaves in the component.

For shared unresolved triplets the counting is simpler (see Fig. 10) and can also be done in $O(1+x)$ time. Here the three leaves of a triplet have different colors. For $\mathbf{CC}\to\mathbf{C}$ compositions one leaf must be in C_1 (colored i), and the two other colored leaves must branch out from a single node on the external path in C_2 to two different subtrees. For $\mathbf{GG}\to\mathbf{G}$ compositions the triplet must branch to three different subtrees of the super root of G, where one leaf (colored i) is in G_1 or G_2, respectively, and the other leaves are in G_2 or G_1, respectively.

7.3 Counting shared resolved quartets Counting shared resolved quartets (Fig. 13) follows the same approach as for counting shared triplets, except that the number of cases to consider increases significantly. The

T_1	T_2		**CC→C**	**GG→G**
Counting resolved-resolved triplets				
		1,2	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{i\uparrow\bullet}^{C_2}$	$\sigma\Big[\sum_{i=1}^{d} n_{(ii)}^{G'} \Big(n_\bullet^{G''} - n_i^{G''}\Big)\Big]$
i i j	i i j	3	$\sum_{i=1}^{d} \binom{n_i^{C_1}}{2} \Big(n_\bullet^{C_2} - n_i^{C_2}\Big)$	
		4	$\sum_{i=1}^{d} \Big(n_\bullet^{C_1} - n_i^{C_1}\Big) n_{(ii)}^{C_2}$	
Counting unresolved-unresolved triplets				
i j k	i j k	1,2,3	$\sum_{i=1}^{d} n_i^{C_1} \Big(n_{\bullet\square}^{C_2} - n_{i\bullet}^{C_2}\Big)$	$\sigma\Big[\sum_{i=1}^{d} n_i^{G'} \Big(n_{\bullet\square}^{G''} - n_{i\bullet}^{G''}\Big)\Big]$

Figure 10: Contribution to the number of shared triplets for component compositions

asymptotic time is also the same, but with a significantly bigger constant. First, each resolved quartet in T_1 can appear as type α-γ, and the identical quartet in T_2 can also appear as one of the types α-γ. This implies nine different cases. Three cases can by symmetry be counted by swapping T_1 and T_2, see Fig. 8. For quartets of type α and γ in T_2 there are six possible splits to consider in the **CC**→**C** composition, whereas type β in T_2 only needs five possible splits to consider. For **GG**→**G** compositions there is one case if the root of the quartet (not the anchor) has degree two (types α and γ) and three cases if the root has degree three (type β), since for degree three we need to consider the possible ways that the three subtrees can be distributed between the two subcomponents (actually there are two and six cases, respectively, since there are also the symmetric cases with G_1 and G_2 swapped, but this is counted using $\sigma[\cdot]$). In Fig. 13 we next to each sum write the distribution of the edges from the two subcomponents covered by the sum, e.g. "34:5" indicate that edges 3 and 4 lead to one component, and 5 to the other.

7.4 Counting disagreeing resolved quartets In this section we count the number of four tuples of leaves that are resolved quartets in both trees, but where the four leaves define different quartets (splits) in the two trees. We take the same general approach as for counting shared resolved quartets. The cases that should be considered are summarized in Fig. 9. The computation of the contribution for each of the cases at a node in a hierarchical decompostion of T_2 is shown in Fig. 16. The additional counters required for the computation are listen in Fig. 14, and their computation in Fig. 15. Sums marked $\star$ are the bottleneck of the computation, since they require $O((1+x)^2)$ time to compute, where x is the number of different colors from $\{1, ..., d\}$ used to color the leaves in the component. The total time to handle a node in a hierarchical decomposition of T_2 is $O((1+x)^2) = O((1+x)d)$, i.e. in the worst case a factor d larger than required by the analysis in Sec. 5 for achieving $O(n \log n)$ running time. It follows that the total time for computing the number of disagreeing resolved quartets becomes $O(dn \log n)$.

The simplest case illustrating the bottleneck in our approach is the computation of the contribution of the case α-α in a **GG** → **G** composition. The quartets in the two trees are

i i j j — quartet in T_1 i j i j — quartet in T_2

and the contribution can be computed in $O((1+x)^2)$ time as $\sum_{i=1}^{d-1} \sum_{j=i+1}^{d} n_{(ij)}^{G_1} \cdot n_{(ij)}^{G_2}$.

Interestingly, we arrived at the same kind of bottleneck when trying to compute the number of unresolved quartets in T_2 that are resolved in T_1 directly.

References

[1] M. S. Bansal, J. Dong, and D. Fernández-Baca. Comparing and aggregating partially resolved trees. *Theoretical Computer Science*, 412(48):6634–6652, 2011.

[2] G. S. Brodal, R. Fagerberg, and C. N. S. Pedersen. Computing the quartet distance between evolutionary trees in time $O(n \log n)$. *Algorithmica*, 38(2):377–395, 2004.

[3] C. Christiansen, T. Mailund, C. N. S. Pedersen, M. Randers, and M. S. Stissing. Fast calculation of the quartet distance between trees of arbitrary degree. *Algorithms for Molecular Biology*, 13, 2006.

[4] D. E. Critchlow, D. K. Pearl, and C. L. Qian. The triples distance for rooted bifurcating phylogenetic trees. *Systematic Biology*, 45(3):323–334, 1996.

[5] W. H. E. Day. Optimal-algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.

[6] G. F. Estabrook, F. R. McMorris, and C. A. Meacham. Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. *Systematic Zoology*, 34(2):193, 1985.

[7] J. Nielsen, A. Kristensen, T. Mailund, and C. N. S. Pedersen. A sub-cubic time algorithm for computing the quartet distance between two general trees. *Algorithms for Molecular Biology*, 6(1):15, 2011.

[8] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

[9] M. S. Stissing, C. N. S. Pedersen, T. Mailund, G. S. Brodal, and R. Fagerberg. Computing the quartet distance between evolutionary trees of bounded degree. In *Proceedings of the 5th Asia-Pacific Bioinformatics Conference (APBC)*, pages 101–110. Imperial College Press, 2007.

