Reconstruction of Phylogenetic Networks

An algorithm for reconstructing Level-2 Binary Networks based on their distances.

Riche Mol
4713796

Department Name: EEMCS
University Name: TU Delft
Date:

Abstract

Traditionally in phylogenetics, the evolutionary history of organisms is represented in a tree structure. However, tree structures might not give a complete picture of the evolution of certain organisms. To that end, over the past years, more advances networks are considered to represent the evolution of species. Van Iersel, Moulton, and Murakami (2020) proved that a level-2 binary phylogenetic network can be uniquely reconstructed based on the matrix of mulitsets of the distances of the leaves. Using a handful of lemma's each describing the steps of identifying cherries, uncontained leaves and blobs in the network, I created and implemented an algorithm that reconstructs level-2 networks based on a matrix of the multisets of inter-leaf distances. This algorithm is of polynomial time, with respect to D, the maximum of the sizes of the multisets, and n, the size of the matrix containing these multisets. This algorithm is tested on more than 35000 networks with 10 to 24 leaves, resulted in no errors, and does so in a couple of seconds.

1 Introduction

Traditionally, in phylogenetics, evolution of species was represented using trees. However, in the past couple of years, tree model have been stated to paint an image that is a bit too simplistic to properly represent the evolution processes, notably those of small organisms (Bapteste et al., 2013). To that end, to improve our understanding of phylogenetics a shift from the traditional tree structures to more general networks is being made. This requires previous assumpions and results to be revisited, and existing data to be re-evaluated to determine if a tree structure is still applicable to it or a different network type is a better fit. Additionally, models and algorithms have to be renewed and created to better fit this new approach. Ergo, not only new challenges are imposed on biologists, but also on mathematicians, computer scientists and programmers [2].

By nature of evolution, a network describing an evolutionary process is expected to be directed, as it is a chronological process. However, because there are manners to estimate the rooted network from its unrooted variant, for example by using outgroups, and the distance based approach of reconstructing the network, for this paper, and the paper on which this one extends upon, the choice for unrooted, undirected networks is made [1]

In this paper, exclusively unweighted, undirected Binary Level 2 networks are considered, where a vertex has either a single or three incident edges. Anytime a mention of a network is made, it refers to such a network. Within a tree structure, two leaves are connected by a single path. In a binary network, the number of paths between two leaves can be very large, and increases as the size and of the network increases; the upper bound for the number of paths between two leaves is 4^n, where n is the number of leaves in the network. The sizes of the multisets can therefore increase in size at an exponential rate for some data.

For these networks an algorithm is proposed to deconstruct a matrix of multisets of distances to a list of steps which is then used to reconstruct a network whose multisets of inter-leaf distances correspond to the original matrix. The deconstruction mainly consist of identifying certain types of subgraphs, contracting these to a single leaf, and adjusting the required multisets in the matrix to correspond to the newly obtained network. This paper is ordered in such a way that all lemmas introduced by Van Iersel, Moulton, and Murakami (2020) to deconstruct a matrix of multisets are revisited in roughly the same order. Where at each step, additions and/or generalizations of existing lemma's are presented (if any) and a piece of pseudocode for that specific contraction is included. When all the deconstruction steps have been handled, the reconstruction is adressed. However, this part is more brief than the deconstruction, as this process is more straightforward and requires significantly less proof. The whole of the algorithm is proven by proving each individual deconstruction and reconstruction step and then combining them all together. It is known that such an algorithm exists[1], but theory is only useful when it can be applied in practice, hence it is key to implement a functioning algorithm that does just that. Additionally, one should always be looking to improve and extend upon existing theory, and to that end, a handful of lemmas will be presented to improve the speed in which the algorithm will run.

This all leads to the following main theorem of this paper.

Theorem 1.1. *There exists a $O(n^3 \cdot D)$-algorithm that reconstructs a level-2 binary network N based on the multisets of its distances.*

Here, n is the number of leaves in the network, and D is the maximum over the sizes of all multisets of distances. The algorithm runs in theoretical polynomial time with respect to the input. However, the size of D is upperbounded by 4^n, which results in the practical polynomial time increasing exponentially as n increases.

2 Preliminaries

Unweighted, unrooted binary network
Let X be a non-empty finite set. An *unweighted unrooted binary network* N on X is a simple graph with
1. $|X|$ vertices of degree-1 (*leaves*); and
2. all other vertices of degree-3 (*internal vertices*).

As these are the only networks considered in this paper, they are simply referred to as networks.

A cut-edge is an edge that upon removal disconnects the graph, that is, it induces a partition X = Y ∪ Z; an edge incident to a leaf is therefore also a cut-edge, but will be referred to as a *trivial cut-edge.*

Given two leaves x and y, we denote the multisets of distances (sometimes simply referenced to as multisets) for those leaves by $d^N(x,y) = \{d_1^{k_1}, ..., d_n^{k_n}\}$ such that $d_i < d_{i+1}$, where k_i represents the number of paths with length d_i from leaf x to leaf y in N. If k_i equals 1, no superscript is added.

Cherry
Two leaves x and y form a cherry if they share a common neighbour.
Equivalently, x and y form a cherry if and only if $d^N(x,y) = \{2\}$.

Chain
A chain of length $k \geq 1$ is a k-tuple of leaves $(a_1, ..., a_k)$ such that $d_m(a_i, a_{i+1}) = 3$ for all $i \in \{1, 2, ..., k-1\}$.
A chain of lenght 1 will be refered to as a ***Single Leaf***, but is considered a chain.

Endpoint
The *Endpoints* of a Chain $(a_1, a_2, ..., a_k)$, are the leaves a_1 and a_k. I.e. the leaves in the chain with at most 1 other leaf to which it has minimal distance 3. Addtionally, a *Single Leaf* is also considered an endpoint. The set of all endpoints will be denoted by E.

Blob
A *Biconnected Component* (blob) of a network is a maximal 2-connected subgraph with at least three vertices. A *Pendant Blob* is a Blob with exactly 1 non-trivial cut-edge.

In this thesis we distinct between a level-1 and a level-2 blob, but no level higher than that. In a level-k blob, we can remove at most k edges such that the remaining structure is a tree.

Let $C = (c_1, .., c_k)$ and $D = (d_1, ..., d_l)$ be chains. If C has an endpoint with minimal distance 4 to an endpoint of D, then we call C and D *adjacent*.
C and D are *adjacent twice* (or *doubly adjacent*) if $d_m(c_1, d_1) = d_m(c_k, d_l) = 4$ or $d_m(c_1, d_l) = d(c_k, d_1) = 4$, but the paths between the pair of endpoints are different.
Note that a chain with only one leaf can be doubly adjacent to another chain, which on its turn may also have just one leaf.

Isomorphic
A graph G is *isomorphic* to a graph H if there exists a bijection f between the vertex sets of G and H, such that two vertices u and v are adjacent in G if and only if the vertices f(u) and f(v) are adjacent in H.
By extension, if G and H are isomorphic, all multisets of distances for 2 adjacent vertices in G are equal the multisets of the corresponding adjacent vertices in H.

In the deconstruction, we only focus on pendant blobs. Every blob in the network that is not pendant will eventually be a pendant blob after a couple of contractions. A pendant blob can either be level 1 or level 2. A level-1 pendant blob can only have 1 chain, which has a length of at least two (if it contains a chain of length 1, the blob has a pair of parallel edges). A pendant level-2 blob can have up to 4 chains, which will be denoted by C^k, C^l, C^m,and C^n. If B is a pendant level-2 blob with 4 chains, C^k and C^l are those that are adjacent twice to each other. C^m and C^n are those that have an endpoint whose minimal distance to any arbitrary leaf not in the blob is minimal. Switching C^k and C^l results in an isomorphic network, as does switching C^m and C^n. In all cases where only one of either k or l is present in the chain, including C^k will be prioritized over the C^l notation-wise, the same with prioritizing including C^m over C^n. Additionally, the chains are ordered such that the minimal distance from C^k_1 to both C^l_1 and C^m_1 is 4 (if they exist in the blob), and the such that C^n_1 has to lowest minimal distance to any leaf not in B.

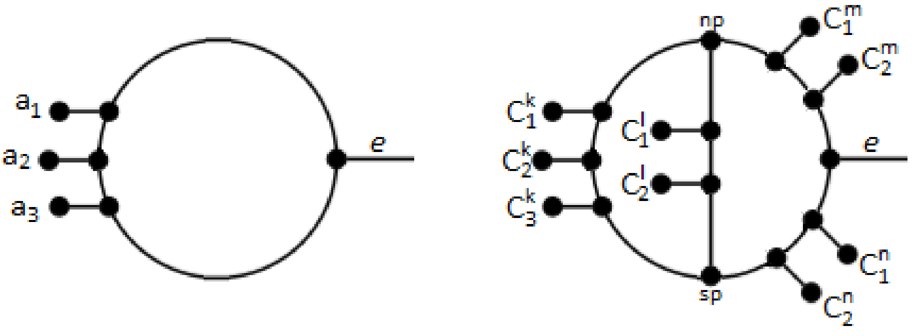


Figure 1: Left: Pendant Level-1 Blob, Right: Pendant Level-2 Blob

Uncontained Leaf
An *uncontained leaf* is a leaf whose neighbour is not a vertex of a blob, but is not contained in a cherry.
Uncontained leaves may be part of some chain with multiple leaves.
If there are no uncontained leaves in the network, then any two chains that are adjacent (twice) belong to the same blob.
At each step in the deconstruction, except for uncontained leaves, a pendant blob or cherry is contracted.Contracting a pendant blob or leaf means that the structure is replaced by a leaf. For a pendant blob, the multisets of distances from all leaves in the network not in that blob to the new leaf, is exactly the same as the multiset of distances to the non-trivial cut-vertex in that blob (although this multiset is not included in the original matrix as it is not a leaf). Because the leaves in

the network are represented by their label in the starting matrix, the size of the matrix is not adjusted upon contractions. That is, for a network on n leaves; with an n × n matrix, the matrix will keep its size throughout the whole deconstruction. Because n is relatively small in the networks the algorithm is used on, this does not result in a significantly larger running time. The index in the new matrix of the leaf obtained from the contraction corresponds to the index in the matrix of a leaf that was in the blob. So, within the algorithm, no new leaf is created.

Reconstructable We call a network N reconstructable from its multisets of distances if it is the only network with those exact multisets.

Theorem 2.1 (Thm. 5.1 of [1]). *Level-2 networks are reconstructible from their multiset of distances.*

The most important result presented in [1]. By this theorem, it is known that a working algorithm for the deconstruction can be created, and moreover, said algorithm works when implementing given lemma's used to prove the theorem.

As Van Iersel, Moulton, and Murakami have proposed a set of Lemma's and Theorems conveying all steps neccessary in the deconstruction of identifying a network from a matrix of the multisets of the leaves, a large portion of these will be used in the algorithm.

By strictly implementing the Lemma's proposed in [1], networks will be contracted properly. However, it turns out that the conditions of some of these lemma's can be simplified, i.e. less steps have to be done in the deconstruction process. All lemma's that require certains distance properties to be fullfilled from each leaf in the network to each other leaf in the network are still valid if only these distance properties are satisfied from each endpoint of each chain to each endpoint of each other chain. Which, as it is likely that chains exists with 3 or more leaves, will result in a shorter practical running time. (Although the theoretical running time is unaffected by only considering the endpoints lemma's.)

Lemma 2.2. *Let a be a leaf.* $d_m(a,x) \geq k \quad \forall x \in X - \{a\}$ *if and only if* $d_m(a,c) \geq k$ *for all c in the set of all endpoints of all chains E-*$\{a\}$.

Proof. Let a be a leaf in X and C = $\{c_1, c_2, ..., c_m\}$ be a chain in X-$\{a\}$, with $\{u_1, u_2, ..., u_m\}$ their respective adjacent vertices. Let c_i be a leaf of C that is not an endpoint of C. Then the shortest path from c_i to a traverses u_i and either u_1 or u_m. Therefore, the path from c_i to a is larger then the $\min(d_m(c_1, a), d_m(c_m, a))$ If both $d_m(c_1, a)$ and $d_m(c_m, a)$ are larger than k, then so is $d_m(c_i, a)$. Because all leaves in X-$\{a\}$ are either in chains or single leaves, if all distances from the endpoints of the chains and all single leaves to leaf a are larger than k, then the distance from all leaves in X-$\{a\}$ to a is larger than k. □

Lemma 2.3. *Let z be a single leaf.* $d_m(x,z) + d_m(y,z) - d_m(x,y) \geq k$ *for any two leaves x,y* $\in X$ - $\{z\}$ *if and only if* $d_m(z,c) + d_m(z,d) - d_m(c,d) \geq k$ *for c,d* $\in E$ - $\{z\}$.

