CS 598CSC: Topics in Graph Algorithms	Lecture date: 10/01/2024
Instructor: Chandra Chekuri	Scribe: CC

1 Cut-Based Hierarchical Decomposition of a Graph

Räcke [Rac02] proved the existence of a cut-based hierarchical decomposition of undirected graphs as a tool to prove the existence of good oblivious routings. As we saw in last lecture, he obtained a different construction for worst-case optimal oblivious routings. The cut-based hierarchical decomposition has found several applications outside the original motivation for oblivious routing, and is also a fundamental structural result in graph theoretic terms. The construction and proof are technical. This is our attempt to teach it and provide some additional commentary along the way, which we hope is of pedagogical value for those interested in understading the details of a construction from [BKR03] — we provide pointers to other constructions/proofs later. First, we set up notation to state the result. We are more interested in the cut-approximation and not so much in the oblivious routing aspect and hence will state the result in that fashion.

Given a graph $G = (V, E)$ a hierarchical decomposition $\mathcal{H}$ is *laminary* family of subsets of the vertex set V. A family of subsets is laminary if no two sets A, B in the family cross - that is, A and B are disjoint or one is contained in the othre. Any laminar family over V, augmented with the entire set V if needed, defines a way to decompose V, and hence implicity the graph, in a recursive fashion. Each such decomposition has an associated rooted tree T. The leaves of the tree are labeled with the vertex set V and each internal node v_t corresponds to a subset S_{v_t} of the vertices of G which are the leaves in the subtree T_{v_t}. We associate the graph $H_{v_t} = G[S_{v_t}]$ with each v_t. Hierarchical decompositions of graphs are used in several settings and the meaning and application of them depend on the application and context. Here we will be interested in decompositions that preserve cuts of the graph, in particular for routing demands in G. We call a hierarchical decomposition T a *cut-tree* by naturally associating capacities to the edges of T as follows. For each edge $(v_t, v_{t'})$, where $v_{t'}$ is the parent of v_t, we assign a capacity equal to $c(\delta_G(S_{v_t}))$.

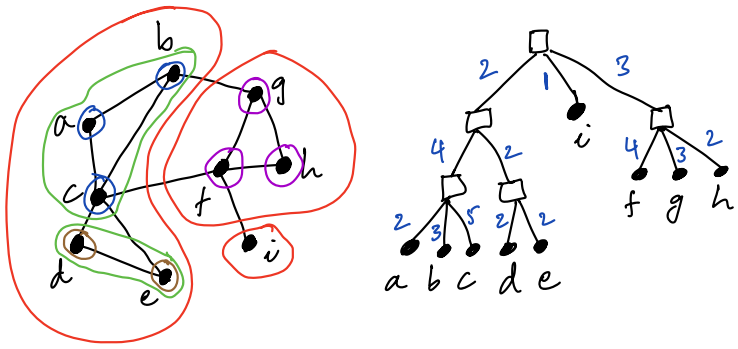

Figure 1: A graph and a hierarchical decomposition and its representation as a cut-tree.

A cut tree has exactly n edges where $n = |V|$ so it is keepting track only a very small number

of cut values. Surprisingly, every graph admits a cut-tree which can approximate all sparse cuts in the graph. To formalize the statement we start with the following simple observation.

Claim 1. *Let $G = (V, E)$ be a graph with edge capacities $c : E \to \mathbb{R}_+$ and let T be any cut-tree for G. Suppose $D \in R_+^{V \times V}$ is a non-negative demand matrix over the vertices of G which we view also a demand matrix over the leaves of T. If D is routable in G then D is routable in T.*

Proof. Recall that a demand matrix is routable in a capacitated tree iff it satisfies the cut condition. The only sparse cuts of interest in a tree are the single edge cuts. If D is routable in G then for any cut $(S, V - S)$ we have $D(S, V - S) \leq c(S, V - S)$ in G. Since T is a cut-tree and D is only between leaves of T, each edge $(v_t, v_{t'})$ in E_T corresponds to the cut $(S_{v_t}, V - S_{v_t})$ in G and the capacity of the edge in T is equal to $c(S, V - S)$. Thus D satisfies the cut-condition in T and hence is routable in T. ■

Now we are ready to state the result of Räcke.

