Acknowledgements

Vaggos Chatziafratis was partially supported by ONR grant N00014-17-1-2562. Rad Niazadeh was supported by Stanford Motwani fellowship. Moses Charikar was supported by NSF grant CCF-1617577 and a Simons Investigator Award. We would also like to thank Leo Keselman, Aditi Raghunathan and Yang Yuan for providing comments on an earlier draft of the paper. We also thank the anonymous reviewers for their helpful comments and suggestions.

A. Supplementary Materials

A.1. Real-world application example.

In phylogenetics, which is the study of the evolutionary history and relationships among species, an end-user usually has access to whole genomes data of a group of organisms. There are established methods in phylogeny to infer similarity scores between pairs of datapoints, which give the user the similarity weights w_{ij}. Often the user also has access to rare structural footprints of a common ancestry tree (e.g. through gene rearrangement data, gene inversions/transpositions etc., see (Patané et al., 2018)). These rare, yet informative, footprints play the role of the structural constraints. The user can follow our pre-processing step to get triplet constraints from the given rare footprints, and then use Ahos BUILD algorithm to choose between regularized or hard version of the HC problem. The above illustrates how to use our workflow and why using our algorithms facilitates HC when expert domain knowledge is available.

A.2. Missing proofs and discussion in Section 2

Proof of Proposition 1. For nodes $u, v \in T$, let $P(u)$ denote the parent of u in the tree and LCA(u, v) denote the lowest common ancestor of u, v. For a leaf node $l_i, i \in [k]$, we say that its label is l_i, whereas for an internal node of T, we say that its label is the label of any of its two children. As long as there are any two nodes a, b that are siblings (i.e. $P(a) \equiv P(b)$), we create a constraint $ab|c$ where c is the label of the second child of $P(P(a))$. We delete leaves a, b from the tree and repeat until there are fewer than 3 leaves left. To see why the above procedure will only create at most k constraints, notice that every time a new constraint is created, we delete two nodes of the given tree T. Since T has k leaves and is binary, it can have at most $2k - 1$ nodes in total. It follows that we create at most $\frac{2k-1}{2} < k$ triplet constraints. For the equivalence between the constraints imposed by T and the created triplet constraints, observe that *all* triplet constraints we create are explicitly imposed by the given tree (since we only create constraints for two leaves that are siblings) and that for any three datapoints $a, b, c \in T$ with LCA(a, c)=LCA(b, c), our set of triplet constraints will indeed imply $ab|c$, because LCA(a, b) appears further down the tree than LCA(a, c) and hence a, b become siblings before a, c or b, c. □

Proof of Fact 1 from (Charikar & Chatziafratis, 2017). We will measure the contribution of an edge $e = (u, v) \in E$ to the RHS and to the LHS. Suppose that r denotes the size of the *minimal* cluster in $\texttt{OPT}$ that contains both u and v. Then the contribution of the edge $e = (u, v)$ to the LHS is by definition $r \cdot w_e$. On the other hand, $(u, v) \in \texttt{OPT}(t), \forall t \in \{0, ..., r-1\}$. Hence the contribution to the RHS is also $r \cdot w_e$. □

Proof of Fact 2 from (Charikar & Chatziafratis, 2017). We rewrite $\texttt{OPT}$ using the fact that

$$w(\texttt{OPT}(t)) \geq 0$$

at every level $t \in [n]$:

$$\begin{aligned} 6k \cdot \texttt{OPT} &= 6k \sum_{t=0}^{n} w(\texttt{OPT}(t)) \\ &= 6k(w(\texttt{OPT}(0)) + \cdots + w(\texttt{OPT}(n))) \\ &\geq 6k(w(\texttt{OPT}(0)) + \cdots + w(\texttt{OPT}(\lfloor \tfrac{n}{6k} \rfloor))) \\ &= \sum_{t=0}^{n} w(\texttt{OPT}(\lfloor \tfrac{t}{6k} \rfloor)) \end{aligned}$$

□

Proof of Lemma 2. By using the previous lemma we have:

$$\begin{aligned} \texttt{CRSC} = \sum_A r_A w(B_1, B_2) \leq \\ \leq O(\alpha_n) \sum_A s_A w(\texttt{OPT}(\lfloor \tfrac{r_A}{6k_A} \rfloor) \cap A) \end{aligned}$$

Observe that $w(\texttt{OPT}(t))$ is a decreasing function of t, since as t decreases, more and more edges are getting cut. Hence we can write:

$$\begin{aligned} &\sum_A s_A \cdot w(\texttt{OPT}(\lfloor \tfrac{r_A}{6k} \rfloor) \cap A) \\ &\leq \sum_A \sum_{t=r_A - s_A + 1}^{r_A} w(\texttt{OPT}(\lfloor \tfrac{r_A}{6k_A} \rfloor) \cap A) \end{aligned}$$