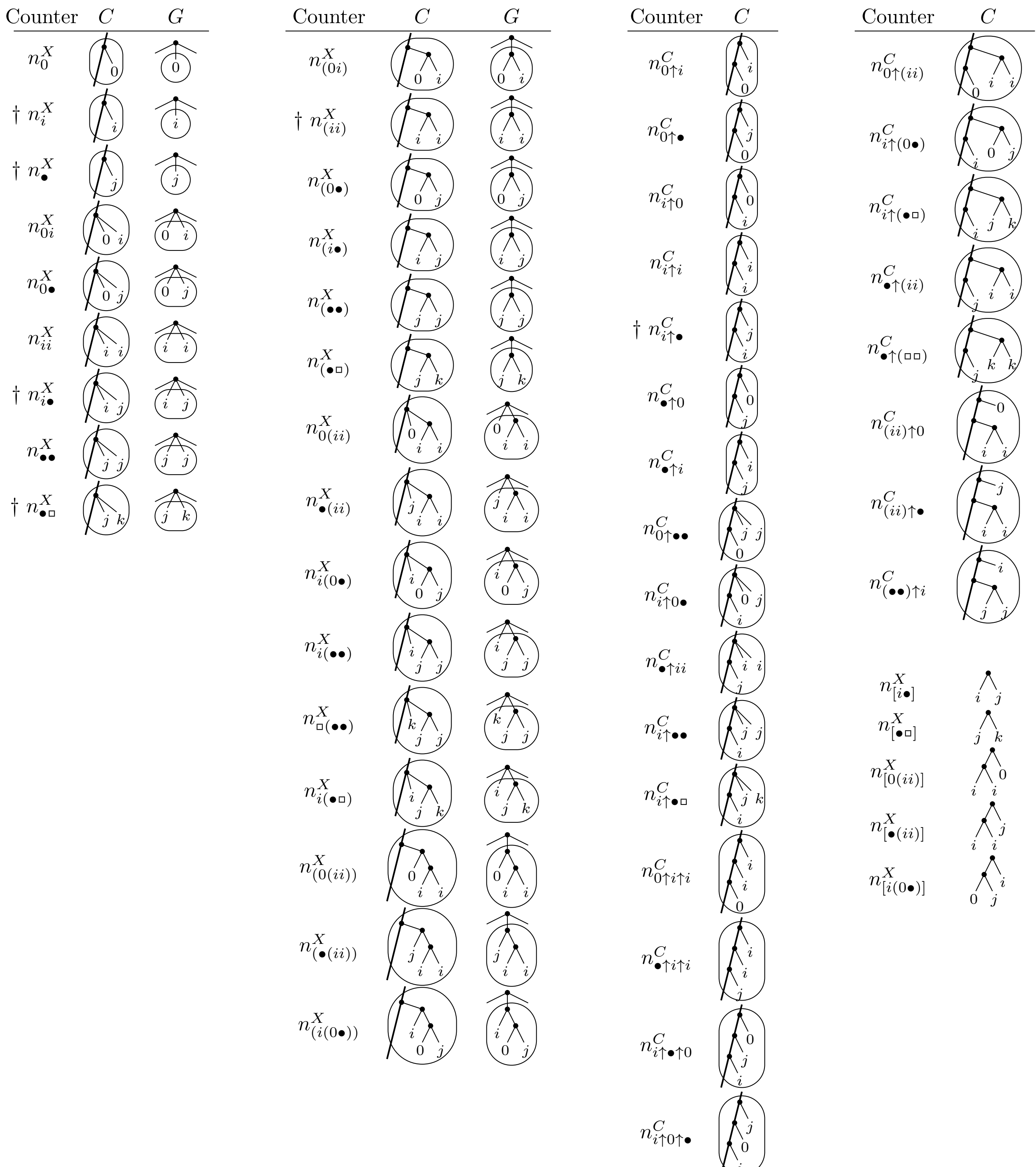
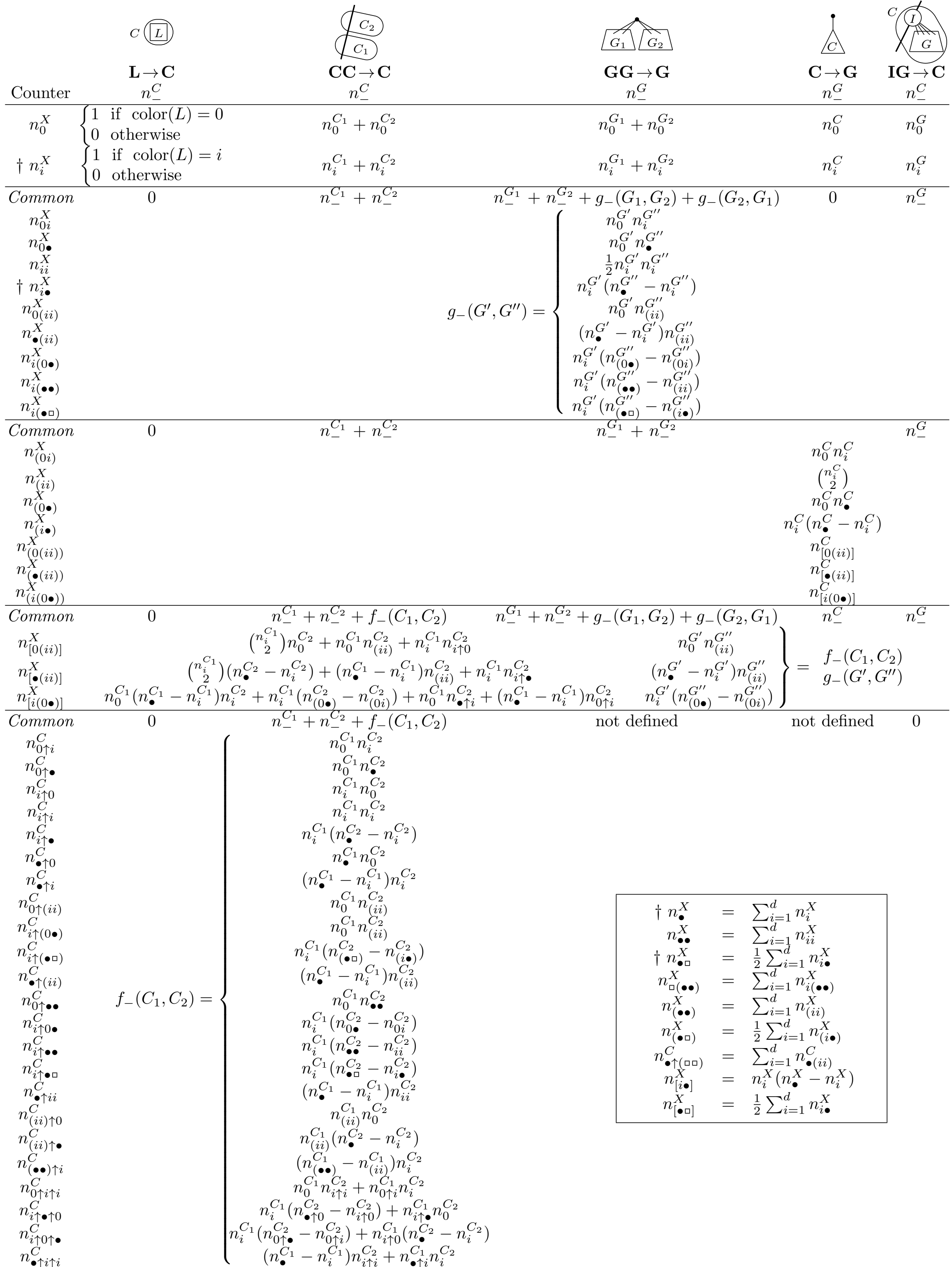


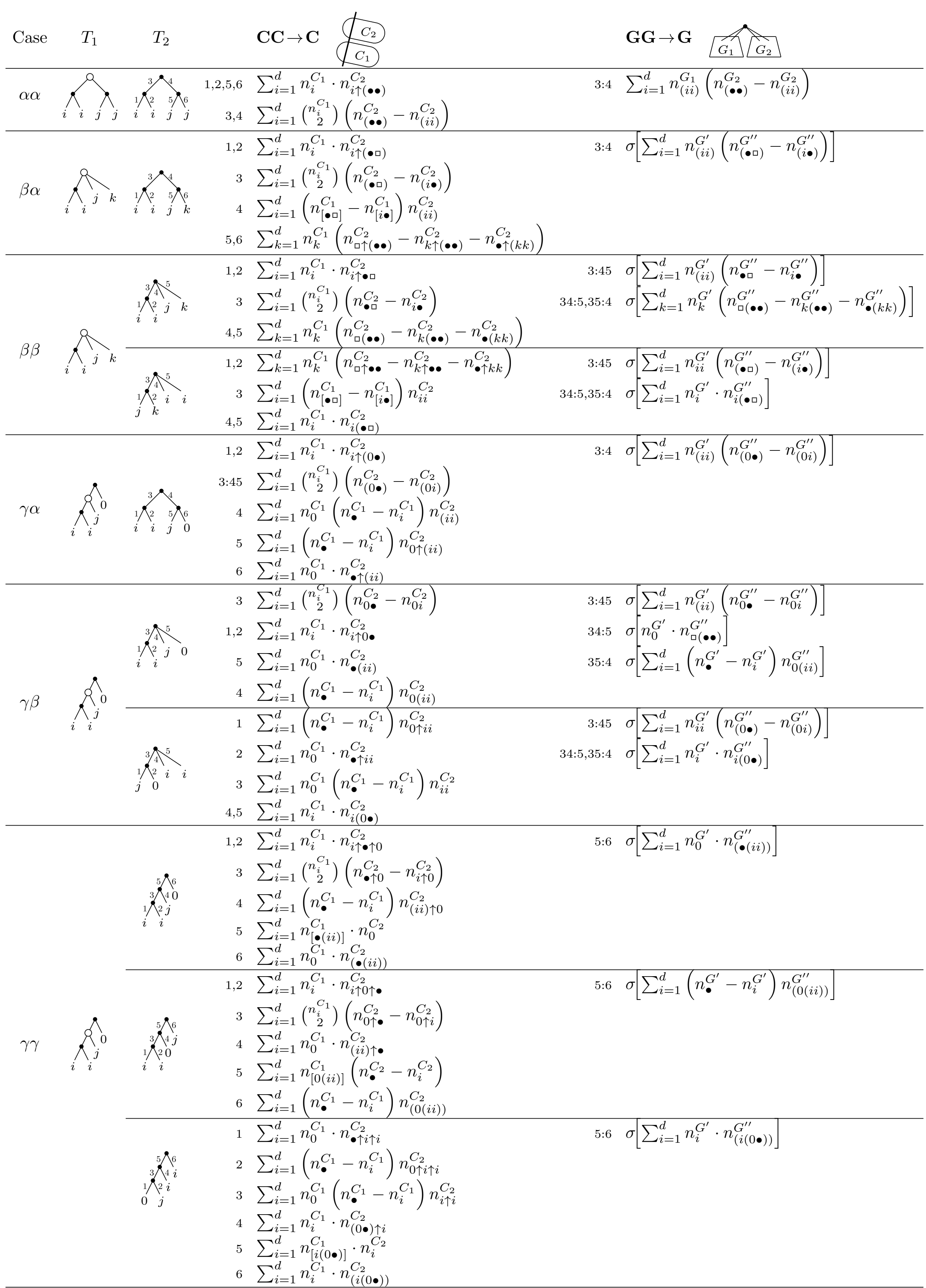
Figure 11: Illustration of the different counters used to count shared triplets and quartets