Proof. Let x, y, and z be leaves, c be the endpoint of the chain containing x, whose adjacent vertex is traversed on the shortest path from x to z, and d be the endpoint of the chain containing y, whose adjacent vertex is traversed on the shortest path from y to z.
$d_m(x,z) + d_m(y,z) - d_m(x,y) =$
$d_m(x,c) + d_m(c,z) - 2 + d_m(y,d) + d_m(d,z) - 2 - d_m(x,y) =$
$d_m(c,z) + d_m(d,z) - d_m(c,d) + d_m(c,d) + d_m(x,c) + d_m(y,d) - d_m(x,y) - 4 \geq$
$k + d_m(c,d) + d_m(x,c) + d_m(y,d) - d_m(x,y) - 4 \qquad (1)$
Now if the shortest path from x to y traverses the vertices adjacent to c and d (denoted by c_v and d_v), then:
$k + d_m(c,d) + d_m(x,c) + d_m(y,d) - d_m(x,y) - 4 =$
$k + d_m(c_v,d_v) + 2 + d_m(x,c_v) + 1 + d_m(y,d_v) + 1 - d_m(x,c_v) - d_m(c_v,d_v) - d_m(d_v,y) - 4$

$k + 4 - 4 = k$, as required.
If the shortest path from x to y does not traverse either c_v, or d_v, then denote those endpoints by c' and d'. Now, $d_m(c,d) + d_m(x,c) + d_m(y,d) \geq d_m(c',d') + d_m(x,c') + d_m(y,d')$, and the result follows by the same steps when substituting this into (1). $\square$

3 Split

When contracting a pendant blob or a cherry to a leaf, the multisets of distances of some leaf in that blob or cherry needs to be updated to correspond to that of the leaf replacing the structure. This new leaf has different multisets than any leaf in the cherry or the pendant blob.
For leaves this is more straightforward than for pendant blobs, and will be discussed in section 4.2. For pendant blobs this requires some more effort, and to that extend, the split function will be introduced.

Define $A \oplus B$ as the Minkowski sum between multisets of integers A and B:
$A \oplus B = \{a + b | a \in A, b \in B\}$

For the split function, it is required that $A \oplus B$ is sorted from smallest to largest.

For example:
$A = \{2, 3, 5\}$
$B = \{1, 2, 3, 4\}$
$A \oplus B = \{2+1, 2+2, 2+3, 2+4, 3+1, 3+2, 3+3, 3+4, 5+1, 5+2, 5+3, 5+4\} = \{3, 4, 5, 6, 4, 5, 6, 7, 6, 7, 8, 9\} = \{3, 4^2, 5^2, 6^3, 7^2, 8, 9\}$

If #A = m and #B = n, then #(A $\oplus$ B) = $m \cdot n$

Now, let A and B be ordered multisets of integers, such that $|B|$ divides $|A|$. Next $\ominus$ is denfined such that:
$(A \oplus B) \ominus B = A$
$(A \oplus B) \ominus A = B$

In most cases we will consider, $(A \oplus B)$ and A are known, and we are required to find B. To that end, $(A \oplus B)$ will be denoted as C.

Lemma 3.1. *Let x be a leaf in a pendant blob B, and b be the leaf that results from contracting B. Then for all $a \in X - B$: $d^N(a, b) = d^N(a, x) \ominus d^N(b, x)$*

Proof. Let x be a leaf whose adjacent vertex is in a pendant blob, and let z be the non-trivial cut-vertex of B. When contracting B to a single leaf b, the non-trivial cut-edge of B still exists the the contracted network, but is now connected to the leaf b. Hence, all paths from any leaf $a \in X - B$ to b are exactly the same as from a to z. Take $a \in X - B$. Suppose there are m different paths from a to z, with distance multiset $\{a_1, .., a_m\}$, and n different paths from z to x, with distance multiset $\{b_1, .., b_n\}$. Every path from a to x traverses z, and therefore, every path from a to z can be extended to a path from a to x in n possible ways. So there are exactly $m \cdot n$ paths from a to x, with corresponding distance multiset $\{a_1 + b_1, ...a_1 + b_n, a_2, b_1, ..., a_2 + b_n, ..., a_m + b_n\}$. However, as previously stated, all paths from a to z are equal to those from a to b. Therefore $d^N(a, b) \oplus d^N(b, x) = d^N(a, x)$, and $d^N(a, b) = d^N(a, x) \ominus d^N(b, x)$. □

Split

```
input  : C: the orginal (sorted) multiset from leaf x to leaf y,
         A: the sorted set used to partition into 4 equal sized sets.
output: B: the updated multiset from leaf x to leaf y'
1 B = [ ]
2 while $\#B$ != $\frac{\#C}{\#A}$ do
3     b = $C_0 - A_0$
4     $B = B \cup b$
5     for $a \in A$ do
6         C = C - {b+a}
7     end
8 end
9 return B
```

Lemma 3.2. *Given an input of an ordered multi set C with size $m \cdot n$ and an ordered set A of size m, Split returns the set B of size n such that $A \oplus B = C$, if such a set exists, and does so in running time O(D), proportional to the maximal size of the input set C.*

Proof. Let $C = \{c_1, c_2, ..., c_{mn}\}$ be an ordered multiset of size m · n, and $A = \{a_1, ..., a_n\}$ be an ordered multiset of size m. Assume there exists a multiset $B = \{b_1, ..., b_m\}$ of size n such that $A \oplus B = C$. Moreover, let B^* be an empty multiset. As all multisets are ordered from small to large, $c_1 = a_1 + b_1$ Therefore, $b_1 = c_1 - a_1$, and it should be added to B^*, and $\{b_1 + a_1$, ..., $b_1 + a_m$ $\}$ can be removed from C.
The resulting multiset C' consists of the elements $\{a_1 + b_2, a_2 + b_2, ..., a_m + b_2, a_1 + b_3, a_2 + b_3, ..., a_m + b_3, ..., a_1 + b_n, a_2 + b_n, ..., a_m + b_n\}$, which corresponds to $A \oplus (B \backslash \{b_1\})$
Now, the smallest element of C' is $a_1 + b_2$, therefore b_2 can be obtained by substracting a_1 from C'_1, and the elements $\{b_2 + a_1, b_2 + a_2, ...b_2 + a_m\}$ will be removed from C', such that the resulting multiset C" equals $A \oplus (B \backslash \{b_1 \cup b_2\})$. This sequence can be applied for a total of n times, such that eventually the returned multiset B^* equals $\{b_1, b_2, ..., b_n\} = B$.
All individual steps of the Split Function are of constant time. All elements in the input multiset are removed from C and a fraction of them are added to B.
Therefore the running time of applying the Split Function on multiset of distances is O(D). □

4 Deconstruction

The aim is to uniquely recreate the network of a multiset matrix using only that matrix. This is done by iteratively checking the matrix for cherries, uncontained leaves and pendant blobs, storing these in a list and adjusting the matrix to correspond to the network that results from the contraction. Then the network is recreated based on the steps in the list and for this, the matrix is not longer used.

The deconstruction is done in the following order:
1.) Contract all cherries in the network.
2.) If it is the first iteration; construct all chains, else: skip
3.) Check for end conditions.
4.) Contract level 1 blobs and single chain level 2 blobs
5.) Contract level 2 kl00-blobs
6.) Remove uncontained leaves
7.) Contract level 2 1000-blobs
8.) Contract level 2 klmn-blobs

If the matrix is adjusted at step 4, 6, 7, or 8, return to step 1 after the step has completed. This is done because contracting a blob can result in a new cherry or uncontained leaf. Cherries can be recognized and contracted fast, and uncontained leaves cause errors in recognizing pendant blobs.
Each step contracts all pendant blobs of its type, then if all are contracted, the chains are adjusted, and then it starts again at step 1.
If a cherry or blob is contracted to a single leaf, that leaf is given the label of a leaf in that cherry or blob. The multisets from and to all others are set to $\{0\}$. Although the size of the matrix does not change, these indices are skipped in future programs, as they do not iterate over all leaves in M, but over all leaves in C, the set of all chains and single leaves, which is updated after each step in the deconstruction.

4.1 Step Storage

As each deconstruction step of the distance matrix corresponds to a step in the construction of the network, it is important to keep track of each step in a structural way.
To that end, a list will be created, in which each time a blob or cherry is contracted or an uncontained leaf or chain is removed, an entry is added that uniquely represents that step.

For blobs, such step will be stored as [B, x, k], where B is the set of all chains in the blob, x is the leaf representing the contracted blob in the new network (either C_1^k or C_m^m depending on the presense of an m-chain), and k is an integer representing the type of blob it used to be.
The chains in the blob are stored in the order k, l, m, n. And such that the first leaf of the k-chain has minimal distance 4 to both the first leaf of the l-chain and the m-chain (if they exists in the blob), and the last leaf of the k-chain has minimal distance 4 to the last leaf of the n-chain (if it exists in the blob). It is important to see that switching C^k and C^l results in an isomorphic network. The same goes for C^m and C^n.

Cherries are stored as [(x,y), x, 1], where (x,y) is the pair of leaves forming the cherry, x is the corresponding leaf in the resulting network, and 1 the integer corresponding to a cherry contraction.

Uncontained leaves and chains are stored differently: [(Y,Z), C, k], where (Y,Z) corresponds to the partition of X on the uncontained chain/leaf, which will be described in detail in section 4.7. C denotes the uncontained chain or the uncontained leaf. In the case of an uncontained chain, C is stored such that the leaves in Y have a shorter minimal distance to C_1 than to C_k.

4.2 Cherries

The first thing that is done is identifying and contracting the cherries. When starting the deconstruction, the chains have not yet been constructed, and therefore all leaves are investigated (C is a list of all seperate leaves initially). In later iterations, the cherries are never contained in chains, and therefore the leaves that are contained in the chains are not investigated.
Two leaves form a cherry if and only if their multiset equals {2}. When contracting a cherry, the matrix is adjusted in such a way that the first leaf of the cherry corresponds to the leaf obtained from the contraction in the new matrix, and the second leaf of the cherry is removed. This is achieved by setting all multisets from and to the second leaf to {0}, and reducing the entries of all multisets from and to the first leaf by 1.
New cherries may occur during the deconstruction. For example, if a pendant blob is contracted and on its non-trivial cut-edge is an uncontained leaf, then after the contraction, the resulting leaf and the uncontained leaf form a new cherry together. Therefore, we need to check if there are cherries in the matrix after each pendant blob contraction.

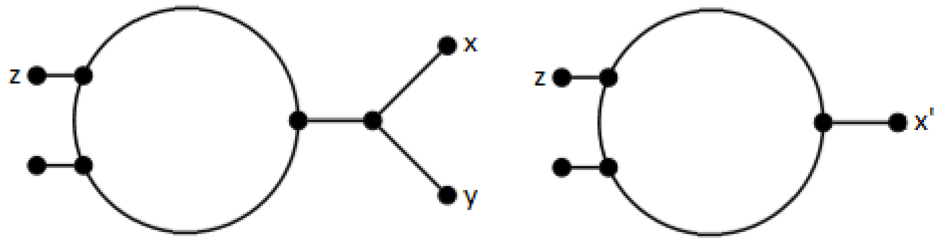


Figure 2: Cherry contraction. $d^N(x,z) = \{4,5\}, d^N(x',z) = \{3,4\}$

In figure 2, the leaf resulting from the contraction is labelled x'. However, in the algrorithm, it will not be relabeled, as the multisets of leaf x will be updated instead of a new leaf being created.

Recognize_Cherries

```
input : M, the matrix of multisets of distances
        C, the set of all chains,
        - In the first iteration, the chains have not yet been constructed.
        In this case, C is still a list of all leaves.
output: M, the updated matrix of multisets.
        s, the deconstruction step to store the process
1  s = [ ]
2  for $i \in (0, ..., \#C - 1), \#C^i == 1$ do
3      for $j \in (i + 1, ..., \#C), \#C^i == 1$ do
4          if $M_{i,j} == \{2\}$ then
5              s = [($C^i$, $C^j$), $C^i$, 1]
6              M = Remove_Cherry(M, C, $C^i$, $C^j$)
7          end
8      end
9  end
10 Call Adjust_Chain for all leaves that result from contractions if it's not the first step.
11 return M, s
```

Remove_Cherry

```
input : M, the matrix of multisets
        C, the set of all chains
        x, the first leaf of the chain
        y, the second leaf of the chain
output: M', the updated matrix of multisets.
1  $M_{x,y} = \{0\}$
2  for $c \in C$ do
3      for $i \in c$ do
4          $M_{i,y} = \{0\}$
5          $M_{y,i} = \{0\}$
6          if $M_{i,x} \neq \{0\}$ then
7              $M_{i,x} = M_{i,x} \ominus 1$
8              $M_{x,i} = M_{i,x}$
9          end
10     end
11 end
12 return M
```

Lemma 4.1. *Applying 'Recognize_Cherries' on a multiset Matrix M of network N results in the Matrix M' corresponding to the Network N' where all cherries of N are contracted, and does so in running time order $O(n \cdot D)$ per cherry.*

Proof. If there are no cherries in N, then there are no multisets of distances in M equal to $\{2\}$. In this case, no cherries are identified, and the matrix remains unchanged. If a cherry is identified, which is done in time order $O(n^2)$, it is contracted by setting the multisets from and to the second leaf of the cherry to $\{0\}$, taking $O(n)$ time, and reducing each entry in all multisets from and to the first leaf by 1, which is based on the size of each multiset adjusted, hence$O(n \cdot D)$. Therefore, the contraction of a single cherry is of order $O(n \cdot D)$, and contracting all cherries is of order $O(n^2 \cdot D)$ □

4.3 Construct Chains

Chains are constructed only once. Afterwards, after every step, the existing chains are adjusted. The chains are constructed in the following way: Starting at the lowest index leaf which we will denote by x, the distance matrix is iterated for leaves such that their minimal distance to x is 3. If x is not contained in a chain, then no such leaves are found, and continue on the next leaf. If x is an endpoint of a chain, then 1 leaf is obtained, and we continue on that leaf, etc. Leaves that are already contained in some chain are skipped. When no leaf is found with minimal distance 3 to the last leaf that is inspected, the chain is complete. If x is in a chain, but not an endpoint, then there are exactly 2 leaves that are obtained. Hence if 2 different leaves are found, the iteration is halted. Then the above process is applied on both leaves seperately, but for the first one, every new leaf that is found is appended at the end of the list, and for the second one, it is inserted at the start. This results in 2 seperate lists that can easily be combined.