To conclude with the proof of the first part all that remains to be shown is that:

$$\sum_A \sum_{t=r_A - s_A + 1}^{r_A} w(\texttt{OPT}(\lfloor \tfrac{t}{6k_A} \rfloor) \cap A) \leq \sum_{t=0}^{n} w(\texttt{OPT}(\lfloor \tfrac{t}{6k} \rfloor))$$

To see why this is true consider the clusters A with a contribution to the LHS. We have that $r_A - s_A + 1 \leq t \leq r_A$, hence $|B_2| < t$ meaning that A is a *minimal* cluster of size $|A| \geq t > |B_2| \geq |B_1|$, i.e. if both A's children are of size less than t, then this cluster A contributes such a term. The set of all such A form a disjoint partition of V because of the definition for minimality (in order for them to overlap in the hierarchical clustering, one of them needs to be ancestor of the other and this cannot happen because of minimality). Since $\texttt{OPT}(\lfloor \frac{t}{6k} \rfloor) \cap A$ for all such A forms a disjoint partition of $\texttt{OPT}(\lfloor \frac{t}{6k} \rfloor)$, the claim follows by summing up over all t.

Note that so far our analysis handles clusters A with size $r_A \geq 6k$. However, for clusters with smaller size $r_A < 6k$ we can get away by using a crude bound for bounding the total cost and still not affecting the approximation guarantee that will be dominated by $O(k\alpha_n)$:

$$\sum_{|A|<6k} r_A w(B_1, B_2) < 6k \cdot \sum_{ij \in E} w_{ij} = 6k \cdot \texttt{OPT}(1) \leq 6k \cdot \texttt{OPT}$$

□

Theorem 6 (The divisive algorithm using balanced cut)**.** *Given a weighted graph $G(V, E, w)$ with k triplet constraints $ab|c$ for $a, b, c \in V$, the constrained recursive balanced cut algorithm (same as $\texttt{CRSC}$, but using balanced cut instead of sparsest cut) outputs a HC respecting all triplet constraints and achieves an $O(k\alpha_n)$-approximation for the HC objective (1).*

Proof. It is not hard to show that one can use access to balanced cut rather than sparsest cut and achieve the same approximation factor by the recursive balanced cut algorithm.

We will follow the same notation as in the sparsest cut analysis and we will use some of the facts and inequalities we previously proved about $\texttt{OPT(t)}$. Again, for a cluster A of size r, the important observation is that the partition $A_1, \ldots, A_l$ (at the end, we will again choose $l = 6k_A$) induced inside the cluster A by $\texttt{OPT}(\frac{r}{l})$ can be separated into two groups, lets say (C_1, C_2) such that $r/3 \leq |C_1|, |C_2| \leq 2r/3$. In other words we can demonstrate a Balanced Cut with ratio $\frac{1}{3} : \frac{2}{3}$ for the cluster A. Since we cut fewer edges when creating C_1, C_2 compared to the partitioning of $\texttt{OPT}(\frac{r}{l})$:

$$w(C_1, C_2) \leq w(\texttt{OPT}(\lfloor \tfrac{r}{l} \rfloor) \cap A)$$

By the fact we used an α_n-approximation to balanced cut we can get the following inequality (similarly to Lemma 1):

$$r \cdot w(C_1, C_2) \leq O(\alpha_n) \cdot s \cdot w(\texttt{OPT}(\lfloor \tfrac{r}{l} \rfloor) \cap A)$$

Finally, we have to sum up over all the clusters A (now in the summation we should write r_A, s_A instead of just r, s, since there is dependence in A) produced by the constrained recursive balanced cut algorithm for Hierarchical Clustering and we get that we can approximate the HC objective function up to $O(k\alpha_n)$. □

Remark 4. Using balanced-cut can be useful for two reasons. First, the runtime of sparsest and balanced cut on a graph with n nodes and m edges are $\tilde{O}(m + n^{1+\epsilon})$. When run recursively however as in our case, taking recursive sparsest cuts might be worse off by a factor of n (in case of unbalanced splits at every step) in the worst case. However, recursive balanced cut is still $\tilde{O}(m + n^{1+\epsilon})$. Second, it is known that an α-approximation for the sparsest cut yields an $O(\alpha)$-approximation for balanced cut, but not the other way. This gives more flexibility to the balanced cut algorithm, and there is a chance it can achieve a better approximation factor (although we don't study it further in this paper).

