

Recognizing Circular Decomposable Metrics

Martin Farach*

March 11, 1997

1 Introduction

In [BD92], Bandelt and Dress introduced a class of metrics called the *circular decomposable metrics* (CDMs). In [DDvH93], Dopazo, Dress and von Haeseler demonstrated the utility of CDMs for phylogenetic studies by applying the metric techniques of In [BD92] to distance data derived from viral DNA sequences.

We show an $O(n^2)$ time algorithm for recognizing CDM metrics. The only algorithm known for this problem solves a more general problem, but when applied to this problem runs in time $O(n^5)$ [BD92]. In [CFT96], we showed that CDM metrics are the same class as Kalmanson metrics. Since an $O(n^2 \log n)$ time algorithm was the best known algorithm for recognizing this metric, we also improve that result.

It is unlikely that any measured data would ever exactly adhere to the definition of circular decomposability. The approach taken in [DDvH93] was to use the algorithm in [BD92] to try to find a CDM which is close to the input data. Their algorithm takes as its input any metric, and it returns a so-called *split prime residue*, which when subtracted from the metric yield a totally decomposable metric. It is then straightforward to decide if this totally decomposable metric is a CDM. In this sense the Bandelt and Dress tried to suggest a *fitting* algorithm which returns a CDM which is closely related to the input distance function. This approach to model fitting is different from that traditionally taken when fitting data to particular metrics. There, the goal is typically to modify the input function as little as possible so that it fits the target class.

As noted above, in this paper we are concerned with a much more modest goal. We would like to simply recognize when we have a CDM. On the one hand, this means that our algorithm will not be directly applicable to data. On the other hand, our algorithm relies on an elegant transformation which maps CDMs into simpler metrics, and may prove valuable in designing fitting algorithms which minimally modify the input matrix to yield CDMs.

We begin by defining terms. We then describe our recognition algorithm.

*Department of Computer Science, Rutgers University; farach@cs.rutgers.edu; <http://www.cs.rutgers.edu/~farach>; Supported by an NSF Career Advancement Award and an Alfred P. Sloan Research Fellowship.

2 Definitions and Notation

We will define many types of metrics on n points. For all these, we will take $B = \{1, \dots, n\}$ to be the point set of the metrics.

Definition 2.1 A metric on a set B is a function $D : B^2 \rightarrow \mathfrak{R}_{\geq 0}$ such that $\forall i, j, k \in B$

- $D[i, j] = 0 \iff i = j$
- $D[i, j] = D[j, i]$
- $D[i, j] \leq D[i, k] + D[k, j]$ (triangle inequality).

When the first condition is weakened to merely require that $\forall i \in B, D[i, i] = 0$, we refer to D as a pseudo-metric.

Definition 2.2 A cut pseudo-metric $D_T, T \subseteq B$, is defined by

$$D[i, j] = \begin{cases} 1 & \text{for } i \neq j \text{ and } |\{i, j\} \cap T| = 1 \\ 0 & \text{otherwise} \end{cases}$$

A weighted cut pseudo-metric is a pseudo-metric of the form αD_T for a positive scalar α and cut metric D_T .

Note that a cut metric is defined by T is the same cut metric is defined by $\bar{T} = B/T$.

Definition 2.3 A metric D is decomposable if it can be written as the sum of weighted cut metrics.

Definition 2.4 A decomposable metric D is a Circular Decomposable Metric (CDM) if the elements of B can be ordered $v_1 \dots v_n$ so that every cut is defined by a set of the form v_i, v_{i+1}, \dots, v_j . We denote such a subrange cut by $\langle i, j, \alpha \rangle$, where α is the weight of the subrange cut defined by i and j . Any ordering $v_1 \dots v_n$ satisfying the subrange constraints is said to be valid for the metric D .

A valid ordering $v_1 \dots v_n$ for a metric defines a total ordering which we denote $<_v$; that is, $v_1 <_v v_2 <_v \dots <_v v_n$.

Definition 2.5 A metric D is monotone if there is some ordering $v_1 \dots v_n$ of the elements of B such that, for all $i <_v j <_v k$, $D[i, j] \leq D[i, k]$ and $D[j, k] \leq D[i, k]$. Any such ordering $v_1 \dots v_n$ is said to be valid for the metric D .

Definition 2.6 A monotone metric D with valid ordering $v_1 \dots v_n$ is strongly monotone if for all $i <_v j <_v k <_v l$, $D[j, k] = D[j, l] \Rightarrow D[i, k] = D[i, l]$ and $D[j, k] = D[i, k] \Rightarrow D[j, l] = D[i, l]$.