Counter	$\mathbf{L \to C}$	$\mathbf{CC \to C}$	$\mathbf{GG \to G}$	$\mathbf{C \to G}$	$\mathbf{IG \to C}$
	n_-^C	n_-^C	n_-^G	n_-^G	n_-^C
n_0^X	$\begin{cases}1 & \text{if color}(L)=0\\ 0 & \text{otherwise}\end{cases}$	$n_0^{C_1}+n_0^{C_2}$	$n_0^{G_1}+n_0^{G_2}$	n_0^C	n_0^G
$\dagger\ n_i^X$	$\begin{cases}1 & \text{if color}(L)=i\\ 0 & \text{otherwise}\end{cases}$	$n_i^{C_1}+n_i^{C_2}$	$n_i^{G_1}+n_i^{G_2}$	n_i^C	n_i^G
Common	0	$n_-^{C_1}+n_-^{C_2}$	$n_-^{G_1}+n_-^{G_2}+g_-(G_1,G_2)+g_-(G_2,G_1)$	0	n_-^G
n_{0i}^X			$g_-(G',G'') =$ $n_0^{G'} n_i^{G''}$		
$n_{0\bullet}^X$			$n_0^{G'} n_\bullet^{G''}$		
n_{ii}^X			$\frac{1}{2} n_i^{G'} n_i^{G''}$		
$\dagger\ n_{i\bullet}^X$			$n_i^{G'}(n_\bullet^{G''}-n_i^{G''})$		
$n_{0(ii)}^X$			$n_0^{G'} n_{(ii)}^{G''}$		
$n_{\bullet(ii)}^X$			$(n_\bullet^{G'}-n_i^{G'})n_{(ii)}^{G''}$		
$n_{i(0\bullet)}^X$			$n_i^{G'}(n_{(0\bullet)}^{G''}-n_{(0i)}^{G''})$		
$n_{i(\bullet\bullet)}^X$			$n_i^{G'}(n_{(\bullet\bullet)}^{G''}-n_{(ii)}^{G''})$		
$n_{i(\bullet\square)}^X$			$n_i^{G'}(n_{(\bullet\square)}^{G''}-n_{(i\bullet)}^{G''})$		
Common	0	$n_-^{C_1}+n_-^{C_2}$	$n_-^{G_1}+n_-^{G_2}$		n_-^G
$n_{(0i)}^X$				$n_0^C n_i^C$	
$n_{(ii)}^X$				$\binom{n_i^C}{2}$	
$n_{(0\bullet)}^X$				$n_0^C n_\bullet^C$	
$n_{(i\bullet)}^X$				$n_i^C(n_\bullet^C-n_i^C)$	
$n_{(0(ii))}^X$				$n_{[0(ii)]}^C$	
$n_{(\bullet(ii))}^X$				$n_{[\bullet(ii)]}^C$	
$n_{(i(0\bullet))}^X$				$n_{[i(0\bullet)]}^C$	
Common	0	$n_-^{C_1}+n_-^{C_2}+f_-(C_1,C_2)$	$n_-^{G_1}+n_-^{G_2}+g_-(G_1,G_2)+g_-(G_2,G_1)$	n_-^C	n_-^G
$n_{[0(ii)]}^X$		$\binom{n_i^{C_1}}{2}n_0^{C_2}+n_0^{C_1}n_{(ii)}^{C_2}+n_i^{C_1}n_{i\uparrow 0}^{C_2}$	$n_0^{G'}n_{(ii)}^{G''}$		
$n_{[\bullet(ii)]}^X$		$\binom{n_i^{C_1}}{2}(n_\bullet^{C_2}-n_i^{C_2})+(n_\bullet^{C_1}-n_i^{C_1})n_{(ii)}^{C_2}+n_i^{C_1}n_{i\uparrow\bullet}^{C_2}$	$(n_\bullet^{G'}-n_i^{G'})n_{(ii)}^{G''}$		
$n_{[i(0\bullet)]}^X$		$n_0^{C_1}(n_\bullet^{C_1}-n_i^{C_1})n_i^{C_2}+n_i^{C_1}(n_{(0\bullet)}^{C_2}-n_{(0i)}^{C_2})+n_0^{C_1}n_{\bullet\uparrow i}^{C_2}+(n_\bullet^{C_1}-n_i^{C_1})n_{0\uparrow i}^{C_2}$	$n_i^{G'}(n_{(0\bullet)}^{G''}-n_{(0i)}^{G''})$		
		$= f_-(C_1,C_2)$	$= g_-(G',G'')$		
Common	0	$n_-^{C_1}+n_-^{C_2}+f_-(C_1,C_2)$	not defined	not defined	0
$n_{0\uparrow i}^C$		$f_-(C_1,C_2) =$ $n_0^{C_1}n_i^{C_2}$			
$n_{0\uparrow\bullet}^C$		$n_0^{C_1}n_\bullet^{C_2}$			
$n_{i\uparrow 0}^C$		$n_i^{C_1}n_0^{C_2}$			
$n_{i\uparrow i}^C$		$n_i^{C_1}n_i^{C_2}$			
$n_{i\uparrow\bullet}^C$		$n_i^{C_1}(n_\bullet^{C_2}-n_i^{C_2})$			
$n_{\bullet\uparrow 0}^C$		$n_\bullet^{C_1}n_0^{C_2}$			
$n_{\bullet\uparrow i}^C$		$(n_\bullet^{C_1}-n_i^{C_1})n_i^{C_2}$			
$n_{0\uparrow(ii)}^C$		$n_0^{C_1}n_{(ii)}^{C_2}$			
$n_{i\uparrow(0\bullet)}^C$		$n_0^{C_1}n_{(ii)}^{C_2}$			
$n_{i\uparrow(\bullet\square)}^C$		$n_i^{C_1}(n_{(\bullet\square)}^{C_2}-n_{(i\bullet)}^{C_2})$			
$n_{\bullet\uparrow(ii)}^C$		$(n_\bullet^{C_1}-n_i^{C_1})n_{(ii)}^{C_2}$			
$n_{0\uparrow\bullet\bullet}^C$		$n_0^{C_1}n_{\bullet\bullet}^{C_2}$			
$n_{i\uparrow 0\bullet}^C$		$n_i^{C_1}(n_{0\bullet}^{C_2}-n_{0i}^{C_2})$			
$n_{i\uparrow\bullet\bullet}^C$		$n_i^{C_1}(n_{\bullet\bullet}^{C_2}-n_{ii}^{C_2})$			
$n_{i\uparrow\bullet\square}^C$		$n_i^{C_1}(n_{\bullet\square}^{C_2}-n_{i\bullet}^{C_2})$			
$n_{\bullet\uparrow ii}^C$		$(n_\bullet^{C_1}-n_i^{C_1})n_{ii}^{C_2}$			
$n_{(ii)\uparrow 0}^C$		$n_{(ii)}^{C_1}n_0^{C_2}$			
$n_{(ii)\uparrow\bullet}^C$		$n_{(ii)}^{C_1}(n_\bullet^{C_2}-n_i^{C_2})$			
$n_{(\bullet\bullet)\uparrow i}^C$		$(n_{(\bullet\bullet)}^{C_1}-n_{(ii)}^{C_1})n_i^{C_2}$			
$n_{0\uparrow i\uparrow i}^C$		$n_0^{C_1}n_{i\uparrow i}^{C_2}+n_{0\uparrow i}^{C_1}n_i^{C_2}$			
$n_{i\uparrow\bullet\uparrow 0}^C$		$n_i^{C_1}(n_{\bullet\uparrow 0}^{C_2}-n_{i\uparrow 0}^{C_2})+n_{i\uparrow\bullet}^{C_1}n_0^{C_2}$			
$n_{i\uparrow 0\uparrow\bullet}^C$		$n_i^{C_1}(n_{0\uparrow\bullet}^{C_2}-n_{0\uparrow i}^{C_2})+n_{i\uparrow 0}^{C_1}(n_\bullet^{C_2}-n_i^{C_2})$			
$n_{\bullet\uparrow i\uparrow i}^C$		$(n_\bullet^{C_1}-n_i^{C_1})n_{i\uparrow i}^{C_2}+n_{\bullet\uparrow i}^{C_1}n_i^{C_2}$			