Theorem 2 ([Rac02])**.** *For every graph $G = (V, E)$ on n nodes there is a cut-tree T such that any demand matrix $D \in R_+^{V \times V}$ that is routable in T is also routable in G with congestion $O(\log^3 n)$.*

In other words there is a very compact data structure, namely, the cut-tree, that captures all the relevant cuts for routing (in particular the sparse cuts) in G approximately to within a poly-logarithmic factor. The first proof of [Rac02] was based on finding optimum sparse cuts and hence did not lead to an efficient algorithm. Soon after two papers, [BKR03] and [HHR03] obtained efficient algorithms with [HHR03] obtained an improved congestion bound of $O(\log^2 n \log \log n)$ which is still the best known. We will give a proof outline of the construction from [BKR03] which obtained a weaker congestion bound of $O(\log^4 n)$. If one wants to approximate only cuts but not the routing aspect then [RS14] gives an improved bound of $O(\log^{1.5} n \log \log n)$. In this lecture we only deal with the former aspect where we want routability. We say that a cut-tree is ρ-approximate if any demand matrix D routable in T can be routed in G with congestion ρ.

1.1 Notation

The proof is technical so we need some notation. Throughout we will be working with induced subgraphs of the original graph that arise as the algorithm constructs the hierarchical decomposition. Given $S \subseteq V$ we typically use $G[S]$ to denote the induced subgraph. We also refer to it as a cluster and also associate it with a node of the cut-tree T. We will be interested in how well can S interface with the rest of the graph in terms of the edges between S and $V - S$. Since we will be interested in G and various induced subgraphs, we will use the notation $E(A, B)$ to denote the set of all edges with one end point in A and the other in B. Note that A and B need not be disjoint. Thus, to represent the $\delta_G(S)$ we will often use $E(S, V - S)$. We will use $c(A, B)$ as a short cut for the total capacity of edges in $E(A, B)$, that is $c(A, B) = \sum_{e \in E(A,B)} c(e)$. We will refer to $c(\delta_G(S))$ sometimes as $\text{out}(S)$ for compactness.

1.2 Intuition, connection to expanders and well-linked decomposition

In most of this lecture we are interested in conductance but we use the word expanders. Suppose $G = (V, E)$ has good conductance if $\phi(G) = \Omega(1)$. That is, for every $S \subset V$, $c(\delta(S))| \geq \phi \cdot \text{vol(S)}$. Then the following holds because of the flow-cut gap and is an exercise.

Exercise 1. *Suppose G has conductance ϕ. Let D be any demand matrix such that the following holds for each $u \in V$: $b(u) = \sum_{v \in V} D(u,v) \leq c(\delta(u))$. Then G satisfies the cut condition for ϕD and hence D is routable in G with congestion $O(\frac{1}{\phi}\sigma(n))$ where $\sigma(n)$ is the flow-cut gap for product multicommodity flow in graphs of at most n nodes. In particular $\sigma(n) = O(\log n)$ in general graphs and is $O(1)$ in planar graphs.*

Thus, if $\phi = \Omega(1)$ then the set of routable demand matrices can be approximately characterized by stating that they should respect the trivial cuts at the vertices. Thus, it is not hard to see the following.

Exercise 2. *Argue that if G is a graph with conductance ϕ then the a star with V as the leaves is a cut-tree with approximation $O(\frac{1}{\phi}\log n)$.*

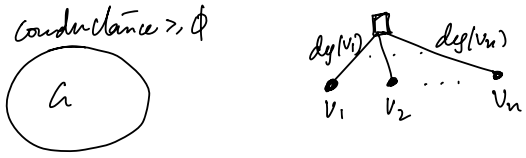

Figure 2: A star is a good cut-tree for a graph with good conductance.

What should one do if G does not have good conductance? A natural approach is to decompose the graph into well-connected pieces, in other words sub-graphs with high conductance, along sparse cuts. The question is how. It is also not difficult to see that decomposing the graph into one or a few levels is not sufficient. For example if one takes a planar graph such as $\sqrt{n} \times \sqrt{n}$ grid then no subgraph has good conductance unless it is essentially of constant size. Thus, the only reasonable thing is to decompose the graph recursively into several levels. For example the grid can be decomposed in a natural fashion into say four equal sized grids and then recursively to obtain a $O(\log n)$ depth cut-tree which one can prove has poly-log approximation.