If two monotone (resp. CDM) metrics share a valid ordering, then we say they are *consistent*.

We now state the **CDM Recognition Problem** as:

Input: A distance function $M : B^2 \rightarrow \mathfrak{R}_{\geq 0}$.

Output: A ordering $v_1 \dots v_n$ of B and a set C of subrange cuts on $v_1 \dots v_n$ realizing M , if such exist.

We will give an algorithm which first finds the ordering and then the cuts, each in time $O(n^2)$, thus giving an $O(n^2)$ time algorithm for this problem.

3 Finding the Ordering

We solve the problem of finding a valid ordering for M by demonstrating a mapping which takes CDMs into strongly monotone metrics with the same valid orderings. We then show how to find a valid ordering for a strongly monotone metric.

3.1 Mapping a CDM into a Strongly Monotone Metric

Fix $v_1 \dots v_n$ to be some valid ordering for M and let M_1, \dots, M_k , $M_i = \langle l_i, r_i, \alpha_i \rangle$, be a decomposition of M into its constituent weighted cut pseudo-metrics. Thus $M[x, y] = \sum_{i=1}^k M_i[x, y]$.

Let $m_{v_1} = \max_i \{M[v_1, i]\}$. Notice that $v_1 \dots v_n$ can be cyclically permuted to produce another valid ordering for M , so our choice of v_1 is arbitrary. For each M_i , define C_i as

$$C_i[x, y] = \begin{cases} 2m_{v_1} - M_i[a, x] - M_i[a, y] & x \neq y \\ 0 & \text{otherwise.} \end{cases}$$

Set $S_i = M_i + C_i$.

Now we can simplify S_i as follows. Assume $v_1 \in [l_i, r_i]$.

$$S_i[x, y] = \begin{cases} 2m_a & |\{x, y\} \cap [l_i, r_i]| < 2 \\ 2m_a - 2\alpha_i & \text{otherwise.} \end{cases}$$

The case where $v_1 \notin [l_i, r_i]$ is similar.

Lemma 3.1 *For $1 \leq i \leq k$, S_i is strongly monotone. $v_1 \dots v_n$ is a valid ordering for S_i .*

Proof: The proof is a straightforward case analysis. ■

Let $\hat{S} = \sum_{i=1}^k S_i$. Notice that $\hat{S}[x, y] = M[x, y] + \sum_{i=1}^k C_i[x, y] = M[x, y] + \sum_{i=1}^k (2m_{v_1} - M_i[a, x] - M_i[a, y]) = M[x, y] + 2km_{v_1} - M[a, x] - M[a, y]$. So \hat{S} can be computed directly from M without knowing the decomposition of M into its cuts. All we need to know is k . Instead, simply consider all subrange cuts which do not occur in our decomposition to be cuts of weight O . Then $k = \binom{n}{2} + n$. Therefore, define S by $S[x, y] = M[x, y] + 2(\binom{n}{2} + n)m_{v_1} - M[a, x] - M[a, y]$, which can be computed directly for any metric M , even if M is not decomposable and therefore has no decomposition into cuts.

We show that S is strongly monotone by the following.

Lemma 3.2 *If M_1 and M_2 are strongly monotone metrics with a consistent ordering $v_1 \dots v_n$, then $M' = M_1 + M_2$ is a strongly monotone metric with valid ordering $v_1 \dots v_n$.*

Proof: Take any $i <_v j <_v k <_v l$. Then trivially, $M'[i, j] \leq M'[i, k]$ and $M'[j, k] \leq M'[i, k]$. Now suppose $M'[j, k] = M'[j, l]$. Then $M_1[j, k] + M_2[j, k] = M_1[j, l] + M_2[j, l]$. Since $M_1[j, k] \leq M_1[j, l]$ and $M_2[j, k] \leq M_2[j, l]$, then $M_1[j, k] = M_1[j, l]$ and $M_2[j, k] = M_2[j, l]$. So, by the strong monotonicity of M_1 and M_2 , $M_1[i, k] = M_1[i, l]$ and $M_2[i, k] = M_2[i, l]$. We conclude that $M'[i, k] = M'[i, l]$, which finishes the proof. ■

So any valid ordering of M yields a valid ordering of S , since the values of S do not depend on $v_1 \dots v_n$. Finally, we establish that it suffices to find a valid ordering of S .