Note that when constructing the chains, all cherries have been removed from the network. If this were not the case, a problem arises when there is an uncontained leaf and a cherry which leaves both have minimal distance 3 to the uncontained leaf.

Construct_Chains

```
   input  : M, the matrix of all multisets of distances
   output: C, the set of all chains
1  Create X; a list with all leaves in M
2  Create empty list C
3  for $x \in X$ do
4      Create empty lists $L^1$, and $L^2$
5      t = 0
6      for $y \in X - \{x\}$ do
7          if $M_{x,y}[0] == 3$ and $t = 0$ then
8              $L^1 = L^1 \cup \{y\}$,    X = X - {y},    t = 1
9          end
10         if $M_{x,y}[0] == 3$ and $t = 1$ then
11             $L^2 = L^2 \cup \{y\}$,    X = X - {y},    t = 2
12             End for-loop
13         end
14     end
15     if $t == 0$ then
16         X = X - {x},    $C = C \cup \{x\}$
17     end
18     if $t == 1$ then
19         c = Iterate_Chain(M,$L^1$, X)
20         $C = C \cup c$
21     end
22     if $t == 2$ then
23         $c_1$ = Iterate_Chain(M, $L^1$, X),    $c_2$ = Iterate_Chain(M, $L^2$, X)
24         reverse $L^1$ and combine $L^1$ and $L^2$. c = $c_1 \cup c_2$ - {x}
25         $C = C \cup c$
26     end
27 end
28 return C
```

Iterate_Chain

```
input  : M, the matrix of multisets of distances
         L, a pair of leaves with minimal distance 3 to each other
         X, the set of all unused leaves
output: L, a subchain
1 for $x \in X$ do
2     if $M_{L_{-1},x}(0) = 3$ then
3         $L = L \cup x$
4         X = X - {x}
5         restart for-loop
6     end
7 end
8 return L
```

Lemma 4.2. *Applying Construct_Chains' on a multiset Matrix M of a network N (without cherries) returns a list with all ordered chains in the network, and does so in running time $O(N^2)$.*

Proof. Let x be a leaf in the network. If is not part of a chain with length 2 or higher, then there is no leaf y in N such that $d_m(x, y) = 3$, and therefore no leaves are added to its chain in the algorithm. If y is the only leaf with minimal distance 3 to x, then x is an endpoint of the chain, and Iterate_Chain continues on y for leaves that have minimal distance 3 to it that are not yet added to some other chain, and when such a leaf is found, again all leaves not yet contained in some chain are checked for minimal distance 3 to that leaf. For each leaf inspected, all other leaves not yet included in a chain are checked for their minimal distance, this is of order $O(n^2)$. If there is a leaf already in some other chain with minimal distance 3 to y, then y would have been added to that chain when considering that chain (and so was x). Once there is a leaf added to the chain with no other leaf not in the chain to which it has minimal distance 3, the chain is returned, as the last leaf is the other endpoint of the chain.
If there are 2 leaves, y and z, with minimal distance 3 to x, the two seperate chains are created, one starting at x going in the direction of y, and the other, also starting with x, going in the direction of z, both $O(n^2)$. This results in 2 seperate subchains that are combined to a single chain. Because both subchains start with x, the first list is reversed so that x is now the last in the subchain. The second is appended to the first, and the extra entry of leaf x is removed resulting in an ordered chain. Hence, the total running time of constructing the chains is $O(n^2)$. □

4.4 End Conditions

At some point in the deconstruction, there are no more uncontained leaves, cherries, and pendant blobs. The network consists of a single blob with no non-trivial cut-edges, or a pair of vertices. There are 4 possible end situations for the deconstruction:
1.) A pair of vertices (x,y) connected by a single edge, which corresponds to $d^N(x, y) = \{1\}$ in the matrix. Since only the leaves in C are iterated, if the number of entries of C is not equal to 2, this needs not to be checked. This situation occurs when all blobs in the last deconstruction step were contracted in the same program, that is, were of the same type. O(1)
2.) A level 1 blob with a single chain with no endpoints, that is, every leaf in the chain has exactly 2 different other leaves in the chain to which it has minimal distance 3. This occurs when C contains exactly 1 element. O(1)
3.) A level 2 blob with 2 chains. In this case, #C = 2, and these chains are doubly adjacent. If the remaining network has 2 chains, but is not a pair of vertices, it can still consists of 2 blobs, both with a single chain, to that end the ajdacency is checked. O(1)
4.) A level 2 blob with 3 chains. In this case, #C = 3, and all of these chains are doubly ajdacent to the other two, but it suffices to check if one chain is adjacent to the other 2. O(1)

End

```
input  : M, the matrix of multisets of distances
         C, the set of all chains
output: s, the step in which the end blob is stored, if nothing is
         done, an empty list will be returned.
         .
if $\#C == 1$ then
    if $d_m(C_1, C_k) == 3$ then
        (Cyclic Level-1 blob), s = [$C^i$, [ ], 13]
        return s
    end
end
if $\#C == 2$ then
    if $adjacent(C^0, C^1) == 2$ then
        (Level-2 blob with 2 chains), s = [[$C^i, C^j$], [ ], 14]
        return s
    end
    if $d^N(C^0_1, C^1_1) == \{1\}$ then
        (Two connected vertices), s = [[x,y],[ ], 12]
        return s
    end
end
if $\#C == 3$ then
    if $adjacent(C^0, C^1) == 2$ and $adjacent(C^0, C^2) == 2$ then
        (Level-2 blob with 3 chains), s = [[$C^i, C^j, C^k$], [ ], 15]
        return s
    end
end
return [ ]
```

The pseudocode for the 'adjacent' function can be found in the Appendix; given 2 chains, it returns 0 if they are not adjacent, 1 if they are adjacent, and 2 if they are adjacent twice.

Lemma 4.3. *Given a Distance Matrix M for a network N, the function 'End' identifies if an end-condition is met, and does so in O(1) time. Moreover, no more contractions can be done if and only if one of the 4 stated criteria is met.*

Proof. All end-conditions are uniquely identified. Identifying each one of them takes a set amount of steps independent from the number of leaves, hence each check is of constant time, and there are 4 situations to be checked.
Suppose no more contractions can be done based on the Matrix. Then there are no pendant blobs, cherries, or uncontained leaves. By nature of the network, if there is no pendant blob, there cannot be 2 or more blobs in the network. So there are either no blobs, or a single blob. If there is no blob, then the network is a tree structure that can be contracted to a pair of leaves by iteratively contracting cherries, resulting in a pair of vertices (1). Suppose there is exactly 1 blob. Any leaf in the network that is not part of a chain in that blob is in a tree structure that can also be contracted by repetitive cherry contractions, thus we end up with a network in which all leaves are contained in the blob. If it is a level-1 blob, then there is exactly 1 chain, as the blob has no cut-edges (2). If the blob is Level-2, then as there are no cut-edges, there are at most 3 sides of the blob on which a chain can occur. If there is only 1 chain, the other 2 sides are parallel edges, which we disallow in the networks we consider. Hence, there are either 2 or 3 chains in the blob. (3) (4). Similarly, if one of the end-conditions is met, the network consists of a single blob or a pair of vertices, neither of which is contracted by means described in the Lemma's. □

Consider a network with 2 pendant blobs of the same type, that is, they are deconstructed in the same step. As the chains are adjusted after all contractions in a step are done, and not after each contraction in the step. The chains of the second blob that are combined after adjusting the chain are still considered seperate chains, and the blob itself will be contracted as if it were a pendant blob, resulting in a pair of leaves after both are contracted.

4.5 Pendant Level 1 Blobs and Level 2 k000 blobs

Lemma 4.4 (Lemma 5.6 of [1]). *Let N be a level-2 network on X. A chain $(a_1, a_2, ..., a_k)$ with $k \geq 2$ is contained in a pendant level-1 blob if and only if $d(a_1, a_k) = \{4, (k+1)\}$.*

Lemma 4.5 (Lemma 5.8 of [1]). *A level-2 network N contains a pendant level-2 blob of the form (k, 0, 0, 0) for $k \geq 2$ with the chain $(a_1, a_2, ..., a_k)$ if and only if $d(a_1, a_k) = \{5, 6, (k+1)\}$.*

As the above lemma's prove, blobs with a single chain can be identified based on the multisets of the endpoints of the chains. When arriving at this step, the chains are already constructed. As there are no level-1 blobs with a single leaf (this would result in a pair of parallel edges), and we treat 1000-level-2 blobs seperately, single leaves need not to be considered. Hence, the identification of such blobs is fairly fast and simple. Even more convenient, as the identification of such blobs is independent from the rest of the network, they can be contracted before uncontained leaves are considered, which can result in an uncontained leaf now being contained in some cherry. This cherry will then be contracted when returning to the cherry contraction step, which is faster than identifying and removing an uncontained leaf.

Lemma 4.6 (Lemma 5.7 of [1]). *Let N be a level-2 network on X in which $(a_1, a_2, ..., a_k)$ is a chain that is contained in a pendant level-1 blob. Let N' be the network on $X' = X \cup \{z\} - \{a_1, a_2, ..., a_k\}$ obtained from N by replacing the pendant blob by a leaf z. For every $x \in X' - \{x\}$, we can uniquely partition the multiset of distances $d^N(x, a_1)$ into two equal sized sets A and B such that $A - 2 = B - (k+1)$. Then the multiset of distances of N' contains the elements*

$$d^{N'}(y, z) = \begin{cases} d^N(x, y) \text{ if } x, y \in X' - \{z\} \\ A - 2 \text{ if } y = z \end{cases}$$

Lemma 4.7 (Lemma 5.9 of [1]). *Let N be a level-2 network on X containing a pendant level-2 blob of the form (k, 0, 0, 0) for $k \geq 1$ with the chain $(a_1, a_2, ..., a_k)$. Then we can replace the pendant blob by a leaf z to obtain a network N' on $X' = X \cup \{z\} - \{a_1, ..., a_k\}$. For every $x \in X' - \{z\}$, we can uniquely partition the multiset of distances $d(x, a_1)$ into four equal sized sets A, B, C, D such that $A - 3 = B - 4 = C - (k+2) = D - (k+3)$. Then the multisets of distances of N' contains the elements*

$$d^{N'}(x, y) = \begin{cases} d^N(x, y) \text{ if } x, y \in X' - \{z\} \\ A - 3 \text{ if } y = z \end{cases}$$

Let $c = \{c_0, ..., c_k\}$ be a chain. By Lemmas 3.1 and 4.6, if follows that in a level-1 single chain blob: for all $x \in X - \{c\}$, $d^{N'}(x, c_1) = d^N(x, c_1) \ominus \{2, (k+1)\}$.
Using Lemmas 3.1 and 4.7: In a single chain level-2 blob: for all $x \in X - \{c\}$, $d^{N'}(x, c_1) = d^N(x, c_1) \ominus \{3, 4, (k+2), (k+3)\}$.
These sets over which $d^N(x, c_1)$ is split, correspond to the multiset of distances from c_1 to the non-trivial cut-vertex of the blob.
The Split Function requires the input lists to be sorted, and (k+1) can be smaller than 5 or larger than 6, therefore it is sorted before calling Split. In both cases $d^N(x, c_i)$ is set to $\{0\}$ for all $2 \leq i \leq k$

Single_Chain_Blob

```
input  : M, the matrix of the multisets of the distances
         C, the set of all chains
output: M, C,
         s; the deconstruction step
s = []
for $c \in C$ do
    if $\#c \geq 2$ then
        $T_1 = \{4, (\#c + 1)\}$, $T_2 = \{5, 6, (\#c + 1)\}$
        Sort both $T_1$ and $T_2$
        x = $M_{c_0,c_k}$
        if $x == T_1$ then
            S = {2, (#c+1)}
            M = Contract_kl00(M, c, S)
            C = C - c
            C = Adjust_Chain(C, $c_0$)
            s = [$C^i$, $C^i_1$, 2]
        end
        if $x == T_2$ then
            S = {3, 4, (#c+2), (#c+3)}
            M = Contract_kl00(M, c, S)
            C = C - c
            C = Adjust_Chain(C, $c_0$)
            s = [$C^i$, $C^i_1$, 3]
        end
    end
end
return M, C, s
```

Lemma 4.8. *Given a network N and a Matrix of Multisets M, 'Single_Chain_Blobs' returns an updated Matrix M' corresponding to the network N', in which all pendant single chain blobs of N are contracted to leaves, and does so in $O(n \cdot D)$ per blob.*