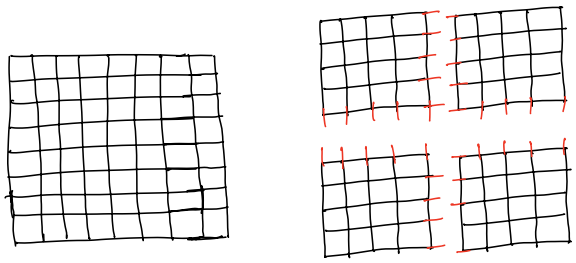

Figure 3: Decomposing a grid recursively into sub-grids yields a good cut-tree. Understanding why is useful for general graphs.

Although grids are easy to decompose due to their nice symmetric properties, it is still non-obvious to formally understand why it yields a good cut-tree. A key is to see that the boundary of the grid is well-linked. How should one do this for a general graph whose structure is not apparent

to us? The key concepts that one needs are being able to understand how a given cluster $H = G[S]$ interfaces with the rest of the graph through its boundary edges (those edges that cross S) and also how a given cluster is decomposed into sub-clusters. To formalize things we will set up some definitions.

Boundary and partition well-linkedness: We saw the notations of well-linkedness before but we will define them again here both for cuts and flows and refine them for our need. Well-linkedness allows us to use the notion of expansion/conductance for subsets of vertices and with arbitrary weights so that we can use it in a refined way.

Definition 1. *Let $G = (V, E)$ be a graph and let $\pi : V \to \mathbb{R}_+$ be bounds on vertices. For scalar $\alpha > 0$, G is (π, α)-cut-well-linked if for all $S \subset V$ $c(\delta(S)) \geq \min\{\pi(S), \pi(V - S)\}$.*

The *support* of π is the set of vertices such that $\pi(v) > 0$.

Definition 2. *Let $G = (V, E)$ be a graph and let $\pi : V \to \mathbb{R}_+$ be weights on vertices. For scalar $\alpha > 0$, G is (π, α)-flow-well-linked if for any demand matrix D that satisfies the condition $\sum_v D(u, v) \leq \pi(u)$ for all $u \in V$, is routable in G with congestion $1/\alpha$.*

Remark 3. *We defined flow-well-linkedness in a stronger form in the above since this is useful for the application in this topic. It is defined in a slightly different form in [?] as follows: G is (π, α)-flow-well-linked if the demand matrix D_π defined as $D(u, v) = \pi(u)\pi(v)/\pi(V)$ is routable in G with congestion $1/\alpha$. The advantage of this definition is that we can check whether G is flow-well-linked efficiently via a multicommodity flow computation. See exercise below to see that the two are closely related and do not matter much for the application in this lecture.*

Exercise 3. *Let $\pi : V \to \mathbb{R}_+$ be non-negative bounds on vertices of $G = (V, E)$. Suppose G can route the product multicommodity flow induced by π. Then prove that G can route any demand matrix D with congestion 2 if $\sum_v D(u, v) \leq \pi(u)$ for all $u \in V$.*

From flow-cut gap we obtain the following.

Claim 4. *If G is (π, α)-cut-well-linked then G is $(\pi, \frac{\alpha}{\sigma(n)})$-flow-well-linked.*

Remark 5. *Note that if G has conductance ϕ then it is (π, ϕ)-cut-well-linked where $\pi(u) = c(\delta(u))$ is the capacitated degree of u in G. And as we saw it also implies that G is $(\pi, \phi/\sigma)$-flow-well-linked. Allowing π to be arbitrary allows us to generalize the notation of conductance and routability to subsets of vertices and to control the amount of flow incident to a vertex.*

In the context of cut-trees we are interested in two specific types of well-linkedness and we give them names so that they evoke the right connection.

Definition 3. *Let $G = (V, E)$ be a graph and $H = G[S]$ be an induced subgraph/cluster. S is α-boundary-cut/flow-well-linked if H is (π, α)-cut/flow-well-linked where $\pi(u) = c(u, V - S)$ is the capacity of the edges on the boundary of S that are incident to u. See Fig 4.*

Note that in the above definition we are requiring the graph $H = G[S]$ to be well-linked as a separate graph. We see that if a cluster is boundary-well-linked then it acts as good router as far as its external interface is concerned. This is an important property that is essentially required

of any cluster in a cut-tree since the capacity of the edge from a cluster to its parent is equal to $c(S, V - S)$, the boundary capacity.

Another important property that is needed for a cluster is with respect to its children in the cut-tree, or in other words, how it is partitioned into sub-clusters. This motivates the following definition.