$\dagger\ n_\bullet^X$	$=$	$\sum_{i=1}^d n_i^X$
$n_{\bullet\bullet}^X$	$=$	$\sum_{i=1}^d n_{ii}^X$
$\dagger\ n_{\bullet\square}^X$	$=$	$\frac{1}{2}\sum_{i=1}^d n_{i\bullet}^X$
$n_{\square(\bullet\bullet)}^X$	$=$	$\sum_{i=1}^d n_{i(\bullet\bullet)}^X$
$n_{(\bullet\bullet)}^X$	$=$	$\sum_{i=1}^d n_{(ii)}^X$
$n_{(\bullet\square)}^X$	$=$	$\frac{1}{2}\sum_{i=1}^d n_{(i\bullet)}^X$
$n_{\bullet\uparrow(\square\square)}^C$	$=$	$\sum_{i=1}^d n_{\bullet(ii)}^C$
$n_{[i\bullet]}^X$	$=$	$n_i^X(n_\bullet^X-n_i^X)$
$n_{[\bullet\square]}^X$	$=$	$\frac{1}{2}\sum_{i=1}^d n_{i\bullet}^X$

Figure 12: Computation of counters for the different component compositions



Case	T_1	T_2		**CC→C**		**GG→G**
$\alpha\alpha$			1,2,5,6	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i\uparrow(\bullet\bullet)}^{C_2}$	3:4	$\sum_{i=1}^d n_{(ii)}^{G_1}\left(n_{(\bullet\bullet)}^{G_2}-n_{(ii)}^{G_2}\right)$
			3,4	$\sum_{i=1}^d \binom{n_i^{C_1}}{2}\left(n_{(\bullet\bullet)}^{C_2}-n_{(ii)}^{C_2}\right)$		
$\beta\alpha$			1,2	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i\uparrow(\bullet\square)}^{C_2}$	3:4	$\sigma\left[\sum_{i=1}^d n_{(ii)}^{G'}\left(n_{(\bullet\square)}^{G''}-n_{(i\bullet)}^{G''}\right)\right]$
			3	$\sum_{i=1}^d \binom{n_i^{C_1}}{2}\left(n_{(\bullet\square)}^{C_2}-n_{(i\bullet)}^{C_2}\right)$		
			4	$\sum_{i=1}^d \left(n_{[\bullet\square]}^{C_1}-n_{[i\bullet]}^{C_1}\right)n_{(ii)}^{C_2}$		
			5,6	$\sum_{k=1}^d n_k^{C_1}\left(n_{\square\uparrow(\bullet\bullet)}^{C_2}-n_{k\uparrow(\bullet\bullet)}^{C_2}-n_{\bullet\uparrow(kk)}^{C_2}\right)$		
$\beta\beta$			1,2	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i\uparrow\bullet\square}^{C_2}$	3:45	$\sigma\left[\sum_{i=1}^d n_{(ii)}^{G'}\left(n_{\bullet\square}^{G''}-n_{i\bullet}^{G''}\right)\right]$
			3	$\sum_{i=1}^d \binom{n_i^{C_1}}{2}\left(n_{\bullet\square}^{C_2}-n_{i\bullet}^{C_2}\right)$	34:5,35:4	$\sigma\left[\sum_{k=1}^d n_k^{G'}\left(n_{\square(\bullet\bullet)}^{G''}-n_{k(\bullet\bullet)}^{G''}-n_{\bullet(kk)}^{G''}\right)\right]$
			4,5	$\sum_{k=1}^d n_k^{C_1}\left(n_{\square(\bullet\bullet)}^{C_2}-n_{k(\bullet\bullet)}^{C_2}-n_{\bullet(kk)}^{C_2}\right)$		
			1,2	$\sum_{k=1}^d n_k^{C_1}\left(n_{\square\uparrow\bullet\bullet}^{C_2}-n_{k\uparrow\bullet\bullet}^{C_2}-n_{\bullet\uparrow kk}^{C_2}\right)$	3:45	$\sigma\left[\sum_{i=1}^d n_{ii}^{G'}\left(n_{(\bullet\square)}^{G''}-n_{(i\bullet)}^{G''}\right)\right]$
			3	$\sum_{i=1}^d \left(n_{[\bullet\square]}^{C_1}-n_{[i\bullet]}^{C_1}\right)n_{ii}^{C_2}$	34:5,35:4	$\sigma\left[\sum_{i=1}^d n_i^{G'}\cdot n_{i(\bullet\square)}^{G''}\right]$
			4,5	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i(\bullet\square)}^{C_2}$		
$\gamma\alpha$			1,2	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i\uparrow(0\bullet)}^{C_2}$	3:4	$\sigma\left[\sum_{i=1}^d n_{(ii)}^{G'}\left(n_{(0\bullet)}^{G''}-n_{(0i)}^{G''}\right)\right]$
			3:45	$\sum_{i=1}^d \binom{n_i^{C_1}}{2}\left(n_{(0\bullet)}^{C_2}-n_{(0i)}^{C_2}\right)$		
			4	$\sum_{i=1}^d n_0^{C_1}\left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{(ii)}^{C_2}$		
			5	$\sum_{i=1}^d \left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{0\uparrow(ii)}^{C_2}$		
			6	$\sum_{i=1}^d n_0^{C_1}\cdot n_{\bullet\uparrow(ii)}^{C_2}$		
$\gamma\beta$			3	$\sum_{i=1}^d \binom{n_i^{C_1}}{2}\left(n_{0\bullet}^{C_2}-n_{0i}^{C_2}\right)$	3:45	$\sigma\left[\sum_{i=1}^d n_{(ii)}^{G'}\left(n_{0\bullet}^{G''}-n_{0i}^{G''}\right)\right]$
			1,2	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i\uparrow0\bullet}^{C_2}$	34:5	$\sigma\left[n_0^{G'}\cdot n_{\square(\bullet\bullet)}^{G''}\right]$
			5	$\sum_{i=1}^d n_0^{C_1}\cdot n_{\bullet(ii)}^{C_2}$	35:4	$\sigma\left[\sum_{i=1}^d \left(n_\bullet^{G'}-n_i^{G'}\right)n_{0(ii)}^{G''}\right]$
			4	$\sum_{i=1}^d \left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{0(ii)}^{C_2}$		
			1	$\sum_{i=1}^d \left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{0\uparrow ii}^{C_2}$	3:45	$\sigma\left[\sum_{i=1}^d n_{ii}^{G'}\left(n_{(0\bullet)}^{G''}-n_{(0i)}^{G''}\right)\right]$
			2	$\sum_{i=1}^d n_0^{C_1}\cdot n_{\bullet\uparrow ii}^{C_2}$	34:5,35:4	$\sigma\left[\sum_{i=1}^d n_i^{G'}\cdot n_{i(0\bullet)}^{G''}\right]$
			3	$\sum_{i=1}^d n_0^{C_1}\left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{ii}^{C_2}$		
			4,5	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i(0\bullet)}^{C_2}$		
$\gamma\gamma$			1,2	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i\uparrow\bullet\uparrow0}^{C_2}$	5:6	$\sigma\left[\sum_{i=1}^d n_0^{G'}\cdot n_{(\bullet(ii))}^{G''}\right]$
			3	$\sum_{i=1}^d \binom{n_i^{C_1}}{2}\left(n_{\bullet\uparrow0}^{C_2}-n_{i\uparrow0}^{C_2}\right)$		
			4	$\sum_{i=1}^d \left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{(ii)\uparrow0}^{C_2}$		
			5	$\sum_{i=1}^d n_{[\bullet(ii)]}^{C_1}\cdot n_0^{C_2}$		
			6	$\sum_{i=1}^d n_0^{C_1}\cdot n_{(\bullet(ii))}^{C_2}$		
			1,2	$\sum_{i=1}^d n_i^{C_1}\cdot n_{i\uparrow0\uparrow\bullet}^{C_2}$	5:6	$\sigma\left[\sum_{i=1}^d \left(n_\bullet^{G'}-n_i^{G'}\right)n_{(0(ii))}^{G''}\right]$
			3	$\sum_{i=1}^d \binom{n_i^{C_1}}{2}\left(n_{0\uparrow\bullet}^{C_2}-n_{0\uparrow i}^{C_2}\right)$		
			4	$\sum_{i=1}^d n_0^{C_1}\cdot n_{(ii)\uparrow\bullet}^{C_2}$		
			5	$\sum_{i=1}^d n_{[0(ii)]}^{C_1}\left(n_\bullet^{C_2}-n_i^{C_2}\right)$		
			6	$\sum_{i=1}^d \left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{(0(ii))}^{C_2}$		
			1	$\sum_{i=1}^d n_0^{C_1}\cdot n_{\bullet\uparrow i\uparrow i}^{C_2}$	5:6	$\sigma\left[\sum_{i=1}^d n_i^{G'}\cdot n_{(i(0\bullet))}^{G''}\right]$
			2	$\sum_{i=1}^d \left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{0\uparrow i\uparrow i}^{C_2}$		
			3	$\sum_{i=1}^d n_0^{C_1}\left(n_\bullet^{C_1}-n_i^{C_1}\right)n_{i\uparrow i}^{C_2}$		
			4	$\sum_{i=1}^d n_i^{C_1}\cdot n_{(0\bullet)\uparrow i}^{C_2}$		
			5	$\sum_{i=1}^d n_{[i(0\bullet)]}^{C_1}\cdot n_i^{C_2}$		
			6	$\sum_{i=1}^d n_i^{C_1}\cdot n_{(i(0\bullet))}^{C_2}$		