Proof. By Lemma's 4.4 and 4.5, Pendant Blobs with a single chain in the network N can be uniquely identified by the multisets of the endpoints of said chain, which takes constant time per chain, and therefore iterating over all chains is of order O(n). If no chain satisties this property, then there is no pendant single chain blob, and therefore both the network and the matrix remain unchanged after running this program. Suppose a chain is identified as a a single chain pendant Blob, then the Matrix will be contracted by the 'Contract_kl00' function, with split set S depending on the Blob being Level-1 or Level-2, which by Lemma's 4.4 and 4.5 properly partitions the multisets into 2 or 4 equal sized sets depending on the level. Per blob, the split function is called in the order of n times, and takes $O(D)$ time per time it is called, $O(n \cdot D)$. Setting the other multisets to $\{0\}$ takes constant time per multiset, but is done for every but one blob in the chain to and from all leaves not in the blob, so $O(n^2)$. As adjusting the chains is of order $O(n)$, and all blobs are contracted simultanuously, the total running time of contracting a single blob is $O(n \cdot D)$, and contracting all blobs is $O(n^2 \cdot D)$. When all blobs are contracted, the resulting matrix of multisets corresponds to the network N' in which all single chain blobs of N are contracted. □

4.6 Level 2 kl00-blobs

Lemma 4.9. *A level-2 network N or X contains a pendant level-2 blob of the form (k,l,0,0) with chains $C^1 = (C^1_1, ..., C^1_k)$ and $C^2 = (C^2_1, ..., C^2_l)$ with k,l $\geq$ 1 if and only if C^1 and C^2 are adjacent twice,and $d^N(C^1_1, C^2_1) = \{4^2, 6^2\}$ if $\#C = \#D = 1$, $d^N(C^1_1, C^1_k) = \{6, k+1, l+5\}$ if $\#C \geq 2$, or $d^N(C^2_1, C^2_l) = \{6, l+1, k+5\}$ if $\#D \geq 2$.*

Proof. Let C^1 and C^2 be doubly adjacent chains. If $\#C^1 = \#C^2 = 1$, then $d^N(C^1_1, C^2_1) = \{4, 4, 6, 6\}$. The 4's in this multiset result from C^1 and C^2 being doubly adjacent, which only occurs in a pendant blob when the chains correspond to the k and l sides. If C^1 and C^2 are doubly adjacent, but not contained in a pendant blob, then there are exactly 2 paths from any leaf in that blob to any other leaf in that blob. Hence, if $\#d^N(C^1_1, C^2_1) > 2$, then these chains are contained in a pendant blob. Therefore there is only 1 cut-vertex in that blob. The paths from C^1_1 to C^2_1 both have length 6 if there are no leaves on the m and n sides of the blob.
If $\#C^1 \geq 2$, and assume there are no leaves on the m and n side of the pendant blob. There are 3 paths from C^1_1 to C^1_k.One that traverses all vertices adjacent to the leaves in C^1 and has length k+1, one that traverses all vertices adjacent to the leaves in chain C^2 and has length l+5, and the last path traverses the cut-vertex and has length 6.
Now suppose $d^N(C^1_1, C^1_k) = \{6, k+1, l+5\}$ for doubly adjacent chains. C^1 and C^2 must be contained in a pendant blob, otherwise the number of paths from C^1_1 to C^1_k would not be 3. □

Lemma 4.10 (Lemma 5.12 of [1]). *Let N be a level-2 network on X containing a pendant level-2 blob of the form (k, l, 0, 0) for $k, l \geq 1$ with chains $(a_1, a_2, ..., a_k)$ and $(b_1, a_2, ..., b_l)$. Then we can replace the pendant blob by a leaf z to obtain a network N' on $X' = X \cup \{z\} - (a \cup b)$. For every $x \in X' - \{z\}$, we can uniquely partition the multiset of distances $d(x, a_1)$ into four equal sized sets A, B, C, D such that $A - 3 = B - (l+4) = C - (k+2) = D - (k+l+3)$. Then the multisets of distances of N' contains the elements*

$$d^{N'}(x,y) = \begin{cases} d^N(x,y) \text{ if } x, y \in X' - \{z\} \\ A - 3 \text{ if } y = z \end{cases}$$

First it is determined which chains are doubly adjacent. That is, if both endpoints of one chain have minimal distance 4 to an endpoint of the other chain. If both chains are single leaves, then checking the minimal distance in the same way does not suffice (For example, in a 1010-blob). Single leaves c and d are double adjacent if their if $d^N(c, d)$ contains 4 exactly twice.Note that in this case, it will be checked whether the size of that multiset is 4 or less first. If only one of the two chains is a single leaf, then no problems occur in considering that single leaf as both endpoints, as if both endpoints of the chain have minimal distance 4 to that single leaf, then they traverse different paths. A chain can only be double adjacent to 1 other chain. Hence, if two chains are determined to be double adjacent, the multiset property is checked if it satisfies the above Lemma.

Contract_kl00_blobs

```
input  : M, The Matrix of multisets of distances
         C, The set of all chains
output: M', the updated matrix of multisets of distances,
         C, The updated set of all chains,
         s, the step to store the deconstruction,
s = [ ]
for $c, d \in C$ do
    if adjacent(M, c, d) == 2 then
        if #c = #d = 1 then
            if $M_{c,d} = \{4, 4, 6, 6\}$ then
                S = {3, (l+4), (k+2), (k+l+3)}, sorted
                s = [[c,d], $c_1$, 4]
                Contract_kl00(M, C, [c, d], S),   return M, C, s
            end
        end
        if #c ¿1 then
            if $M_{c_1,c_k} = \{6, k+1, l+5\}, sorted$ then
                S = {3, (l+4), (k+2), (k+l+3)}, sorted
                s = [[c,d], $c_1$, 4]
                Contract_kl00(M, C, [c, d], S),    return M, C, s
            end
        end
        if #d ¿1 then
            if $M_{d_1,d_l} = \{6, l+1, k+5\}, sorted$ then
                S = {3, (l+4), (k+2), (k+l+3)}, sorted
                s = [[c,d], $c_1$, 4]
                Contract_kl00(M, C, [c, d], S)    return M, C, s
            end
        end
    end
end
Return M, C, s
```

Lemma 4.11. *Given a Matrix of multisets of distances M for network N, 'Contract_kl00_Blobs' returns an updated Matrix M', corresponding to network N' in which all pendant blobs of the form kl00 of N have been contracted; and does so in Theoretical Running Time of $O(n \cdot D)$ per blob.*

Proof. By Lemma 4.9, kl00 blobs can be uniquely identified based on the Multiset of Distances of the endpoints with respect to their lengths; O(1) per chain, O(n) for all chains, and can be contracted by applying the split function on $d^N(c_1, x)$ for all $x \in X - \{c, d\}$ with S = {3, (l+4), (k+2), (k+l+3)}, and setting all distance multisets from and to all $x \in c, d - \{c_1\}$ to {0}; $O(n \cdot D)$ for applying the split function on the multisets of leaf c_1. Now the matrix M is properly adjusted to represent the theoretical network N in which all pendant kl00-blobs have been contracted. The running time of the contraction of a single blob is $O(n \cdot D)$, and for all blobs is $O(n^2 \cdot D)$. □

4.7 Uncontained Leaves

Lemma 4.12. *Let N be a level-2 network on X where $|X| \geq 3$. A leaf x is not contained in a blob if and only if there exists a unique partition $Y \cup Z$ of X - $\{x\}$ such that $Y, Z \neq$ and $d_m(y,z) = d_m(x,y) + d_m(x,z) - 2$ for all $y \in (E \cap Y)$ and $z \in (E \cap Z)$.*

Proof. Let x be an uncontained leaf. By Lemma 5.4 of [1], X - {x} can be partitioned into 2 sets Y and Z such that $d_m(y,z) = d_m(x,y) + d_m(x,z) - 2$ for all $y \in Y$ and $z \in Z$, which can be generalised to $d_m(y,z) = d_m(x,y) + d_m(x,z) - 2$ for all $y \in (E \cap Y)$ and $z \in (E \cap Z)$ by Lemma 2.3. □

The idea here is that if the paths from a leaf y to a leaf z traverse the vertex adjacent to an uncontained leaf x, then y and z belong to different sets of the partition. If the paths do not traverse the adjacent vertex of x, then y and z belong to the same set in the partition.

Observation 4.13. *Let $C = \{c_1, c_2, ..., c_k\}$ be a chain. If c_i is an uncontained leaf for some i, then the whole chain C consists of uncontained leaves.*

Let C = $\{c_1, c_2, ..., c_k\}$ be a chain and c_i an uncontained leaf, with $1 \leq i \leq k$. Any path from c_i to some leaf x not contained in C is at least length 4. For all j in {1,2,...,k-1}, $d_m(c_j, c_{j+1}) = 3$. So if c_i is uncontained, so are c_{i-1} and c_{i+1}. By inductively applying the same argument on those leaves, we find that all leaves in C are uncontained.

Lemma 4.14. *Let $C = \{c_1, c_2, ..., c_k\}$ be a chain of uncontained leaves. Then for all i: X - $\{c_i\}$ = Y $\cup$ Z = (Y' U $\{c_1, c_2, ..., c_{i-1}\}$) $\cup$ ($\{c_{i+1}, c_2, ..., c_k\}$ $\cup$ Z') such that $d_m(y,z) = d_m(x,y) + d_m(x,z) - 2$ for each $y \in$ Y and $z \in$ Z. And moreover; X - C = Y' $\cup$ Z'.*

Proof. Let c = $\{c_1, ..., c_k\}$ be a chain, and c_1 be the leaf under investigation. For all inspected leaves not in that chain, the partition can be created based on the distances to the endpoints of c - $\{c_1\}$. In this case, the partition is created as follows: Denote C - c as $\{d^1, ..., d^n\}$.Start with j = 1: If $d_m(d^1_1, c_2) > d_m(d^1_1, c_k)$, then $d^1 \in Z$, else $d^1 \in Y$.
Put d^j in either Y or Z in the same way for every j. Every time a chain is added to the the partition, check if:
$d_m(c_0, y) + d_m(c_0, d^j_{0/k_j}) - d_m(y, d^j_{0/k_j}) \geq 2$ for all $y \in Y$ if d^j in Z,
and $d_m(c_0, z) + d_m(c_0, d^j_{0/k_j}) - d_m(z, d^j_{0/k_j}) \geq 2$ for all $z \in Z$ if d^j in Y.
If this property does not hold, then c is not an uncontained chain.
As every chain will return a partition Y $\cup$ Z = X - {C}, checking the distance property each time a new chain will return False on contained chains earlier.
For each leaf not in the chain, the minimal distance to both endpoints of the chain is inspected. O(n). The distance properties are checked each time an instance is added to either of the set, which takes O(n) time on every addition. Therefore, the running time of recognizing an Uncontained Chain is $O(n^2)$.

□

Uncontained Chains

```
input : M, a matrix of multisets of distances
        C, a list of chains
output: M, C, s
c' = deepcopy(c)
c'.remove($c_0$)
for $x \in (c'_0, c'_k)$ do
    for $j \in (0, .., \#C)$ do
        for $y \in (0, -1)$ do
            if $d_m(x, C^j_y) == d_m(c_0, C^j_y) + d_m(c_0, x) - 2$ and $c \notin Z$ and $C^j \notin Y$ then
                if $c \notin Y$ then
                    Y = Y ∪ c
                end
                if $C^j \notin Z$ then
                    Z = Z ∪$C^j$
                end
            end
            else
                if $c \notin Z$ and $C^j \notin Z$ then
                    if $c \notin Y$ then
                        Y = Y ∪ c
                    end
                    if $C^j \notin Y$ then
                        Y = Y ∪$C^j$
                    end
                end
            end
        end
    end
end
if $Z \neq []$ and $\#Y + \#Z + 1 = \#C$ and $Z \neq Y$ then
    $y, z = smallest\_distance(M, Y, Z, c)$
    if $d_m(c_0, y) > d_m(c_1, y)$ then
        reverse c,    s = [[Y,Z], $C^i$, 11]
        Contract_Uncontained(M, c, Y, Z)
    end
    return M, C, s
end
return M, C, [ ]
```

Contract_Uncontained

```
input  : M, a matrix of multisets of distances
         c, an uncontained chain or leaf
         Y, Z, the partition on the uncontained leaf or chain
output: M
1  for $y \in Y$ do
2      for $z \in Z$ do
3          $M_{y,z} = M_{y,z} \ominus \#c$
4      end
5  end
6  for $x \in Y \cup Z$ do
7      for $y \in c$ do
8          $M_{x,y} = \{0\}$
9      end
10 end
11 return M
```

Uncontained Leaf: [[Y,Z], x, 10]
The partitions need to be stored for the reconstruction, as they are needed to determine the unique edge on which the uncontained leaves have to be placed.