Lemma 3.3 *If $v_1 \dots v_n$ is a valid ordering of S then it is a valid ordering of M .*

Proof: We proceed by induction. Suppose $n = 4$. We prove the base case by contradiction. Let $v_1 \dots v_n$ be a valid ordering for S and $v'_1 \dots v'_n$ be a valid ordering for M . Assume, w.l.o.g, that $1 <_v 2 <_v 3 <_v 4$.

Notice that any cut of the form $\langle i, i, \alpha \rangle$ has no effect on the value of S . So we will ignore all such singleton cuts. Then the only cuts we need consider are cuts which separate one pair from the other, since all other cuts are singletons. If $1 <_{v'} 2 <_{v'} 3 <_{v'} 4$ is not a valid ordering of the points for M , it must be because M contains a cut separating 1 and 3 from 2 and 4. Call the weight of such a cut α .

Assume, that $1 <_{v'} 3 <_{v'} 2 <_{v'} 4$. The other cases are symmetric. Let α' be the (possibly 0) weight of the cut separating 1 and 3 from 2 and 4. Notice that the pivot chosen to map M to S was either 1 or 4. By symmetry, we assume it was 4. Then $S[1, 2] = M[1, 2] + 6m_a - M[1, 4] - M[2, 4] = 6m_a$. $S[1, 3] = 6m_a - 2\alpha$, and $S[1, 4] = 6m_a$. Since $\alpha > 0$, then 1, 2, 3, 4 cannot be a valid ordering for S , giving a contradiction and proving the base case.

As for the inductive hypothesis, notice that the set of valid orderings for M are all those which are consistent with the valid orderings on all quartets. The same is true for S . But by induction, this is the same set of orderings, thus establishing our lemma. ■

Thus, S is strongly monotone. We now show how to find a valid ordering of S .

3.2 Finding a valid ordering for a monotone metric

We can think of the symmetric function S as defining an undirected weighted graph. Throughout this subsection, we will use this graph interpretation both implicitly and explicitly. To begin with, by the monotonicity of S , a minimum weight Hamiltonian path of S yields such a valid ordering of S . It remains, therefore, to show how to find such a Hamiltonian path. We start as follows:

Lemma 3.4 *A minimum weight Hamiltonian path P of a strongly monotone metric S with valid ordering $v_1 \dots v_n$ is also a minimum spanning tree of S .*

Proof: Suppose we have a minimum size counter-example of the claim. Let T be an MST for the metric. Then all edges of T and P must be distinct, else we could identify the endpoints of the shared edges and have a smaller counter-example. Let (i, j) be the smallest weight edge in P . Consider the path from i to j in T . It must contain some edge (k, l) such that $k \leq_v i$ and $l \geq_v j$. Now $S[i, j] \leq S[k, l]$ by the monotonicity of S , and we can replace (k, l) with (i, j) , yielding tree T' . Either $S[i, j] < S[k, l]$, yielding a contradiction to T being an MST, or we can take T' and P and identify the nodes i and j , thus yielding a smaller counterexample. Therefore P is an MST of S . ■

Our algorithm will follow somewhat the outline of Kruskal's Algorithm [AHU74] for MST construction. In Kruskal's algorithm, we sort the edges and add edges to the MST from lightest to heaviest, as long as doing so does not create a cycle. In our case, we cannot afford to sort the edges, and we must be careful how we add edges so that not only are cycles not created, but the MST is a path.

We start by setting $m = \min_{i \neq j} \{S[i, j]\}$. Let $G = (V, E)$ be defined by $V = \{1, \dots, n\}$, $E = \{\{i, j\} | S[i, j] = m\}$. Let C_1, \dots, C_i be the connected components of G . Now, by monotonicity,

the nodes of the C_i must appear consecutively in any minimum weight Hamiltonian tour of S . Thus we need to decide how to order the nodes of each C_i . If C_i has one or two nodes, then its nodes are trivially ordered. We therefore concentrate on C_i with at least three nodes.

We order each $C_i = (V_i, E_i)$ in two steps: first we find which of the nodes in V_i should begin and end the subtour on C_i ; then we order the nodes between the endpoints. For each $v_j \in V_i$, let $m_j = \min_{v \notin V_i} \{S[v_j, v]\}$, and let $N_j = \{v | v \notin V_i, S[v_j, v] = m_j\}$.