Figure 13: Contribution to the number of shared resolved quartets for component compositions

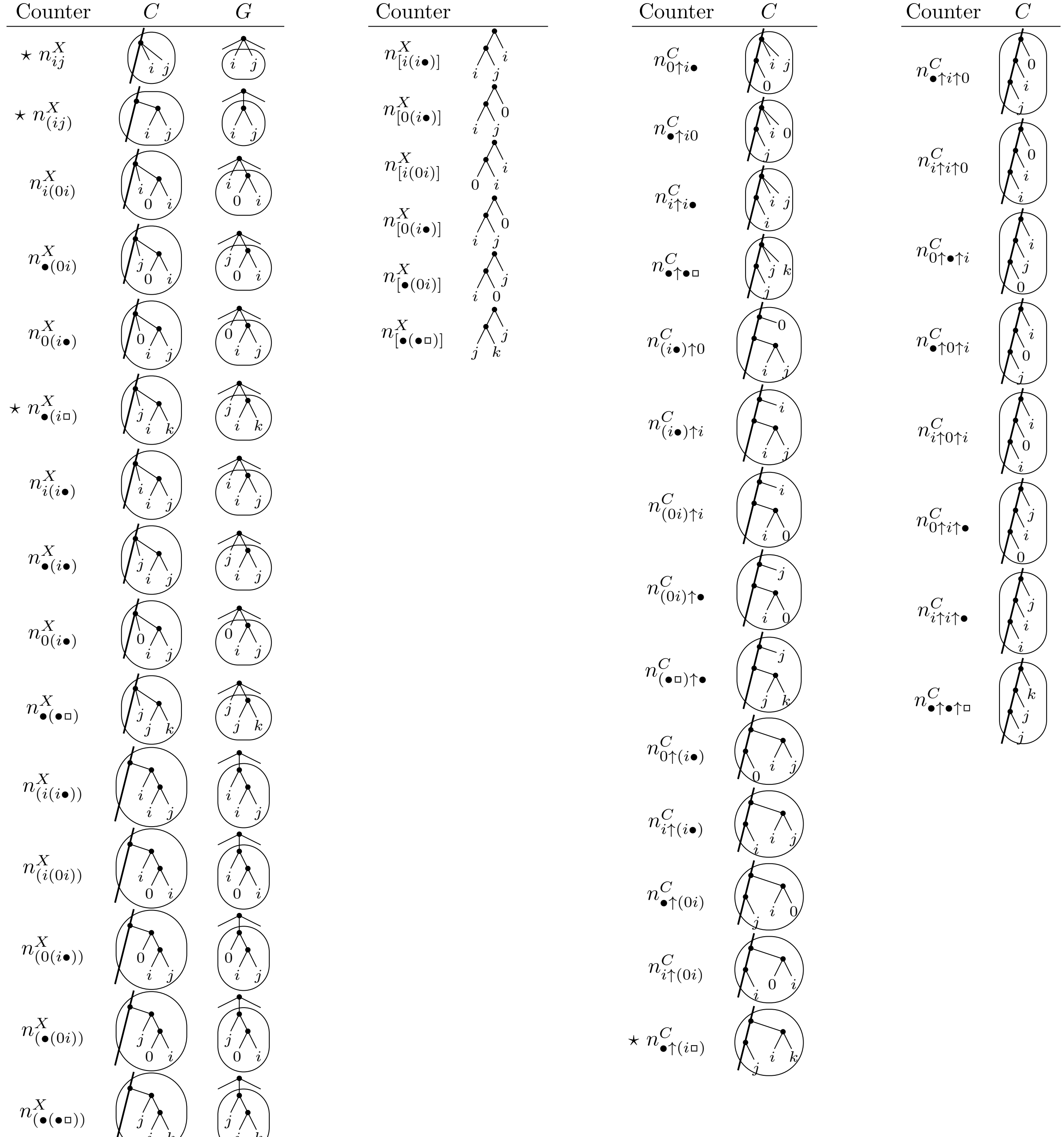
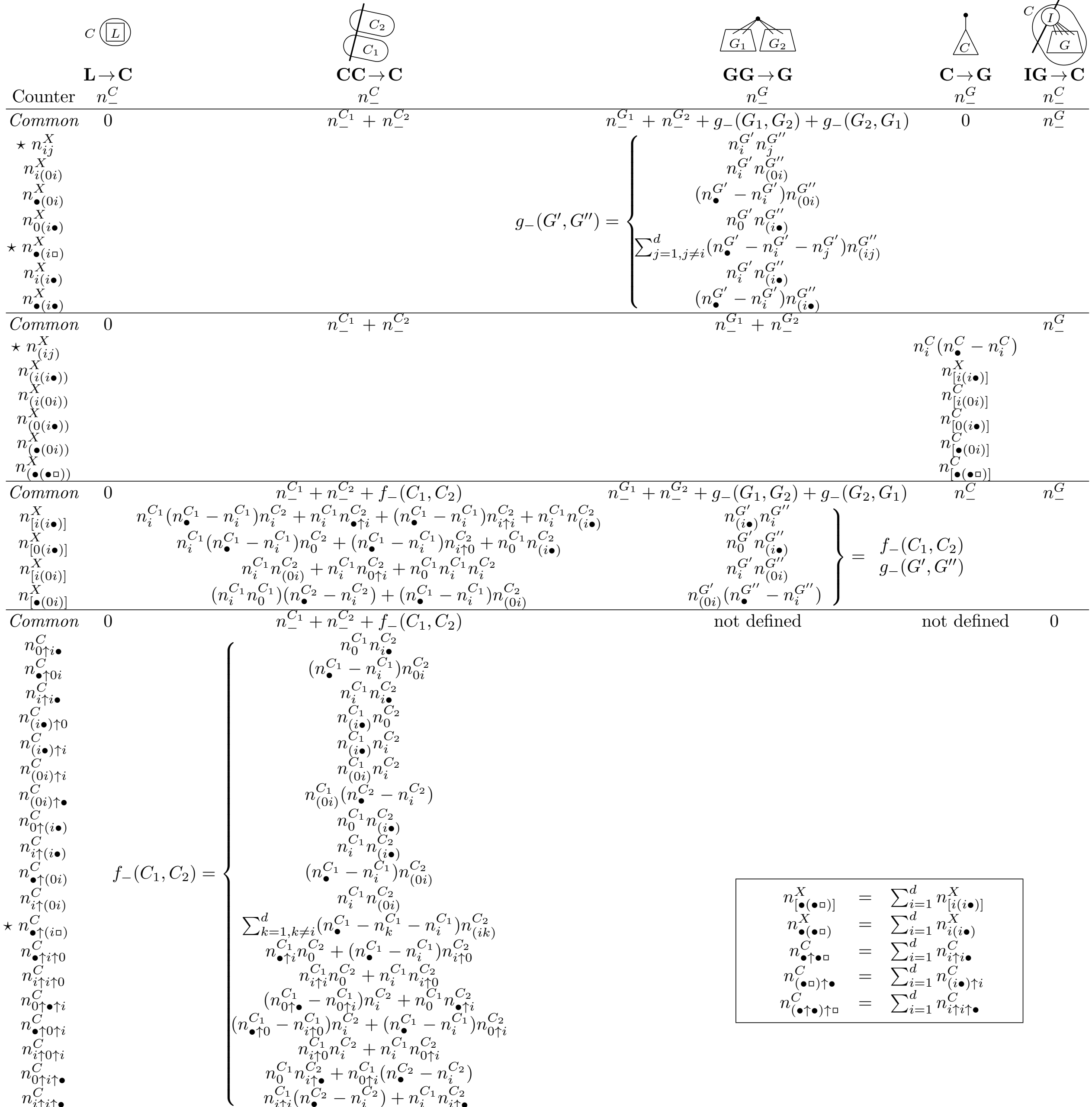