Lemma 4.15. *Given a Matrix of Multisets of Distances M for a theoretical network N, applying 'Uncontained_Leaves' and 'Uncontained_Chains' returns a matrix M' corresponding to the network N' in which all uncontained leaves of N have been removed, and does so in Running Time $O(n^2 \cdot D)$.*

Proof. By Lemma's 4.12 and 4.14, uncontained leaves (and chains) can be uniquely identified based on the distance properties of the endpoints of the chains. The first property checks the minimal distance from all leaves to the inspected leaf, O(n). The second property checks all endpoints to all other endpoints, $O(n^2)$.
The partition for the uncontained chains are based on the minimal distance from and to the last leaf in the inspected chain. For an uncontained leaf, this same method cannot be used. In this case, a partition is made based on the second distance criteria.
The matrix is updated by decreasing all distances from leaves in y to leaves in z by the length of the chain $O(n^2 \cdot D)$, and setting all distance multisets from and to all uncontained leaves to $\{0\}$, $O(n^2)$ for both the leaf and the chain. □

4.8 Level 2 1000-blobs

Corollary 4.15.1. *A level-2 network N contains a pendant level-2 blob of the form (1, 0, 0, 0) containing the leaf a if and only if $d_m(x, z) \geq 6$ for all $x \in E - \{z\}$ and for any two leaves $c, d \in E - \{z\}, d_m(c, z) + d_m(d, z) - d_m(c, d) \geq 8$.*

Proof. By Lemma (..) of [1], a network contains a pendant 1000-blob if and only if the above properties hold for all leaves in X - {z}. By Lemma's 2.2 and 2.3, it is known that these conditions hold for all leaves if and only if they hold for all endpoints of all the chains, that is for all x $\in E - \{z\}$. □

Because this leaf a is not in some chain of length $\geq$ 2, only then entries c in C with #c = 1 are inspected.
Moreover, there are no cherries and uncontained leaves at this point. First, all endpoints and single leafs are iterated to check their minimal distances to the leaf a. If there is a leaf x such that $d_m(a, x) \leq 5$, then leaf a is not of this form and we continue with the next. If x itself was a single leaf, then leaf x need not to be inspected when it is encountered. Once it has been verified that there is no such other leaf x, the next property is investigated. It is now know that leaf a is the only leaf in the blob it is in, otherwise, it would have a leaf to which its minimal distance is 4. This leaves us with 2 scenarios, the blob in which leaf a is contained is either pendant or non-pendant. A non-pendant blob has 2 cut-edges. The second property will not be satisfied is one takes a leaf x that traverses any cut-edge on its paths to a, and another leaf y that traverses another cut-edge on its paths to leaf a. If all required pair of leaves are inspected for this property and all of them satisfy it, then we can conclude a is Level 2 1000-blob, and it can be contracted in the same way as a Level 2 k000-blob can be contracted.
Afterwards, all distances from and to a are updated to have the multisets to the new leaf a' corresponding to the contracted blob. The split function is called N-1 times, as all leaves in $X - \{a\}$ are adjusted, hence, as the split function is O(D), the running time of the update of the matrix is $O(n \cdot D)$.

Contract_1000_blobs

```
input  : M, C
output: M, C
s = [] for $c$ in $C$, $\#c = 1$ do
    if $min\_distance\_largerthan\_k(M, C, c, 6) == True$ then
        for $d_1, d_2 \in C - \{c\}$ do
            for $y_1, y_2 \in (0, -1)$ do
                x = cross_distance_check(M, c, $d_1, d_2$, 8) if $x == False$ then
                    return M, C, [ ]
                end
            end
        end
    end
    S = {3,3,4,4}
    s = [c, c, 4]
    M = Contract_kl00(M, S, c)
end
return M, C, s
```

Lemma 4.16. *Given the Distance Matrix M for a network N, 'Contract_1000_Blobs' returns a matrix M corresponding to a network N' in whcih all pendant 1000-blobs of N have been contracted to a leaf, and does so in running time order $O(n^2 \cdot D)$.*

By Corollary 4.16, a leaf x is part of a 1000 level-2 pendant blob if and only if the two distance properties are satisfied for all endpoints. The function Contract_1000_Blobs first calls the function that checks if all distances are larger than 6, and if that is the case, it will proceed checking the second property over all endpoints of the chains. If any pair of endpoints does not satisfy this criteria, the programs terminates without contracting the blob. If such a blob is identified, it will contract it by means of Lemma (..).
Checking the if the distance of all endpoints to a leaf a is larger than 6 takes time proportional to the number of leaves, O(n).
Determining if the second distance property holds for all pair of endpoints is of running time $O(n^2)$.
Once the blob is identified, all distances from and to the leaf a are updated using the split function, $O(n \cdot D)$.
Contracting one 1000-blob in the network is therefore of order $(n \cdot D)$, and contracting all blobs is of order $(n^2 \cdot D)$
As all steps are done sequentially, the total theoretical running time is the maximum over all individual running times, which is $O(n^2 \cdot D)$.

4.9 Level 2 klmn-blobs

Recalling the following definition and results from [1]:

(**Definition 5.13** of [1]) A chain-adjacency graph (CAG) has a vertex for each chain, and between two vertices,
- we insert a red edge if the chains are adjacent once and two red edges if the chains are adjacent twice; and
- if the two chains are adjacent once, we insert a green edge for each length-5 path between endpoints of the chains (one per chain) that does not contain any edges of the two chains.

(**Theorem 5.14** of [1]) Let N be a level-2 network on X with at least two blobs, where no pendant blobs are of the form (k, 0, 0, 0) and (k, l, 0, 0) in which all leaves are contained in blobs. For $k, l, m, n \geq 1$, N contains a pendant level-2 blob of the form
- (k,0,m,0) if and only if there exist vertices a and c which form a blob in the CAG with 1 red edge and 2 green edges between them.
- (k,l,m,n) if and only if there exists vertices a, b, and c which form a blob in the CAG, where a and b are connected by 2 red edges and the other two pairs are connected by 1 red edge and 1 green edge.
- (k,0,m,n) if and only if there exists vertices a, c, and d which form a blob in the CAG, wehere every pair of vertices are connected by 1 red edge, and a and b are connected by and additional red edge.

(**Lemma 5.15** of [1]) Let $k, l, m, n \geq 1$, and let B be a pendant level-2 blob that is of the form (k, 0, m, 0); (k, l, m, 0); (k, 0, m, n); or (k, l, m, n). Then we can replace the pendant blob by a leaf z to obtain a network N' on $X' = X \cup \{z\} - (a \cup b \cup c \cup d)$, such that the multiset of distances of N' contains the elements

$$d^{N'} = \begin{cases} d^N(x, y) \text{ if } x, y \in X' - \{z\} \\ A - 2 \text{ if } y = z \end{cases}$$

where if B is of the form
- (k, 0, m, 0), then we uniquely partition $d(x, c_m)$ into three equal sized sets A, B, C such that$A - 2 = B - (m + 3) = C - (k + m + 3)$.
- (k, l, m, 0), then we uniquely partition $d(x, c_m)$ into three equal sized sets A, B, C such that $A - 2 = B - (l + m + 3) = C - (k + m + 3)$.
- (k,0,m,n), then we uniquely partition $d(x, c_m)$ into three equal sized sets A,B,C such that $A - 2 = B - (m + n + 3) = C - (k + m + n + 3)$
- (k,l,m,n), then we uniquely partition $d(x, c_m)$ into three equal sized sets A,B,C such that $A - 2 = B - (l + m + n + 3) = C - (k + m + n + 3)$

Consider a pendant klmn level-2 blob, and denotethe cut-vertex of the blob incident to the non-trivial cut-edge by v. This cut-vertex corresponds to the leaf that results from contracting the blob to a leaf. The elements {2, (l+m+n+3), (k+m+n+3)} are exactly the paths from the cutvertex v to the vertex c_m, the endpoint of the m-chain that has the shortest minimal distance to v. (where l and n are 0 if there is no chain on its corresponding side in the blob.)

By Theorem 5.14 of [1], Pendant Level-2 klmn blobs can be identified by determining the adjecency of the chains and counting the number of paths with length 5.
To this end, a matrix A is constructed, with $A_{i,j} = \{r, g\}$, where r ∈ {0,1,2}, representing the number of times chains i and j are adjacent, and g is the number of paths with length 5 paths from the endpoints of chain i to the endpoints of chain j, not passing any edges in either chain, minus 0,1, or 2, based on the sizes of the chains (ref.) Determining if two chains are adjacent (or twice) takes constant time, and as it is applied on all chains to all other chains, this substep is of order $O(n^2)$. Checking the number of length 5 paths between the endpoint of 2 chains is of constant time. Ergo, the construction

of matrix A is of order $O(n^2)$
After constructing the matrix A, it is divided into submatrices based, such that each submatrix corresponds to a pendant klmn-blob. As two chains in the same pendant blob are always adjacent at least once, each entry of the matrix is only checked once, and therefore dividing the matrix into submatrices is of order O(n).
At this point in the deconstruction, there are no uncontained leaves, therefore if $4 \in d^N(a,b)$, with a and b not in the same chain, then a and b are endpoints of adjacent chains, and therefore in the same blob.
Having constructed the submatrices, each of these submatrices is checked if entries adhere to the properties as presented in Theorem 5.18.
An important part in the deconstruction is storing the chains in a uniform way:
Step $= [[C^k, C^l, C^m, C^n], C_1^k, s]$, where C^k is the chain corresponding to the k-side of the blob, etc. and s an integer corresponding to the type of blob.
The chains are ordered such that C_1^k has minimal distance 4 to C_1^l and C_1^m, andC_k^k has minimal distance 4 to C_n^n Note that switching C^k and C^l results in an isomorphic network in the deconstruction, as do switiching C^m and C^n, so in that regard, there are 4 proper ways of storing the blob in the steplist.
To that end, take an arbitrary leaf x that is not contained in the blob that is investigated. The endpoints of the m and n chain that have the shortest minimal distances to x are C_m^m and C_1^n respectively (although the choice which is which is arbitrary), if the pendant blob has a chain on both the m and n sides. If there is only a chain on the m side, then this minimal distance is unique and the endpoint corresponds to C_m^m. If the blob has both chains on the k and l side, these are always the only doubly adjacent chains and can be distinguised by this property. Therefore, now only the minimal distance of the endpoints of both C^k and C^l to C^m have to be inspected. If they do not satisfy the criteria, both C^k and C^l are reversed, that is, C_1^k switches position with C_k^k, C_2^k switches with C_{k-1}^k and vice versa.

klmn_contraction

```
input  : M, C
output: M, C
s = [ ]
R = #C x #C matrix filled with 0's
for $c \in (0, ..., \#C - 1)$ do
    for $d \in (c + 1, \#C)$ do
        $R_{c,d} = adjacency(c, d)$   *returns 0,1, or 2, depending on the number of times c and d are adjacent.*
    end
    S, X = create_submatrices(R)   *S is the set of the submatrices, and X is a set containing the set of chains of each of the submatrices*
end
L = [], an empty list to store all leaves remaining after the contraction. for $k \in (1, ..., \#S)$ do
    G = $\#S^k \cdot \#S^k$ matrix filled with 0's
    for $i \in (0, ..., \#S^k - 1)$ do
        for $j \in (i, ..., \#S^k)$ do
            if $S^k_{i,j} = 2$ then
                $G_{i,j} = \{2, 0\}$
            end
            if $S^k_{i,j} = 1$ then
                A = $d^N(c_0, d_0).count(5)$,   B = $d^N(c_0, d_{-1}).count(5)$
                C = $d^N(c_{-1}, d_0).count(5)$,   D = $d^N(c_{-1}, d_{-1}).count(5)$
                Where c and d are the chains corresponding to $S^k_i$ and $S^k_j$ respectively.
                x = sum of A, B, C, and D with respect to Table 1 on p14 of [1].
                $G_{i,j}$ = [1,x]
            end
        end
    end
    if $\#G = 2$, and $G_{1,2} = \{1,2\}$ then
        The obtained matrix G corresponds to a pendant blob of the form (k,0,m,0)
        $X_k$ = sortblob($X_k$),   s = [$X_k$, $C^m_m$, 6],   L = L ∪ $C^m_m$
    end
    if $\#G = 3$ then
        if {2,0} ∈ G and {1,1} ∈ G twice then
            The obtained matrix G corresponds to a pendant blob of the form (k,l,m,0)
            $X_k$ = sortblob($X_k$),   s = [$X_k$, $C^m_m$, 7],   L = L ∪ $C^m_m$
        end
        if {1,1} ∈ G three times then
            The obtained matrix G corresponds to a pendant blob of the form (k,0,m,n)
            $X_k$ = sortblob($X_k$),   s = [$X_k$, $C^m_m$, 8],   L = L ∪ $C^m_m$
        end
    end
    if $\#G = 4$ then
        if {2,0} ∈ G and {1,1} and/or {1,0} ∈ G 5 times then
            The obtained matrix G corresponds to a pendant blob of the form (k,l,m,0)
            $X_k$ = sortblob($X_k$)   s = [B, $C^m_m$, 9],   L = L ∪ $C^m_m$
        end
    end
    M = contract_klmn(M, C, S, B)
end
for $l \in L$ do
    C = adjust_chains(M, C, l)
end
return M, C, s
```

Lemma 4.17. *Given a Matrix of Multisets of Distances for a network N, the function 'klmn_contraction' returns a matrix M' corresponding to N', the network in which all pendant klmn-blobs of N have been contracted to a leaf, and does so in Running Time of Order $O(n \cdot D)$ per blob.*

Proof. By Theorem 5.14 of [1], level-2 pendant blobs are uniquely identified based on number of the so-called red and green edges. As there are no uncontained leaves in the network at this point in the deconstruction, $d_m(x, y) = 4$ if and only if x and y in the same blob, and if x and y in different chains, then x and y are endpoints of these chains, and the chains are adjacent.