Pick any $v_j, v_k \in V_i$. By strong monotonicity, $N_j = N_k$ or $N_j \cap N_k = \emptyset$. We can therefore represent each N_j by n_j , the smallest index member of N_j . Some of these n_j will lie to the right of C_i in a valid ordering, and some to the left. Therefore, there can be at most two distinct such n_j . Call then n_l and n_r . By monotonicity, a node $n_j \in V_i$ can only be a left (right) endpoint if its distance to n_l (n_r) is minimum over all nodes in C_i . We therefore modify C_i as follows.

1. Find n_l and n_r (possibly equal).
2. Create a pair of nodes n'_l, n''_l and another pair n'_r, n''_r . Connect n'_l to n''_l and n'_r to n''_r .
3. Connect n''_l (resp. n''_r) to all nodes in C_i which minimize the distance to n_l (resp. n_r).

Now n'_l and n'_r are the new end points of C_i . When ordering the nodes in C_i , we need no longer consider any effect of distances to the points outside of C_i . We apply the following lemma to this modified C_i .

Lemma 3.5 *If S is strongly monotone, then each C_i is an interval graph. Any valid ordering of C_i as an interval graph is also a valid subtour of the minimum weight Hamiltonian tour for S .*

Proof: Any Hamiltonian path in C_i gives an MST since G has only minimum weight edges. An interval graph ordering gives a Hamiltonian path in G . So we need to show that C_i is an interval graph. Let v_1, \dots, v_k be the order of the nodes of C_i in some valid ordering of S . By monotonicity, those nodes at distance m from v_i must form an interval around v_i . Thus C_i is an interval graph. ■

This shows how to order the nodes of any C_i by finding an interval graph embedding. This can be done in time linear in the number of edges in $|C_i|$ [BL76]. Once this is done, we can collapse the nodes of C_i into a single node V_i . We set the distance $S[V_i, v] = \min_{n_j \in V_i} \{S[n_j, v]\}$. At the end of the computation, we need to determine which orientation the tour of V_i must appear in. But this can be determined in constant time by checking the distance of the endpoints of the tour to some reference point, say, the neighbors of the tour on either side.

Once we are done with this stage, in which all nodes of weight m have been processed, we can proceed to the next stage of the algorithm. Notice that the smallest weight in the updated S will be strictly greater than m . The algorithm is correct, by the correctness of Kruskal's algorithm, and by Lemma 3.5, which says that we are handling batches of minimum weight edges correctly. The remaining question is the time complexity.

In order to implement this procedure in $O(n^2)$ time, we need a method to find the minimum weight remaining edge in S , and we must show how to construct the C_i . Since each C_i is processed only once, we can afford quadratic time $O(n|V_i| + |V_i|^2)$. This is enough time to build the C_i and to find its endpoints, as long as we have a list of the vertices of C_i . This also gives us enough time to order the vertices of C_i , as noted above.

Notice that at each stage, we get rid of at least one node, so we can afford $O(n)$ book keeping time per round. At each stage, each node keeps track of its minimum connections to other nodes, by keeping a list of its minimum distance neighbors. When the nodes of C_i are collapsed, we can update this list for the new node in $O(n|V_i|)$ time.

Finally, in each round, we start by finding in $O(n)$ time the minimum weight edge, by scanning the minimums of all the remaining nodes, and put in a list all nodes which have a neighbor at this globally minimum distance. Now we can actually construct the C_i in $O(n|V_i|)$ time.

Theorem 3.6 *The CDM ordering problem can be solved in $O(n^2)$ time.*

4 Realizing the Cuts

Let $v_1 \dots v_n$ be a valid ordering of some CDM M . We will call a cut $\langle l, r, \alpha \rangle$ an *internal cut* if $l \neq 1$ and $r \neq n$. Otherwise, we will call it a *prefix cut*. Any cut $\langle l, n, \alpha \rangle$ will instead be represented as the equivalent cut $\langle 1, l - 1, \alpha \rangle$.

The problem we solve is the *Cut Realization Problem*, defined as follows.

Input: A CDM M and a linear ordering $v_1 \dots v_n$ of the points of M .

Output: A set C of triples $\langle l_i, r_i, \alpha_i \rangle$ such that $l_i \leq r_i$, $\alpha_i > 0$ and such that $\forall x < y$

$$M[x, y] = \sum_{|\{k, l\} \cap [l_i, r_i]|=1} \alpha_i,$$

i.e. C realizes M .