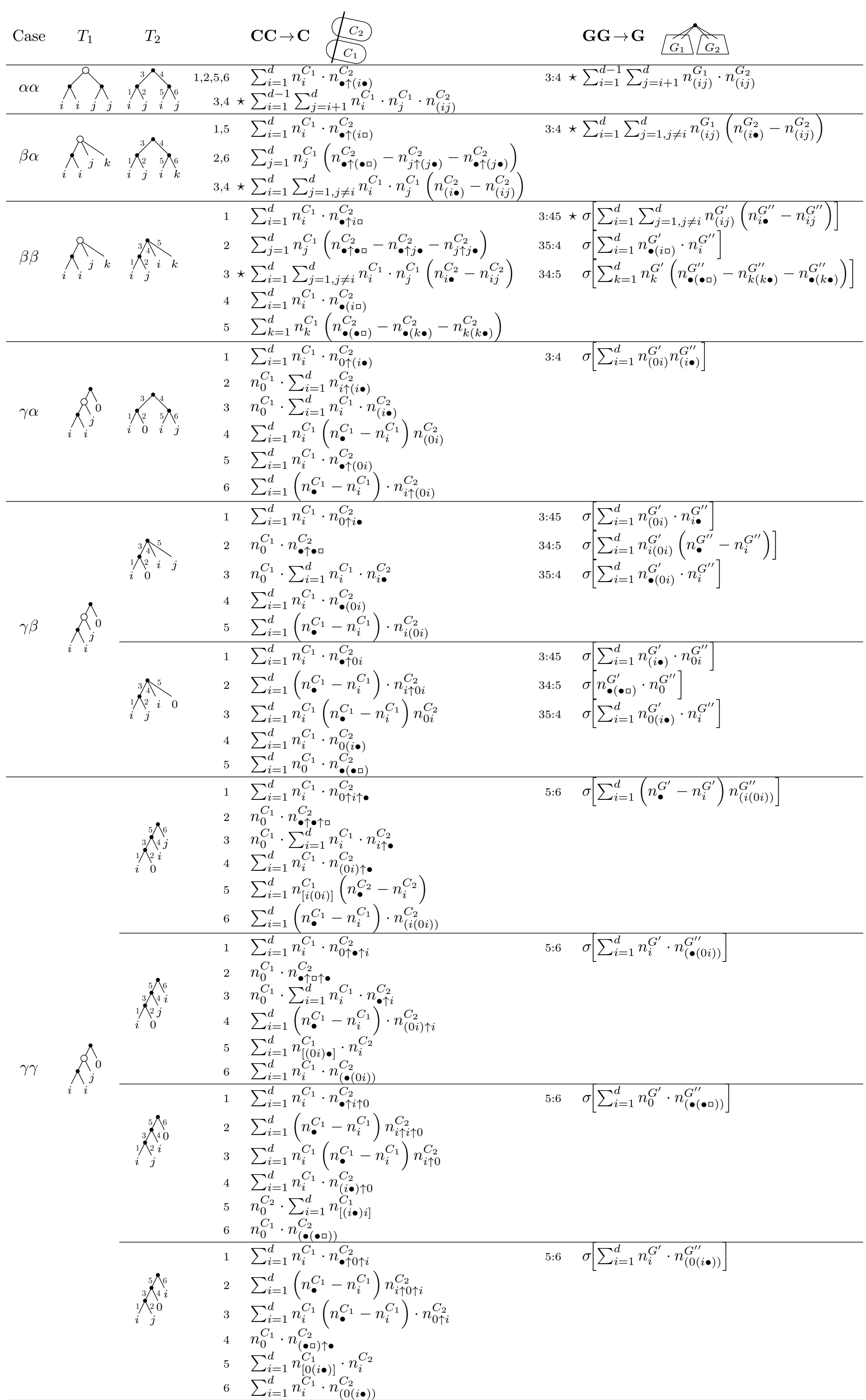
Figure 14: The additional counters used to compute the number of disagreeing resolved quartets