Determining the adjacency of two chains takes constant time. As we examine all chains agains all other chains, the running time of this part is $O(n^2)$.
To create the submatrices, one takes a chain and add all that are adjacent to it to the submatrix, then all chains in this submatrix are adjacent to each other. In this process, each chain is checked for adjacency with 1 chain in each existing submatrix, because a pendant level 2 blob, all existing chains are at least one time adjacent to each other chain. Then all submatrices of size 1 are removed. $O(n^2)$.
As all chains that are adjacent to each other are in the same blob, the multisets have at most size 4 for 2 leaves in the same blob. Moreover, there are at most 4 chains in a blob. Hence, the counting of green edges is of order O(1).
Re-arranging the chains in the blob such that they are in the proper order takes constant time, as identifying C^m and C^n is based on the minimal distance to a single arbitrary leaf not in the blob, and the correct order of C^k and C^l O(1) is determined by the minimal distance to C_1^m.
The contraction of a single blob consist of applying the split function on $d^N(C_k^k, x)$ for all x ∈ X - {B} ($O(n \cdot D)$, and setting the multisets from and to all other leaves in the blob to $\{0\}$, hence $O(n \cdot D)$ per blob. As all steps are done sequentially, the running time of deconstructing a klmn-blob is of order $O(n \cdot D)$ per blob. □

Note that multiple blobs can be contracted in the same iteration of the function, and the total theoretical running time of this subalgorithm is $O(n^2 \cdot D)$. But as throughout the whole deconstruction process, at most n blob contractions are done, the running time is considered per individual blob.

4.10 Adjust Chains

After each step in the deconstruction process, the chains are adjusted, as contracting a blob or cherry may result in the resulting leaf being part of some chain. If it is not an endpoint of that chain, then 2 chains are combined into 1. Let x be the leaf that resulted from a blob or cherry contraction. The distance to all single leaves and endpoints of existing chains is inspected to see it their minimal distance to x is 3 in the updated matrix. If a is a leaf such that it minimal distance to x is 3, the leaf x is added to the chain containing a. If there are 2 leaves with minimal distance 3 to x, then 2 chains have to be combined into 1 chain, also containing x. This is done by adding the leaf x to both chains, adding the union of these chain to the set of all chains, and removing the original chains from the set. Taking the position of x into account in the original updated chains, the resulting union will have the proper ordering of the leaves. However, this method can result in some issues when contracting cherries close to uncontained leaves. Let z be an uncontained leaf, and (x_1, x_2) be a cherry, such that the resulting contracted leaf x is a cherry with leaf y. Now, after updating the matrix, the leaf x_1 will have minimal distance 3 to the uncontained leaf y, but it unwanted that this will be considered a chain in the continuation of the deconstruction. Therefore, the chain will be updated after all contraction in each step are done. That way, the resulting cherry (x,y) will be contracted without adjusting the chains first. (And the resulting leaf will then form a cherry with z, which will also be contracted) Only the leaves that result after all these contractions will call the Adjust Chains function. Additionally, the Adjust Chains function will not be called if cherries are contracted in the first step of the deconstruction process, as the next step will be the construction of the chains.
As the list of chains is iterated over once, the distance checks from x to the endpoints of all chains are of $O(n)$ time. Then, if two chains need to be combined, the union of the two chains is created, which, also is of order $O(n)$.
As these steps are done sequential, the total running time of adjusting the chains is $O(n)$ each time it is called.

Adjust_Chains

```
input  : M, C, x
output: C
a = 0
L={0,0}
if $a < 2$ then
    for $c$ in $C$ do
        if $d_m(c_1, x) == 3$ then
            $C = C - c$
            $L_1 = x \cup c$ (first position in list)
            a = a+1
        end
        else if $d_m(c_k, x) == 3$ then
            $C = C - c$
            $L_0 = c \cup x$ (last position in list)
            a = a+1
        end
    end
end
if $a == 1$ then
    if $L_0 \neq 0$ then
        $C = C \cup L_0$
        return C
    end
    if $L_1 \neq 0$ then
        $C = C \cup L_1$
        return C
    end
end
if $a == 2$ then
    d = $L_0 \cup L_1$
    d = d - {x}
    $C = \cup d$
    return C
end
```

4.11 Contraction

Contract_klmn

```
input  : M, C, B, S
output: M
for $x \in C^k, C^l, C^m, C^n, x \neq C^m_m$ do
    for $y \in (0, ..., \#M)$ do
        $M_{x,y} = \{0\}$
        $M_{y,x} = \{0\}$
    end
end
for $x \in (0, ...\#M), x \notin C^k, C^l, C^m, C^n$ do
    $M_{x,C^m_m} = M_{x,C^m_m} \ominus S$
    $M_{C^m_m,x} = M_{x,C^m_m}$
end
```

Contract_kl00

```
input  : M, C, B, S
output: M
for $x \in C^k, C^l, x \neq C^k_1$ do
    for $y \in (0, ..., \#M)$ do
        $M_{x,y} = \{0\}$
        $M_{y,x} = \{0\}$
    end
end
for $x \in (0, ...\#M), x \notin C^k, C^n$ do
    $M_{x,C^k_1} = M_{x,C^k_1} \ominus S$
    $M_{C^k_1,x} = M_{x,C^k_1}$
end
```

Lemma 4.18. *Contract_klmn and Contract_kl00 return the updated matrix of multisets corresponding to the network in which the blob is contracted to a singe leaf.*

As all leaves in the blob, except for C^m_m or C^k_1 as they corresponds to the resulting leaf in their respective blob type, are removed from the network, therefore there are no paths from and to those leaves in the network. These programs set the multisets of these leaves all to $\{0\}$, so there are no paths from and to those leaves. (Setting it to $\{0\}$ is prefered to removing the leaf from the matrix, as the algorithm relies on the ID's of the leaves in the network). Then the split function with the set S is applied on the remaining multisets from and to C^m_m or C^k_1. Here the set S used to partition the multisets into 4 equal sized sets based on the lengts of the chains. Now all multisets that have to be updated are updated, and all multisets from and to leaves not in the blob remain unchanged.

4.12 Deconstruction Sequence

As all programs to deconstruct the cherries, uncontained leaves, and pendant blobs have been defined, the whole network can be deconstructed.
All pendant blobs, uncontained leaves, and cherries can be identified and pruned off. At each step the matrix of multisets of distances is adjusted in such a way to now represent the network resulting from the contraction. At each step, the resulting network is strictly smaller than it was in the previous step, and it will return the list of steps once one of the end conditions is met.

Deconstruction_Sequence

```
input  : M, the Matrix of multisets of distances
output: S, a set of all steps
1  C = [ ]
2  boole = 1
3  S = [ ], an empty list to store the steps in
4  while boole == 1 do
5      M, C, boole, s = Sequence(M, C, S)
6  end
7  return S
```

Sequence

```
input  : M, C, S
output: S, a set of all steps
1  if S == [ ] then
2      C = Construct_Chains(M)
3  end
4  M, C, s = Contract_Cherries(M, C)
5  M, C, s = End(M, C)
6  M, C, s = Single_Chain_Blob(M, C)
7  M, C, s = kl00Blobs(M, C)
8  M, C, s = Uncontained_Leaves(M, C)
9  M, C, s = 1000Blobs(M, C)
10 M, C, s = klmnBlobs(M, C)
11 In between each step: if s != [ ] then
12     $S = S \cup s$
13     return M, C, 1, S
14 end
```

Theorem 4.19. *Given a Matrix of Multisets of Distances corresponding to a theoretical network N, the Algorithm 'Deconstruction_Sequence' returns a list containing all contractions and removals in such a way that the network N can be reconstructed solely based on this steplist; and does so in Running Time Order $O(n^3 \cdot D)$.*

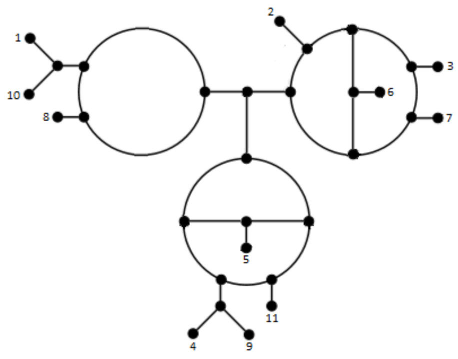
Proof. By Theorem 2.1, it is known that a Level-2 Binary Phylogenetic Network can be deconstructed based on the Matrix of Multisets of Distances between each of the leaves. Every individual deconstruction step is proven to properly adjust the matrix to correspond to the theoretical network resulting from the contraction. By Lemma 4.3, no contractions can be done if and only if an end condition is satisfied. Iif the 'End' function returns a False, then there must be (at least) one contraction that can be done. If that contraction is done, the sequence will return to cherry contractions, and then advance to the 'End' function again. Therefore the deconstruction sequence will always truncate when the deconstruction is finished.

As the deconstruction steps are done sequentially, each call of the sequence is of the maximum over the Running Time Orders of the individual steps, which is $O(n^2 \cdot D)$. Each step on a network with n leaves results in a network with at most n-1 leaves, therefore the total running time of the whole deconstruction phase is of order $O(n^3 \cdot D)$. $\square$

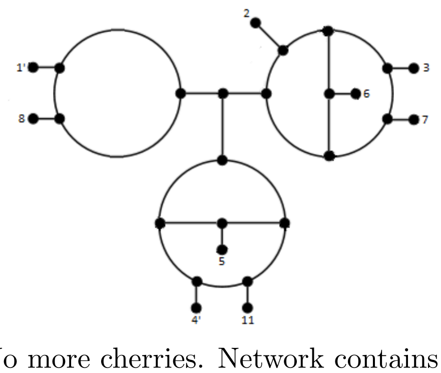
Deconstruction Example

As the deconstruction sequence containing all steps to deconstruct a binary level-2 network has been introduced, applying it to such a network will stepwise contract such a network and return a list of steps that can be used to reconstruct a network that is isomorphic to the original network. This deconstruction process will now be demonstrated using an example. The network on which we will apply the deconstruction process is actually unknown at the start of the deconstruction, but corresponds to a matrix of multisets of distances which we know. Each deconstruction step shown in the figure below is actually only applied on the matrix, but as the number of paths in such a network is high, displaying the matrix at each iteration will take up tons of space, and contains tons of information we will not use at a particular step. However, the matrix in each iteration corresponds to some unknown but existing network.

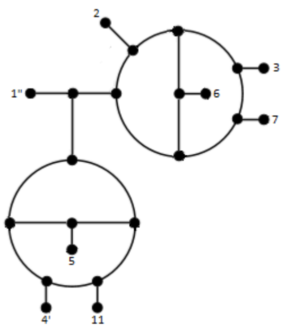
The network on 11 leaves contains 2 cherries: (1,10) and (4,9).



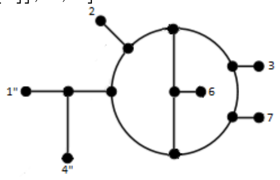
$s_1 = [[1, 10], 1, 1], \quad s_2 = [[4, 9], 4, 1]$



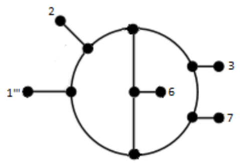
No more cherries. Network contains a pendant level-1 blob: (1', 8)
$s_3 = [[1, 8], 1, 2]$



No cherries or pendant level-1 blobs. Note that 1 is now an uncontained leaf.
Network contains a pendant kl00 blob: ((4, 11), 5)
$s_4 = [[[4, 11], [5]], 4, 4]$



The uncontained leaf is now part of a cherry (1, 4), and we contract cherries first.
$s_5 = [[1, 4], 1, 1]$



Now we've obtained one of the 4 end conditions, namely a single blob with three chains.
$s_6 = [[[1, 2], [6], [3, 7]], [], 15]$

So, the above network is fully contracted and all steps are stored into a list which will be used to reconstruct the network.

5 Reconstruction

After the deconstruction process, a list of steps is returned. As the third entry of each list is an integer value corresponding to a certain type of contraction, the list is iterated over these integers from the back to the front of the list.

5.1 Level 2 Blobs

Every pendant level-2 blob has a comparable structure. The edges in the blob all belong to one of the following types:
- Connecting a leaf to its adjacent vertex.
- Connecting two adjacent vertices of leaves in the same chain.
- Connecting endpoints of each chain in the blob to either the ‘north- ’ or ‘southpole’ of the blob; the non-cut-vertices of the blob.
- Connecting endpoints of the chains of the m and n side, or the ’north- ’ or ’southpole’ to the cut-vertex of the pendant blob.
The northpole is the vertex that connectects the k, l, and m sides of the blob.
The soutpole connectects the k, l, and n sides.
The maximum number of edges in a level 2 blob $\leq 4 + 2 \cdot (k + l + m + n) \leq 4 + 2 \cdot N$
As for the reconstruction of a level 2 blob, just adding the edges is all that needs to be done, and adding an edges takes constant time, the reconstruction of a level 2 blob is O(n).