Our algorithm will proceed iteratively by finding the cuts in each suffix of the linear ordering. Suppose we have found all cuts in a suffix $v_{i+1} \dots v_n$. We wish to find all cuts in $v_i \dots v_n$. Notice that if $\langle j, k, \alpha \rangle$ is an internal cut of $v_{i+1} \dots v_n$, then it is an internal cut of $v_i \dots v_n$. We will therefore focus only on prefix cuts of $v_{i+1} \dots v_n$. Some of these will become internal cuts of $v_i \dots v_n$ and some will have to be extended to be prefix cuts of $v_i \dots v_n$. More generally, if $\langle i + 1, k, \alpha \rangle$ is a cut of $v_{i+1} \dots v_n$, then there will be two cuts of $v_i \dots v_n$, namely $\langle i, k, \alpha' \rangle$ and $\langle i + 1, k, \alpha'' \rangle$, $w', w'' \geq 0$, $\alpha' + \alpha'' = \alpha$.

We first eliminate all effects of internal cuts on the distances from v_i to v_k , $k > i$. We will do so by reference to an array I , which is initialized to 0. For every internal cut $\langle j, k, \alpha \rangle$, we set $I[j] = I[j] + \alpha$ and $I[k + 1] = I[k + 1] - \alpha$. Since no internal cut can end at n , $k + 1 \leq n$. We can now eliminate the effect of internal cuts by setting $M'[j] = M[i, j] - I^*[j]$, $i < j \leq n$. Here, for any array $I[1, n]$, we will take $I^*[1, n]$ to be the *prefix sum* of I , that is $I^*[1] = I[1]$, $I^*[k] = I^*[k - 1] + I[k]$, $1 < k \leq n$. While there may be many internal cuts, and so computing I may be expensive, we can reuse I from stage to stage of the algorithm, updating as above each time we find a new internal cut. Then at any stage, we will only need to compute I^* from I .

Let $P[k] = \alpha$, $k > i$, if $\langle i + 1, k, \alpha \rangle$ is a prefix cut of $v_{i+1} \dots v_n$. Given P and M' , we wish to add v_i to the ordering, while updating P and I , and outputting any new internal cuts we find. We do so by means of the following lemma.

Lemma 4.1 *If $\langle i + 1, k, P[k] \rangle$ is a prefix cut of $v_{i+1} \dots v_n$, then $\langle i, i, M'[n] - P^*[n] \rangle$, $\langle i, k, \alpha' \rangle$ and $\langle i + 1, k, \alpha'' \rangle$ are cuts of $v_i \dots v_n$, where*

$$\alpha' = \frac{P[k] + M'[k + 1] - M'[k]}{2}$$

$$\alpha'' = \frac{P[k] - M'[k + 1] + M'[k]}{2}.$$

Proof: As noted above, $\alpha' + \alpha'' = P[k]$. Consider any cut ending at $l < k$. Any such cut $\langle i, l, \alpha \rangle$ increments $M[i, k]$ and $M[i, k + 1]$ by α . Conversely any such cut $\langle i + 1, l, \alpha \rangle$ increments $M[i, k]$ and $M[i, k + 1]$ by α if $l > k$. So the only difference between $M'[k]$ and $M'[k + 1]$ must be due to cuts $\langle i, k, \alpha' \rangle$ and $\langle i + 1, k, \alpha'' \rangle$. We conclude that $M'[k + 1] - M'[k] = \alpha' - \alpha''$. Solving for α' and α'' give the desired values. Finally, a cut at singleton i may change all distance from i to a $j > i$ by a constant. This constant is therefore $M'[n] - P^*[n]$. ■

So in $O(n - i)$ we can update P and I , thus giving a $O(n^2)$ time algorithm for the Cut Realization problem.

Finally,

Theorem 4.2 *The CDM Recognition Problem can be solved in $O(n^2)$ time.*

Proof: The only thing we need to check is that the answer we get at the end really does correspond to our original metric M , but this is easily done in $O(n^2)$ time. ■

References

- [AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [BD92] H-J. Bandelt and A.W.M. Dress. A canonical decomposition theory for metrics on a finite set. *Advances in Mathematics*, 92:47–105, 1992.
- [BL76] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.
- [CFT96] G. Christopher, M. Farach, and M. Trick. The structure of circular decomposable metrics. *Proc. of the 4th Annual European Symposium on Algorithms*, pages 486–500, 1996.
- [DDvH93] Joaquin Dopazo, Andreas Dress, and Arndt von Haeseler. Split decomposition: A technique to analyze viral evolution. *Proceedings of the National Academy of Science*, 90:10320–10324, 1993.