Counter	$\mathbf{L}\to\mathbf{C}$ n_-^C	$\mathbf{CC}\to\mathbf{C}$ n_-^C	$\mathbf{GG}\to\mathbf{G}$ n_-^G	$\mathbf{C}\to\mathbf{G}$ n_-^G	$\mathbf{IG}\to\mathbf{C}$ n_-^C
Common	0	$n_-^{C_1}+n_-^{C_2}$	$n_-^{G_1}+n_-^{G_2}+g_-(G_1,G_2)+g_-(G_2,G_1)$	0	n_-^G
$\star\ n^X_{ij}$			$g_-(G',G'') = \{$ $n_i^{G'}n_j^{G''}$		
$n^X_{i(0i)}$			$n_i^{G'}n_{(0i)}^{G''}$		
$n^X_{\bullet(0i)}$			$(n_\bullet^{G'}-n_i^{G'})n_{(0i)}^{G''}$		
$n^X_{0(i\bullet)}$			$n_0^{G'}n_{(i\bullet)}^{G''}$		
$\star\ n^X_{\bullet(i\square)}$			$\sum_{j=1,j\neq i}^d(n_\bullet^{G'}-n_i^{G'}-n_j^{G'})n_{(ij)}^{G''}$		
$n^X_{i(i\bullet)}$			$n_i^{G'}n_{(i\bullet)}^{G''}$		
$n^X_{\bullet(i\bullet)}$			$(n_\bullet^{G'}-n_i^{G'})n_{(i\bullet)}^{G''}$		
Common	0	$n_-^{C_1}+n_-^{C_2}$	$n_-^{G_1}+n_-^{G_2}$		n_-^G
$\star\ n^X_{(ij)}$				$n_i^C(n_\bullet^C-n_i^C)$	
$n^X_{(i(i\bullet))}$				$n^X_{[i(i\bullet)]}$	
$n^X_{(i(0i))}$				$n^C_{[i(0i)]}$	
$n^X_{(0(i\bullet))}$				$n^C_{[0(i\bullet)]}$	
$n^X_{(\bullet(0i))}$				$n^C_{[\bullet(0i)]}$	
$n^X_{(\bullet(\bullet\square))}$				$n^C_{[\bullet(\bullet\square)]}$	
Common	0	$n_-^{C_1}+n_-^{C_2}+f_-(C_1,C_2)$	$n_-^{G_1}+n_-^{G_2}+g_-(G_1,G_2)+g_-(G_2,G_1)$	n_-^C	n_-^G
$n^X_{[i(i\bullet)]}$		$n_i^{C_1}(n_\bullet^{C_1}-n_i^{C_1})n_i^{C_2}+n_i^{C_1}n_{\bullet\uparrow i}^{C_2}+(n_\bullet^{C_1}-n_i^{C_1})n_{i\uparrow i}^{C_2}+n_i^{C_1}n_{(i\bullet)}^{C_2}$	$n_{(i\bullet)}^{G'}n_i^{G''}$		
$n^X_{[0(i\bullet)]}$		$n_i^{C_1}(n_\bullet^{C_1}-n_i^{C_1})n_0^{C_2}+(n_\bullet^{C_1}-n_i^{C_1})n_{i\uparrow 0}^{C_2}+n_0^{C_1}n_{(i\bullet)}^{C_2}$	$n_0^{G'}n_{(i\bullet)}^{G''}$	$\} = f_-(C_1,C_2)$, $g_-(G',G'')$	
$n^X_{[i(0i)]}$		$n_i^{C_1}n_{(0i)}^{C_2}+n_i^{C_1}n_{0\uparrow i}^{C_2}+n_0^{C_1}n_i^{C_1}n_i^{C_2}$	$n_i^{G'}n_{(0i)}^{G''}$		
$n^X_{[\bullet(0i)]}$		$(n_i^{C_1}n_0^{C_1})(n_\bullet^{C_2}-n_i^{C_2})+(n_\bullet^{C_1}-n_i^{C_1})n_{(0i)}^{C_2}$	$n_{(0i)}^{G'}(n_\bullet^{G''}-n_i^{G''})$		
Common	0	$n_-^{C_1}+n_-^{C_2}+f_-(C_1,C_2)$	not defined	not defined	0
$n^C_{0\uparrow i\bullet}$		$n_0^{C_1}n_{i\bullet}^{C_2}$			
$n^C_{\bullet\uparrow 0i}$		$(n_\bullet^{C_1}-n_i^{C_1})n_{0i}^{C_2}$			
$n^C_{i\uparrow i\bullet}$		$n_i^{C_1}n_{i\bullet}^{C_2}$			
$n^C_{(i\bullet)\uparrow 0}$		$n_{(i\bullet)}^{C_1}n_0^{C_2}$			
$n^C_{(i\bullet)\uparrow i}$		$n_{(i\bullet)}^{C_1}n_i^{C_2}$			
$n^C_{(0i)\uparrow i}$		$n_{(0i)}^{C_1}n_i^{C_2}$			
$n^C_{(0i)\uparrow\bullet}$		$n_{(0i)}^{C_1}(n_\bullet^{C_2}-n_i^{C_2})$			
$n^C_{0\uparrow(i\bullet)}$		$n_0^{C_1}n_{(i\bullet)}^{C_2}$			
$n^C_{i\uparrow(i\bullet)}$		$n_i^{C_1}n_{(i\bullet)}^{C_2}$			
$n^C_{\bullet\uparrow(0i)}$		$f_-(C_1,C_2) = \{$ $(n_\bullet^{C_1}-n_i^{C_1})n_{(0i)}^{C_2}$			
$n^C_{i\uparrow(0i)}$		$n_i^{C_1}n_{(0i)}^{C_2}$			
$\star\ n^C_{\bullet\uparrow(i\square)}$		$\sum_{k=1,k\neq i}^d(n_\bullet^{C_1}-n_k^{C_1}-n_i^{C_1})n_{(ik)}^{C_2}$			
$n^C_{\bullet\uparrow i\uparrow 0}$		$n_{\bullet\uparrow i}^{C_1}n_0^{C_2}+(n_\bullet^{C_1}-n_i^{C_1})n_{i\uparrow 0}^{C_2}$			
$n^C_{i\uparrow i\uparrow 0}$		$n_{i\uparrow i}^{C_1}n_0^{C_2}+n_i^{C_1}n_{i\uparrow 0}^{C_2}$			
$n^C_{0\uparrow\bullet\uparrow i}$		$(n_{0\uparrow\bullet}^{C_1}-n_{0\uparrow i}^{C_1})n_i^{C_2}+n_0^{C_1}n_{\bullet\uparrow i}^{C_2}$			
$n^C_{\bullet\uparrow 0\uparrow i}$		$(n_{\bullet\uparrow 0}^{C_1}-n_{i\uparrow 0}^{C_1})n_i^{C_2}+(n_\bullet^{C_1}-n_i^{C_1})n_{0\uparrow i}^{C_2}$			
$n^C_{i\uparrow 0\uparrow i}$		$n_{i\uparrow 0}^{C_1}n_i^{C_2}+n_i^{C_1}n_{0\uparrow i}^{C_2}$			
$n^C_{0\uparrow i\uparrow\bullet}$		$n_0^{C_1}n_{i\uparrow\bullet}^{C_2}+n_{0\uparrow i}^{C_1}(n_\bullet^{C_2}-n_i^{C_2})$			
$n^C_{i\uparrow i\uparrow\bullet}$		$n_{i\uparrow i}^{C_1}(n_\bullet^{C_2}-n_i^{C_2})+n_i^{C_1}n_{i\uparrow\bullet}^{C_2}$			

$$n^X_{[\bullet(\bullet\square)]} = \sum_{i=1}^d n^X_{[i(i\bullet)]}$$
$$n^X_{\bullet(\bullet\square)} = \sum_{i=1}^d n^X_{i(i\bullet)}$$
$$n^C_{\bullet\uparrow\bullet\square} = \sum_{i=1}^d n^C_{i\uparrow i\bullet}$$
$$n^C_{(\bullet\square)\uparrow\bullet} = \sum_{i=1}^d n^C_{(i\bullet)\uparrow i}$$
$$n^C_{(\bullet\uparrow\bullet)\uparrow\square} = \sum_{i=1}^d n^C_{i\uparrow i\uparrow\bullet}$$