Reconstruction klmn-blob

```
input  : Step, G, E
output: G, E
1  k, l, m, n = Step[0][0], Step[0][1], Step[0][2],Step[0][3]
2  These variables now correspond to the chains of the blob.
3  x = Step[1]
4  Rename leaf x to ”x_bc_i”, where i is the entry of the step in the steplist.
5  Add the incident edge of x_bc_i to E.
6  for $z$ = $k, l, m, n$ do
7      when adding edges, it is implied that any vertex incident to the edge that is not yet in the
       graph is added
8      Add all incident edges from the leaves:
9      for $j \in (0, ...\#z)$ do
10         G = G + $(z_j, z_j\_b\_i)$
11         if $j \neq \#z - 1$ then
12             Add all edges between the adjacent vertices of the leaves in the chain:
13             G = G + $(z_j\_b\_i, z_{j+1}\_b\_i)$
14         end
15     end
16 end
17 G = G + $(k_0\_b\_i, N\_b\_i)$
18 G = G +$(l_0\_b\_i, N\_b\_i)$
19 G = G +$(m_0\_b\_i, N\_b\_i)$
20 G = G +$(k_{-1}\_b\_i, S\_b\_i)$
21 G = G +$(l_{-1}\_b\_i, S\_b\_i)$
22 G = G +$(n_{-1}\_b\_i, S\_b\_i)$
23 G = G +$(n_0\_b\_i, x\_bc\_i)$
24 G = G +$(m_{-1}\_b\_i, x\_bc\_i)$
```

For other types of pendant level 2 blobs, this process is very similiar.

The for-loop iteration k, l, m, and n only iterates over the chains present in that type of blob, and the edges that are manually added are different.

If the l-chain is not present, then instead of the edges $(l_{-1}\_b\_i, S\_b\_i)$ and $(N\_b\_i, l_0\_b\_i)$, there is a single edge (S_b_i, N_b_i).
If the m-chain is not present, then instead of the edges $(m_{-1}\_b\_i, N\_b\_i)$ and (N_b_i, x_bc_i), there is a single edge (x_bc_i, N_b_i).
If the n-chain is not present, then instead of the edges $(n_{-1}\_b\_i, S\_b\_i)$ and (S_b_i, x_bc_i), there is a single edge (x_bc_i, S_b_i).

5.2 Level 1 Blobs

Pendant level 1 blobs do not have north- and southpoles.
The edges in these blobs are:
- Connecting a leaf to its adjacent vertex.
- Connecting two adjacent vertices of leaves in the same chain.
- Connecting the vertices adjacent to the endpoints to the cut-vertex.
Therefore, the number of edges in a Level 1 blob is equal to $1 + 2 \cdot k \leq 1 + 2 \cdot n$; where k is the length of the chain, and n is the size of the original matrix used in the deconstruction phase.
Hence, the reconstruction of a level 1 blob is also of order O(n).

```
   input  : Step, G
   output: G
 1 k = Step[0]
 2 x = Step[1]
 3 Rename leaf x to "x_bc_i", where i is the entry of the step in the steplist.
 4 Add the incident edge of x_bc_i to E.
 5 ---
 6 when adding edges, it is implied that any vertex incident to the edge that is not yet in the
     graph is added
 7 Add all incident edges from the leaves:
 8 for $j \in (0, ...\#k)$ do
 9     G = G + $(k_j, k_j\_b\_i)$
10     Add all edges between the adjacent vertices of the leaves in the chain:
11     if $j \neq \#k - 1$ then
12         G = G + $(k_j\_b\_i, k_{j+1}\_b\_i)$
13     end
14 end
15 G = G + $(k_0\_b\_i, x\_bc\_i)$
16 G = G + $(k_{-1}\_b\_i, x\_bc\_i)$
```

5.3 Cherries

Cherries are quite straightforward. The cherry deconstruction step is stored as [[x,y], x, 1]. The leaf x is in the reconstructed network, but is renamed when expanding it into a cherry. Then only the edges from the orignal leaf to x and y have to be added to the network to finish the reconstruction of the cherry. O(1).

```
   input  : Step, G
   output: G
1  Rename leaf x to "x_cc_i", where i is the entry of the step in the steplist
2  G = G + (x, x_cc_i)
3  G = G + (y, x_cc_i)
```

5.4 Uncontained Leaves

To add an uncontained leaf x to the network, a single cut-edge is removed, and replaced by 2 edges, each sharing one endpoint with the original edge, and the other endpoint being the vertex adjacent to the uncontained leaf, to which the third and last edge needs to be added. The position of the uncontained leaf is a little tricky, as it is not always on a cut-edge connecting 2 blobs. It can also be on an edge to an uncontained leaf that has been expanded into a cherry of blob. To properly determine the edge on which the uncontained leaf has to be stored, the intersection is taken from all paths from endpoints of chains in the set Y to endpoints of chains in the set Z, where Y ∪ Z = X-{x}, the partition as described in Lemma (..).
This intersection contains exactly 1 cut-edge, and therefore we remove and add the required edges on this position. An uncontained chain C^i is stored as [[Y,Z], C^i, 11], with C^i ordered such that $d_m(C^i_1, y) \leq d_m(C^i_k, y)$ for all y ∈ Y.

Uncontained Leaves

```
input : Step,
        G: the graph
        E: the set of all cut-edges
output: G
1  x, Y, Z = Step[1], Step[0][0], Step[0][1]
2  L = [ ], an empty list to store all shortest paths in.
3  for $i \in (0, .., \#Y)$ do
4      for $j \in (0, ..., \#Z)$ do
5          path = Shortest Path from $Y^i_0$ to $Z^j_0$ in G.
6          if $i == 0$ and $j == 0$ then
7              L = L ∪ path
8          end
9          else
10             L = The intersection over the edges of L and path
11         end
12     end
13 end
14 P = Empty Graph for $i \in (0, .., \#L - 1)$ do
15     P = P + (Lt(i), L(i+1))
16 end
17 for $e \in P.edges$ do
18     if $e \in E.edges$ then
19         G = G + (x, x_unc)
20         G = G + ($e_0$, x_unc)
21         G = G + ($e_1$, x_unc)
22         G = G - ($e_0, e_1$)
23         E = E + (x, x_unc)
24         return G
25     end
26 end
```

Lemma 5.1. *Given a step s, containing the partition of the leaves of the leaves as a result from its deconstruction, and the chain or leaf itself; the reconstruction of uncontained leaves and chains is done in Running Time $O(n^5)$.*

Proof. The location at which the uncontained leaf has to be placed now has to be an edge in the intersection that is not in some blob. This intersection cannot be empty as we have the partition Y

∪ Z, where Y and Z are only connected in the network by the edges of the vertex adjacent to the uncontained leaf x, but not the edge connecting these 2 vertices. Each path from any leaf in Y to any leaf in Z must traverse these edges. In the network after the uncontained leaf is removed (the current network in the reconstruction), these 3 edges are replaced by a single edge. Therefore, this is the edge on which we place the uncontained leaf, and must exist in our current reconstructed network.
At each step in the reconstruction, we have stored the non-trivial cut-edges in a list E. Therefore, the edge on which the uncontained leaf must be placed is in E. Suppose there are 2 edges in E that are also in the obtained intersection. Then each path from any leaf in Y to any leaf in Z traverses those 2 cut-edges. But each blob in the network has a leaf, so there exists a pair $y \in Y$, $z \in Z$ that traverses only 1 cut-edge, which contradicts our assumption that there were multiple edges that are traversed by all shortest paths.

Let Y ∪ Z be the partition induced by an uncontained leaf x. For one endpoint of all chains in Y, the shortest path to an endpoint of each chain in Z is calculated. Let finding the shortest path between 2 leaves be of order O(k). Then the process of finding all these shortest paths is of order $O(n^2 \cdot k)$. Each time a new shortest path is obtained, its intersection is taken with the intersection of all previously obtained shortest paths. The length of a shortest path from any 2 leaves in a level-2 binary network is upper bounded by $3 \cdot n + 1$ (which occurs when the whole network consists of exclusively level-2 blobs with at most 2 non-trivial cut-vertices each.). Hence, taking the intersection is of order $O(n^2)$, and is done within the loops over Y and Z. Checking which edge of the intersection exists in E is of order O(n), but is done after the previous steps. So the total running time is $O(n^4 \cdot k)$. There exists functions that find a shortest path in $O(V + E)$, with V the number of vertices, and E the number of edges (Breadth-first search). Both V and E are upper bounded by a constant factor on n, so O(k) corresponds to O(n), and our total running time is of order $O(n^5)$. □

5.5 Reconstruction Sequence

Reconstruction

```
input  : S, the list of deconstruction steps
output: G, the network corresponding to the Matrix of Multisets of Distances
1  G = Empty Graph
2  E = Empty Graph (Used to store Cut-Edges In)
3  for $i \in (\#S, \#S-1, ...1)$ do
4      if $S_{i_3}$ = 1 then
5          G = R_Cherry($G, S_i, i, E$)
6      end
7      if $S_{i_3} \in (2, 3, 4)$ then
8          G = R_single_chain($G, S_i, i, E$)
9      end
10     if $S_{i_3} \in (5, 6, .., 9)$ then
11         G = R_klmn($G, S_i, i, E$)
12     end
13     if $S_{i_3} \in (10, 11)$ then
14         G = R_uncontained($G, S_i, i, E$)
15     end
16     if $S_{i_3} \in (12, .., 15)$ then
17         G = R_end($G, S_i, i, E$)
18     end
19 end
```

Theorem 5.2. *Given a list S containing the deconstruction steps obtained by applying 'Deconstruction_Sequence' on the Matrix M of Multisets of Distances of the theoretical network N, the function 'Reconstruction' returns a network G isomorphic to N, and does so in theoretical Running Time of Order $O(n^3)$.*

Proof. Each deconstruction is stored using a unique ID for that type of deconstruction, and each blob is always stored in the same way. As every blob, cherry, and uncontained leaf is added to the network on a unique position, and all steps in the steplist are iterated, the resulting network G is isomorphic to the original theoretical network N corresponding to the Matrix of Multisets of Distances M.
All reconstruction steps are done sequentially, and there are at most n reconstruction steps that need to be done, the total running time of the deconstruction is n times the maximum over all individual running times of the reconstruction steps: $O(n^6)$. □

Now everything that is needed to prove the main theorem is handled and proven, which allows us to proof the main theorem of this paper:

Theorem 5.3. *There exists a $O(n^3 \cdot D)$-algorithm that reconstructs a network N based on the multisets of its distances.*

Proof. Let M be a matrix of multisets of distances for a (Level-2 unrooted, unweighted, undirected Binary) Network N. By Theorem 4.19, it is known that we can take this matrix and turn it into a list of deconstruction steps. An by Theorem 5.2, this list of steps can be utilized to construct a network G that is isomorphic to the network N.
As on every iteration, the deconstruction steps are called in the same order; if the algorithm is run multiple times on the same network (such that the indices of the leaves in the matrix is the same every time), it returns the same list of deconstruction steps every time. As the reconstruction iterates over this list back to front, the resulting network is the same at each iteration.
Let M_1 and M_2 be two different matrices for a network N. That is, there exist at least 2 leaves in

N whose indices in M_1 is different from their indices in M_2. As every iteration in the deconstruction step iterates from the lowest index to the highest, it might occur that the list of deconstruction steps for M_1 is different than that of M_2. However, by Theorems 4.19 and 5.2, in both cases, the resulting networks G_1 and G_2 are isomorphic to N. Therefore G_1 is isomorphic to G_2. So a network N will always be reconstructed and deconstructed to an isomorphic network G regardless of how the leaves are indexed in the matrix of multisets.
Finally, as the deconstruction and reconstruction are done sequentially, the running time order of the whole process is the maximum of the individual running time order of both individual algorithms, which is $O(N^3 \cdot D)$. $\square$

6 Running Time

		Deconstruction	Reconstruction
1.	Cherry	$O(n \cdot D)$	$O(1)$
2.	Chain	$O(n^2)$	-
3.	End	$O(1)$	$O(n)$
4.	Single Chain	$O(n \cdot D)$	$O(n)$
5a.	Uncontained Leaf	$O(n^2 \cdot D)$	$O(n^2)$
5b.	Uncontained Chain	$O(n^2 \cdot D)$	$O(n^5)$
6.	1000-Blob	$O(n \cdot D)$	$O(1)$
7.	kl00-Blob	$O(n \cdot D)$	$O(n)$
8.	klmn-Blob	$O(n \cdot D)$	$O(n)$
9.	Adjust	$O(n)$	-

The Running Times are per cherry/blob

n	m	Phase	mean	var	q0.05	q0.25	q0.50	q0.75	q0.95
10	10000	s	0,20858	0,10893	0,032000	0,07800	0,125	0,21900	0,625
		d	0,02642	9,328e-05	0,015000	0,01600	0,03100	0,03100	0,04700
		r	0,02642	0,00010	0,015000	0,01500	0,03100	0,03100	0,04700
12	10000	s	0,65252	1,38699	0,08500	0,17799	0,32999	0,65400	2,14299
		d	0,03530	0,00023	0,01600	0,02799	0,03199	0,04099	0,06200
		r	0,03400	0,00016	0,01599	0,03099	0,03100	0,03800	0,05400
14	7500	s	1,69134	65,02822	0,16300	0,36900	0,71700	1,45875	5,43799
		d	0,05247	0,44657	0,02199	0,03100	0,03800	0,04700	0,06900
		r	0,03833	0,000143	0,01600	0,03100	0,03799	0,04699	0,05400
16	3000	s	3,55094	93,36329	0,27989	0,69950	1,40149	3,13775	11,62060
		d	0,05875	0,03646	0,02999	0,03400	0,04600	0,05299	0,07800
		r	0,041148	0,00012	0,02699	0,03199	0,04100	0,04699	0,06200
18	3000	s	8,87849	1544,423	0,56200	1,34299	2,960499	6,86099	29,90450
		d	0,17152	16,95976	0,03099	0,04699	0,04700	0,06299	0,14000
		r	0,046445	0,00021	0,03099	0,03200	0,04700	0,04700	0,06299
20	1500	s	15,97809	5608,236	0,76424	1,93700	4,77999	12,06400	46,64870
		d	0,69705	279,3856	0,03099	0,03200	0,04670	0,06270	0,21975
		r	0,04294	0,00145	0,03010	0,03100	0,04670	0,04700	0,06300
22	400	s	42,08291	13132,050	1,62419	5,20549	11,6300	32,4489	153,6444
		d	0,63815	20,48602	0,046000	0,06200	0,07799	0,10999	0,95735
		r	0,058317	0,000233	0,031000	0,046999	0,062000	0,06299	0,07899
24	250	s	65,71547	71272,348	1,79269	6,51650	16,09199	35,68174	219,22359
		d	2,68403	923,79420	0,04799	0,06300	0,078999	0,13950	1,12590
		r	0,065887	0,00024	0,044000	0,05399	0,06299	0,07799	0,09399

n: the number of leaves in the network.
m: the number of networks created, de- and reconstructed.
s: the setup phase; creating the network and multisets.
d: the deconstruction phase.
r: the reconstruction phase.