A.3. Missing proofs in Section 3

Proof sketch of Proposition 2. Here the main obstacle is similar to the one we handled when proving Theorem (1): for a given cluster A created by the $\texttt{R-HSC}$ algorithm, different constraints are, in general, active compared to the $\texttt{OPT}$ decomposition for this cluster A. Note of course, that $\texttt{OPT}$ itself will not respect all constraints, but because we don't know which constraints are active for $\texttt{OPT}$, we still need to use a charging argument to low levels of $\texttt{OPT}$. Observe that here we are allowed to cut an edge ab even if we had the $ab|c$ constraint (incurring the corresponding cost $c_{ab|c}$), however we cannot possibly hope to charge this to the $\texttt{OPT}$ solution, as $\texttt{OPT}$, for all we know, may have respected this constraint. In the analysis, we crucially use a merging procedure between sub-clusters of A having active constraints between them and this allows us to compare the cost of our $\texttt{R-HSC}$ with the cost of $\texttt{OPT}$. □

3-hyperedges to triangles for general weights . Even though the general reduction presented in Section 3 (Figure 3) to transform a 3-hyperedge to a triangle is valid even for general instances of HSC with 3-hyperedges and arbitrary weights, the reduced sparsest cut problem may have negative weights, e.g. when $w_{bc|a} + w_{ac|b} < w_{ab|c}$. To the best of our knowledge, sparsest cut with negative weights has not been studied. Notice however that if the original weights $w_{bc|a}, w_{ac|b}, w_{ab|c}$ satisfy the triangle inequality (or as a special case, if two of them are zero which is usually the case when we have a triplet constraints), then we can actually solve (approximately) the HSC instance, as the sparsest cut instance will only have non-negative weights. □

A.4. Missing proofs in Section 4

Proof of Theorem 3. We start by looking at the objective value of any algorithm as the summation of contributions of different triples i, j and k to the objective, where $(i, j) \in E$ and k is some other point (possibly equal to i or j).

$$\texttt{OBJ} = \sum_{(i,j)\in E} w_{ij}|T_{ij}| = \sum_{(i,j)\in E, k\in V} w_{ij}\mathbf{1}\{k \in \text{leaves}(T_{ij})\}$$
$$= \sum_{(i,j)\in E} \sum_{k\in V} \mathcal{Y}_{i,j,k},$$

where random variable $\mathcal{Y}_{i,j,k}$ denotes the contribution of the edge (i, j) and vertex k to the objective value. The vertex k is a leaf of T_{ij} if and only if right before the time that i and j gets separated k is still in the same cluster as i and j. Therefore,

$$\mathcal{Y}_{i,j,k} = w_{ij}\mathbf{1}\{i \text{ separates from } k \text{ no earlier than } j\ \}$$

We now show that $\mathbf{E}[\mathcal{Y}_{i,j,k}] = \frac{2}{3}w_{ij}$. Given this, the expected objective value of recursive random cutting algorithm will be at least $\frac{2n}{3}\sum_{(i,j)\in E} w_{ij}$. Moreover, the objective value of the optimal hierarchical clustering, i.e. maximizer of the Dasgupta's objective, is no more than $n\sum_{(i,j)\in E} w_{ij}$, and we conclude that recursive random cutting is a $\frac{2}{3}$-approximation. To see why $\mathbf{E}[\mathcal{Y}_{i,j,k}] = \frac{2}{3}w_{ij}$, think of randomized cutting as flipping an independent unbiased coin for each vertex, and then deciding on which side of the cut this vertex belongs to based on the outcome of its coin. Look at the sequence of the coin flips of i, j and k. Our goal is to find the probability of the event that for the first time that i and j sequences are not matched, still i's sequence and k's sequence are matched up to this point, or still j's sequence and k's sequence are matched up to this. The probability of each of these events is equal to $\frac{1}{3}$. To see this for the first event, suppose i's sequence is all heads (H). We then need the pair of coin flips of (j, k) to be a sequence of (H, H)'s ending with a (T, H), and this happens with probability $\sum_{i\geq 1}(\frac{1}{4})^i = \frac{1}{3}$. The probability of the second event is similarly calculated. Now, these events are disjoint. Hence, the probability that i is separated from k no earlier than j is exactly $\frac{2}{3}$, as desired. □