Figure 15: Computation of counters for the different component compositions



Case	T_1	T_2		$\mathbf{CC}\to\mathbf{C}$		$\mathbf{GG}\to\mathbf{G}$
$\alpha\alpha$			1,2,5,6	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet\uparrow(i\bullet)}^{C_2}$	3:4	$\star \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} n_{(ij)}^{G_1} \cdot n_{(ij)}^{G_2}$
			3,4	$\star \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} n_i^{C_1} \cdot n_j^{C_1} \cdot n_{(ij)}^{C_2}$		
$\beta\alpha$			1,5	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet\uparrow(i\square)}^{C_2}$	3:4	$\star \sum_{i=1}^{d} \sum_{j=1,j\neq i}^{d} n_{(ij)}^{G_1} \left(n_{(i\bullet)}^{G_2} - n_{(ij)}^{G_2}\right)$
			2,6	$\sum_{j=1}^{d} n_j^{C_1} \left(n_{\bullet\uparrow(\bullet\square)}^{C_2} - n_{j\uparrow(j\bullet)}^{C_2} - n_{\bullet\uparrow(j\bullet)}^{C_2}\right)$		
			3,4	$\star \sum_{i=1}^{d} \sum_{j=1,j\neq i}^{d} n_i^{C_1} \cdot n_j^{C_1} \left(n_{(i\bullet)}^{C_2} - n_{(ij)}^{C_2}\right)$		
$\beta\beta$			1	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet\uparrow i\square}^{C_2}$	3:45	$\star\ \sigma\left[\sum_{i=1}^{d} \sum_{j=1,j\neq i}^{d} n_{(ij)}^{G'} \left(n_{i\bullet}^{G''} - n_{ij}^{G''}\right)\right]$
			2	$\sum_{j=1}^{d} n_j^{C_1} \left(n_{\bullet\uparrow\bullet\square}^{C_2} - n_{\bullet\uparrow j\bullet}^{C_2} - n_{j\uparrow j\bullet}^{C_2}\right)$	35:4	$\sigma\left[\sum_{i=1}^{d} n_{\bullet(i\square)}^{G'} \cdot n_i^{G''}\right]$
			3	$\star \sum_{i=1}^{d} \sum_{j=1,j\neq i}^{d} n_i^{C_1} \cdot n_j^{C_1} \left(n_{i\bullet}^{C_2} - n_{ij}^{C_2}\right)$	34:5	$\sigma\left[\sum_{k=1}^{d} n_k^{G'} \left(n_{\bullet(\bullet\square)}^{G''} - n_{k(k\bullet)}^{G''} - n_{\bullet(k\bullet)}^{G''}\right)\right]$
			4	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet(i\square)}^{C_2}$		
			5	$\sum_{k=1}^{d} n_k^{C_1} \left(n_{\bullet(\bullet\square)}^{C_2} - n_{\bullet(k\bullet)}^{C_2} - n_{k(k\bullet)}^{C_2}\right)$		
$\gamma\alpha$			1	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{0\uparrow(i\bullet)}^{C_2}$	3:4	$\sigma\left[\sum_{i=1}^{d} n_{(0i)}^{G'} n_{(i\bullet)}^{G''}\right]$
			2	$n_0^{C_1} \cdot \sum_{i=1}^{d} n_{i\uparrow(i\bullet)}^{C_2}$		
			3	$n_0^{C_1} \cdot \sum_{i=1}^{d} n_i^{C_1} \cdot n_{(i\bullet)}^{C_2}$		
			4	$\sum_{i=1}^{d} n_i^{C_1} \left(n_\bullet^{C_1} - n_i^{C_1}\right) n_{(0i)}^{C_2}$		
			5	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet\uparrow(0i)}^{C_2}$		
			6	$\sum_{i=1}^{d} \left(n_\bullet^{C_1} - n_i^{C_1}\right) \cdot n_{i\uparrow(0i)}^{C_2}$		
$\gamma\beta$			1	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{0\uparrow i\bullet}^{C_2}$	3:45	$\sigma\left[\sum_{i=1}^{d} n_{(0i)}^{G'} \cdot n_{i\bullet}^{G''}\right]$
			2	$n_0^{C_1} \cdot n_{\bullet\uparrow\bullet\square}^{C_2}$	34:5	$\sigma\left[\sum_{i=1}^{d} n_{i(0i)}^{G'} \left(n_\bullet^{G''} - n_i^{G''}\right)\right]$
			3	$n_0^{C_1} \cdot \sum_{i=1}^{d} n_i^{C_1} \cdot n_{i\bullet}^{C_2}$	35:4	$\sigma\left[\sum_{i=1}^{d} n_{\bullet(0i)}^{G'} \cdot n_i^{G''}\right]$
			4	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet(0i)}^{C_2}$		
			5	$\sum_{i=1}^{d} \left(n_\bullet^{C_1} - n_i^{C_1}\right) \cdot n_{i(0i)}^{C_2}$		
			1	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet\uparrow 0i}^{C_2}$	3:45	$\sigma\left[\sum_{i=1}^{d} n_{(i\bullet)}^{G'} \cdot n_{0i}^{G''}\right]$
			2	$\sum_{i=1}^{d} \left(n_\bullet^{C_1} - n_i^{C_1}\right) \cdot n_{i\uparrow 0i}^{C_2}$	34:5	$\sigma\left[n_{\bullet(\bullet\square)}^{G'} \cdot n_0^{G''}\right]$
			3	$\sum_{i=1}^{d} n_i^{C_1} \left(n_\bullet^{C_1} - n_i^{C_1}\right) n_{0i}^{C_2}$	35:4	$\sigma\left[\sum_{i=1}^{d} n_{0(i\bullet)}^{G'} \cdot n_i^{G''}\right]$
			4	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{0(i\bullet)}^{C_2}$		
			5	$\sum_{i=1}^{d} n_0^{C_1} \cdot n_{\bullet(\bullet\square)}^{C_2}$		
$\gamma\gamma$			1	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{0\uparrow i\uparrow\bullet}^{C_2}$	5:6	$\sigma\left[\sum_{i=1}^{d} \left(n_\bullet^{G'} - n_i^{G'}\right) n_{(i(0i))}^{G''}\right]$
			2	$n_0^{C_1} \cdot n_{\bullet\uparrow\bullet\uparrow\square}^{C_2}$		
			3	$n_0^{C_1} \cdot \sum_{i=1}^{d} n_i^{C_1} \cdot n_{i\uparrow\bullet}^{C_2}$		
			4	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{(0i)\uparrow\bullet}^{C_2}$		
			5	$\sum_{i=1}^{d} n_{[i(0i)]}^{C_1} \left(n_\bullet^{C_2} - n_i^{C_2}\right)$		
			6	$\sum_{i=1}^{d} \left(n_\bullet^{C_1} - n_i^{C_1}\right) \cdot n_{(i(0i))}^{C_2}$		
			1	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{0\uparrow\bullet\uparrow i}^{C_2}$	5:6	$\sigma\left[\sum_{i=1}^{d} n_i^{G'} \cdot n_{(\bullet(0i))}^{G''}\right]$
			2	$n_0^{C_1} \cdot n_{\bullet\uparrow\square\uparrow\bullet}^{C_2}$		
			3	$n_0^{C_1} \cdot \sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet\uparrow i}^{C_2}$		
			4	$\sum_{i=1}^{d} \left(n_\bullet^{C_1} - n_i^{C_1}\right) \cdot n_{(0i)\uparrow i}^{C_2}$		
			5	$\sum_{i=1}^{d} n_{[(0i)\bullet]}^{C_1} \cdot n_i^{C_2}$		
			6	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{(\bullet(0i))}^{C_2}$		
			1	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet\uparrow i\uparrow 0}^{C_2}$	5:6	$\sigma\left[\sum_{i=1}^{d} n_0^{G'} \cdot n_{(\bullet(\bullet\square))}^{G''}\right]$
			2	$\sum_{i=1}^{d} \left(n_\bullet^{C_1} - n_i^{C_1}\right) n_{i\uparrow i\uparrow 0}^{C_2}$		
			3	$\sum_{i=1}^{d} n_i^{C_1} \left(n_\bullet^{C_1} - n_i^{C_1}\right) n_{i\uparrow 0}^{C_2}$		
			4	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{(i\bullet)\uparrow 0}^{C_2}$		
			5	$n_0^{C_2} \cdot \sum_{i=1}^{d} n_{[(i\bullet)i]}^{C_1}$		
			6	$n_0^{C_1} \cdot n_{(\bullet(\bullet\square))}^{C_2}$		
			1	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{\bullet\uparrow 0\uparrow i}^{C_2}$	5:6	$\sigma\left[\sum_{i=1}^{d} n_i^{G'} \cdot n_{(0(i\bullet))}^{G''}\right]$
			2	$\sum_{i=1}^{d} \left(n_\bullet^{C_1} - n_i^{C_1}\right) n_{i\uparrow 0\uparrow i}^{C_2}$		
			3	$\sum_{i=1}^{d} n_i^{C_1} \left(n_\bullet^{C_1} - n_i^{C_1}\right) \cdot n_{0\uparrow i}^{C_2}$		
			4	$n_0^{C_1} \cdot n_{(\bullet\square)\uparrow\bullet}^{C_2}$		
			5	$\sum_{i=1}^{d} n_{[0(i\bullet)]}^{C_1} \cdot n_i^{C_2}$		
			6	$\sum_{i=1}^{d} n_i^{C_1} \cdot n_{(0(i\bullet))}^{C_2}$		

Figure 16: Contribution to the number of disagreeing resolved quartets