In Figure 3, one can observe in similarities in the shape of the histogram for the setup- and deconstruction times. As both the setup- and deconstruction time are based on the sizes of the multisets of distances, this is what one would expect. A network that takes a long time to setup, takes such a long time because all paths are iterated for each leaf to each other leaf when the multisets are initialized. These same entries are each adjusted in the deconstruction. In the reconstruction of the

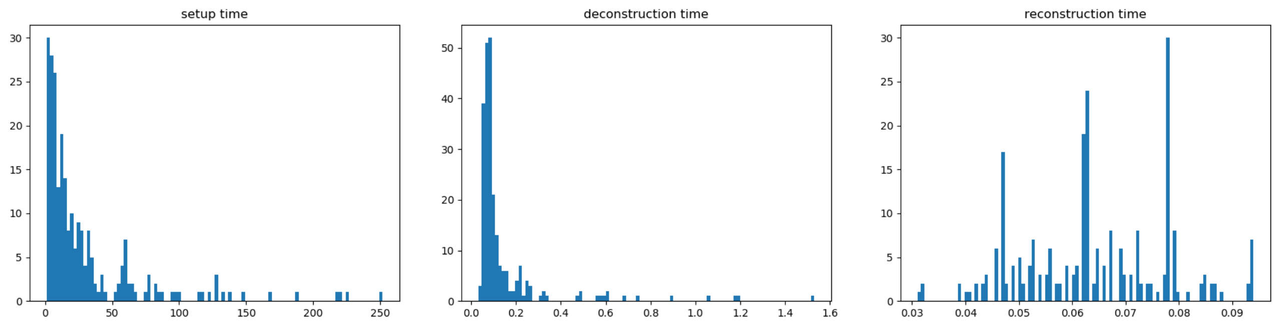


Figure 3: Running times for 250 iterations of a 24-leaf network.(largest 3 entries removed)

network, this no longer has a large impact.

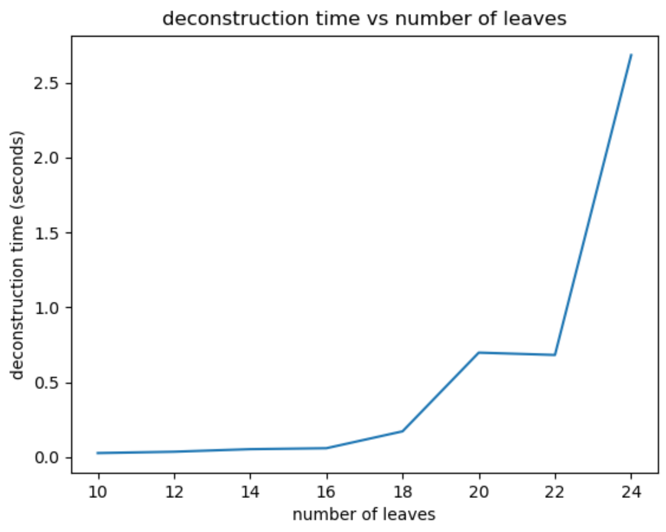


Figure 4: The mean of the deconstruction times versus the number of leaves

A seemingly odd result in the deconstruction times can be observed. The mean of the deconstruction times for 20 leaves is slightly higher than that of the networks on 22 leaves. This can be explained by the low variation in the test results for n=22 in comparison to that of n=20. However, if one looks at the 0.95 quantile of both datasets, it can be seen that the value for 22 leaves is significantly higher than that for 20 leaves (as one would expect). Both observations indicate that there are a handful of networks on 20 leaves that took a long time to be deconstructed due to the sizes of their multisets of distances, which, as there are almost 4 times as many networks tested on 20 leaves than on 22 leaves, is plausible.

As the setup time takes exponentially longer as the number of leaves is increased, the largest network on which the algorithms are tested (to analyze running times) is set to 24.
However, there are no structures that arise in a network on 50, 100 or more leaves that will not occur in a network on 24 or fewer leaves.

	$\frac{d}{s}$		$\frac{r}{s}$		$\frac{r}{d}$	
n	mean	var	mean	var	mean	var
10	0,24132	0,03849	0,24511	0,036709	1,08148	0,16885
12	0,12367	0,01042	0,12356	0,01095	1,0207	0,11078
14	0,07048	0,003643	0,06887	0,003767	0,97231	0,08317
16	0,04280	0,00158	0,04054	0,00154	0,91107	0,05461
18	0,02451	0,00060	0,02298	0,00062	0,87508	0,07227
20	0,01607	0,00301	0,01435	0,00026	082814	0,08462
22	0,01028	9,43381e-05	0,00822	9,5017e-05	0,71951	0,10512
24	0,01041	0,000167	0,00769	9,84318e-05	0,69381	0,09082

The most important results are the deconstruction and reconstruction times.
The reconstruction time is very fast regardless of the complexity of the network. This is because the only thing that is done is adding edges to the graph based on the step list, except for the uncontained leaves. For these, the intersection of shortest paths from leaves in Y to leaves in Z need to be taken in order to determine where in the network the uncontained leaf should be placed. The increase in running time for the reconstruction is approximately linear in practice, although the theoretical running time is quadratic. This is because uncontained leaves do not occur very frequently in networks on which the algorithm is tested, as the frequency of these leaves in the random network generator is set to be low, as a phylogentic network for which this algorithm is designed generally has few uncontained leaves.
The practical running time of the deconstruction does increase at a non-linear rate. This is also what one would expect looking at the theoretical running time. The sizes of the multisets of distances exponentially increase as we increase the number of leaves in the network. And at each contraction a lot of these multisets are adjusted. So both the sizes of the multisethats increase and the number of those larger multisets that need to be adjusted increase. Observe the running times for n=24. The mean of the deconstruction times is approximately 2.68 seconds, but the 0.95 quantile is 1.125. This indicates that there are a handfull of networks that take quite long to run and pump up the average running time. These networks resulting in a longer running time are those with a large number of blobs, and thus larger multisets. In phylogenetics, the networks are usually more compact, and the expected running time for these should be on the lower side.

7 Discussion

Although the algorithm runs in theoretical polynomial time, as D, the maximum over the size of all multisets of distances, is upperbounded by 4^n, this process is not of polynomial time in a practical sense. If we can uniquely reconstruct Level-2 binary networks based on their shortest and longest distances, then it is likely to have a practical polynomial running time. A lot of Lemma's use minimal distances only. There are only a handful that actually use the whole multiset, but in these cases, it checks if the multiset is equal to some set with 3 or 4 entries. A lot of exponentially large multisets are therefore stored and splitted multiple times, and are only used in a small number of deconstruction steps. Only using the minimal and maximal distance, every multiset has either 1 or 2 entries, hence it takes constant time to update these distances, and $O(n^2)$ to update each multiset in the matrix. Hence, it is likely that an algorithm using only the minimal and maximal distances will run in polynomial time with respect to the number of leaves as the only parameter.
Currently, the only contractions that use the multisets are single chain blobs and level 2 pendant blobs. kl00 blobs can also be identified based on the shortest distances of the leaves in the network ([1]), but when using distance multisets, it is quicker to only use the distance multisets to identify such a blob.

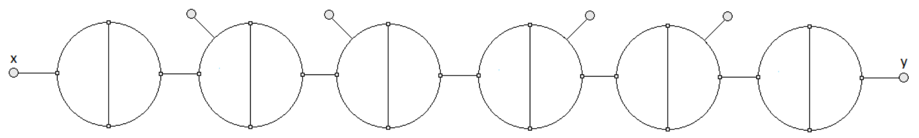


Figure 5: The network on 6 leaves with the largest multiset of distances possible between 2 leaves x and y.

The maximum size of the multiset (4^n), is obtained when the whole network consists of level 2 blobs with a single leaf, with only 2 of them pendant. (See figure 5) The networks frequently used in phylogenetics are more compact. The leaves are 'closer' to each other, with most of them contained in some chain with more than one leaf. Consider the network used to demonstrate the deconstruction process on page 32. In figure 5, the number of paths from x to y equals $4^6 = 4086$. The largest number of paths for any pair of leaves in the deconstruction example equals 16, and that network consists of 11 leaves.
In a practical sense, it is safe to assume that the running time of the reconstruction and deconstruction is done faster than the 0.95 quantile for the test data on that number of leaves, which on its turn is usually quite smaller than the corresponding mean (because of a handful of large outliers).

Moreover, revisiting some deconstruction steps after assessing the running times, ways to improve the algorithm have been found. For example, an uncontained chain can be identified based on the number of paths between its leaves. That is, if c is a chain of uncontained leaves, each leaf has exactly 1 path to each other leaf in the chain. Including these changes in the algorithm requires re-running the algorithm on the bulk of networks to properly adress the improvement on the practical running time.

Finally, this algorithm focuses on unweigthed and undirected graphs, therefore the resulting network is a simplified picture of the actual evolution of species. This does not mean the algorithm provides no useful information, but it is incomplete. Including these extensions in the algorithm is not likely a trivial matter, as for all lemmas used in the deconstruction, the assumption is made that the network is unweighted and undirected. Hence, the lemma do no longer hold, and need to be either adjusted or even completely reworked. Additionally, the algorithm assumes that the input matrix corresponds to some Level-2 binary network and will not properly run if the input matrix corresponds to a network of a higher level, or one that is non-binary. Therefore some additional theory and implementation should be done to be able to deconstruct and reconstruct phylogenetic networks. However, level 2 binary networks is already an improvement over the traditional tree structure, and therefore serves as a nice frame to extend upon.

8 Appendix

8.1 General Pseudocodes

Adjacent

```
input  : M, the matrix of multisets of distances
         C, D, two chains
output: n
if $\#C == 1$ and $\#D == 1$ then
    n = $d^N(C_1, D_1)$.count(4)
    return n
end
if $\#C > 1$ and $\#D == 1$ then
    n = $(d_m(C_1, D_1) == 4) + (d_m(C_k, D_1) == 4)$
end
if $\#C == 1$ and $\#D > 1$ then
    n = $(d_m(C_1, D_1) == 4) + (d_m(C_1, D_l) == 4)$
end
if $\#C > 1$ and $\#D > 1$ then
    n = $(d_m(C_1, D_1) == 4) + (d_m(C_1, D_l) == 4 + (d_m(C_k, D_1) == 4) + (d_m(C_k, D_l) == 4))$
end
return n
```

min_distance_largerthan_k

```
input  : M, C, c, k
output: boole
for $d \in C - \{c\}$ do
    for $y \in (1, k_d)$ do
        if $d_m(d_y, c_1) \leq k$ then
            return False
        end
    end
end
return True
```

cross_distance_check

```
input  : M, the matrix of multisets of distances
         a, y, z, a triplet of leaves
         k, an integer
output: boole
if $d_m a, y + d_m a, z - d_m y, z \geq k$ then
    return True
end
return False
```

create_submatrices

```
input  : M, the matrix of multisets of distances
         R, a matrix of chains
output: S, a list containing the submatrices,
        X, a list containing the set of chains for each submatrix.
1  L = A list of all chains in R.
2  X = [[$R_1$]],    $L = L - R_1$
3  for $c \in L$ do
4      for $s \in X$ do
5          if $R_{c,s} > 0$ then
6              $s = s \cup c$,    $L = L - c$
7          end
8      end
9  end
10 for $s \in X$,    If $\#s = 1$ do
11     X = X - {s}
12 end
13 Create an empty list S to store the submatrices in.
14 for $s \in X$ do
15     Create matrix $m$ of size $\#s \cdot \#s$
16     Fill $m$ based on adjacency values in R
17     $S = S \cup m$
18 end
19 Return S, X
```

8.2 Seeds

n	$seed_{min}$	$seed_{max}$
10	0	9999
12	10000	19999
14	20000	27499
16	27500	30499
18	30500	33499
20	33500	34999
22	35000	35399
24	35400	35649

References

[1] Van Iersel L., Moulton V., Murakami Y (2020) *Reconstructibility of unrooted level-k phylogenetic networks from distances.*

[2] Bapteste E., Van Iersel L., Janke A., Kelchner S., Kelk S., McInerney J. O., Morrison D. A., Nahkleh L., Steel M., Stougie L., Whitfield J. (2013) *Networks: expanding evolutionary thinking.*