Proof of Theorem 4. We derandomize the recursive random cutting algorithm using the *method of conditional expectations*. At every recursion, we go over the points in the current cluster one by one, and decide whether to put them in the "left" partition or "right" partition for the next recursion. Once we make a decision for a point, we fix that point and go to the next one. Now suppose for a cluster C we have already fixed points $S \subseteq C$, and now we want to make a decision for $i \in C \setminus S$. The reward of assigning to left(right) partition is now defined as the expected value of recursive random cutting restricted to C, when the points in S are fixed (i.e. it is already decided which points in S are going to the left partition and which ones are going to the right partition), i goes to the left(right) partition and $j \in C \setminus (\{i\} \cup S)$ are randomly assigned to either the left or right. Note that these two rewards (or the difference of the two rewards) can be calculated *exactly* in polynomial time by considering all triples consisting of an edge and another vertex, and then calculating the probability that this triple contributes to the objective function (this is similar to the proof of Theorem 3, and we omit the details for brevity here). Because we know the randomized assignment of i gives a $\frac{2}{3}$-approximation (Theorem 3), we conclude that assigning to the better of left or right partition for every vertex will remain to be at least a $\frac{2}{3}$-approximation. For running time, we have at most n clusters to investigate. Moreover, a careful counting argument shows that the total number of operations required to calculate the differences of the rewards of assigning to left and right partitions for all vertices is at most $n(n + 2m)$. Hence, the running time is bounded by $O(n^2(n + m))$. □

Proof sketch of Theorem 5. Before starting to prove the theorem, we prove the following simple lemma.

Lemma 3. *There is no edge between any two classes in the same layer $\mathcal{I}_l$.*

Proof of Lemma 3. If such an edge exists, then there is a path of length $l + 1$ from $\mathcal{C}$ to a class in $\mathcal{I}_l$, a contradiction. □

Now, similar to the proof of Theorem 3, we consider every triple $\{x, y, z\}$, where $(x, y) \in E$ and z is another point , but this time we only consider z's that are not involved in any triplet constraint (there are at least $n - k$ such points). We claim with probability at least $\frac{2}{3\cdot\text{DMC}(\{c_1,\ldots,c_k\})}$ the supernode containing z is still in the same cluster as supernodes containing x and y right before x and y gets separated. By summing over all such triples, we show that the algorithm gets a gain of at least $\frac{2(n-k)}{3\cdot\text{DMC}(\{c_1,\ldots,c_k\})}\sum_{(x,y)\in E} w_{xy}$, which proves the α-approximation as the optimal clustering has a reward bounded by $n\sum_{(x,y)\in E} w_{xy}$.

To prove the claim, if (x, y) is not the base of any triplet constraint then a similar argument as in the proof of Theorem 3 shows the desired probability is exactly $\frac{2}{3}$ (with a slight adaptation, i.e. by looking at the coin sequences of supernodes containing x and y, which are going to be disjoint in this case at all iterations, and the coin sequence of z). Now suppose (x, y) is the base of any constraint c and suppose c belongs to a class $\mathcal{C}$. Consider the layered dependency subgraph of $\mathcal{C}$ as in Definition 2 and let the layers to be $\mathcal{I}_0, \ldots, \mathcal{I}_L$. In order for z to be in the same cluster as x and y when they get separated, a chain of $L + 1$ independent events needs to happen. These events are defined inductively; for the first event, consider the coin sequence of z, coin sequence of (the supernode containing all the bases of) constraints in $\cup_{l=0}^{L}\mathcal{I}_l$ and coin sequences of all the keys of constraints in $\mathcal{I}_L$ (there are $\sum_{\mathcal{C}'\in\mathcal{I}_L}|\mathcal{C}'|$ of them). Without loss of generality, suppose the coin sequence of (the supernode containing) $\cup_{l=0}^{L}\mathcal{I}_l$ is all heads. Now the event happens only if at the time z flips its first tales all keys of $\mathcal{I}_L$ have already flipped at least one tales. Conditioned on this event happening, all the constraints in $\mathcal{I}_L$ will be resolved and z remains in the same cluster as x and y. Now, remove $\mathcal{I}_L$ from the dependency subgraph and repeat the same process to define the events $2,\ldots,L$ in a similar fashion. For the l^{th} event to happen, we need to look at $1+\sum_{\mathcal{C}'\in\mathcal{I}_L}|\mathcal{C}'|$ number of i.i.d. symmetric geometric random variable, and calculate the probability that first of them is no smaller than the rest. This event happens with a probability at least $\left(1+\sum_{\mathcal{C}'\in\mathcal{I}_L}|\mathcal{C}'|\right)^{-1}$. Moreover the events are independent, as there is no edge between any two classes in $\mathcal{I}_l$ for $l\in[L]$, and different classes have different keys. After these L events, the final event that needs to happen is when all the constraints are unlocked, and z needs to remain in the same cluster as x and y at the time they get separated. This event happens with probability $\frac{2}{3}$. Multiplying all of these probabilities due to independence implies the desired approximation factor. □