Definition 4. *Let $G = (V, E)$ be a graph and $H = G[S]$ be an induced subgraph/cluster. Let $\mathcal{D}$ be a partition of S into sub-clusters $H_1, H_2, \ldots, H_r$ where $H_j = G[S_j]$. S is α-boundary-and-partition-cut/flow-well-linked H is (π, α)-cut/flow-well-linked where $\pi(u)$ for each $u \in S$ is the capacity of edges going outside its sub-cluster. See Fig 4.*

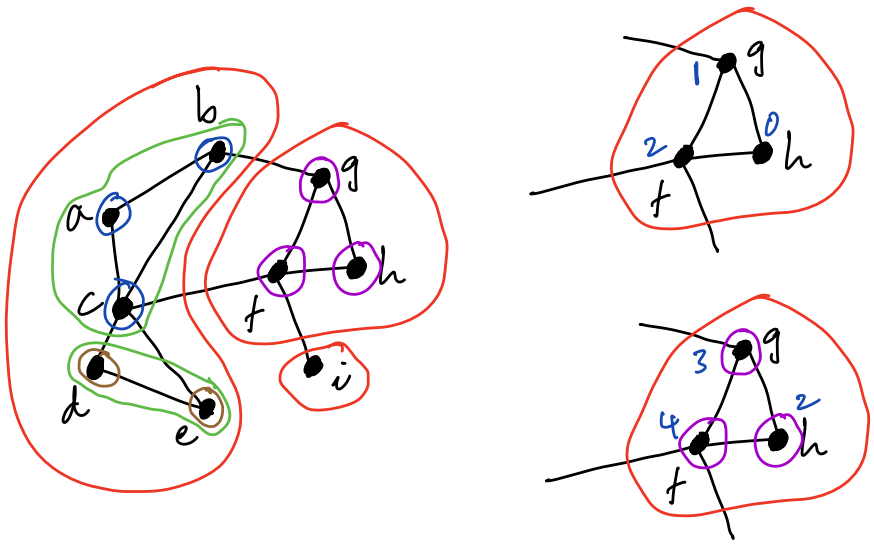

Figure 4: Weights for boundary well-linkedness and partition-well-linkedness of a cluster.

To simplify notation we say that S is α-partition-well-linked with resepect to $\mathcal{D}$ and omit $\mathcal{D}$ when it is implicit in the context.

To see the intuition and need for the preceding defintion we consider a cluster S in the cut-tree and its children $S_1, S_2, \ldots, S_r$. The edge connecting S_j to S in the tree has capacity equal to $c(\delta_G(S_j))$. In a sense the cluster S acts as a single node in the tree connecting the children together. Thus S should be able to route any set of demands that go between the children as long as the total demand leaving each S_j is at most $c(\delta_G(S_j))$.

It is sometimes useful to view the boundary well-linkedness and partition-well-linkedness by sticking dummy vertices as leaves and saying that the dummy vertices are well-linked (with weight 1). For example, in Fig 4, imagine adding a dummy vertex to edge sticking out of the cluster.

2 Sufficient condition for a cut-tree to be a good one

Now that we have set up the definitions the first part of the proof is to understand what conditions should the tree T satisfy such that it has good properties. Interestingly only two properties are

required: low depth/height and partition-well-linkedness of each cluster in the decomposition. This is captured by the following lemma.

Lemma 6. *Let T be a cut tree for G of height h. Suppose each cluster S of the tree (which corresponds to an internal node) is α-flow-partition-well-linked then T is $O(\frac{h}{\alpha})$-approximate.*

The power of the preceding characterization is that it has essentially captures what we need from the hierarchical decomposition. We will see a construction that guarantees that $h = O(\log n)$ and $\alpha = \Omega(1/\log^3 n)$, and this gives us a cut-tree that is $O(\log^4 n)$-approximate. We now prove the lemma. Although one can actually construct an oblivious routing as well, we will not do so to keep the proof simpler.

Recall that each cluster $H = G[S]$ in the tree is α-partition-and-external-well-linked. For a cluster S we let π_S be the weights on vertices of S which correspond to the external degree of the vertices. Similarly we let π'_S denote the weights corresponding to the partition $\mathcal{D}$ of S induced by its children. Note that $\pi'_S(u) \geq \pi(u)$ for all $u \in S$. Also note that $\pi_S(S) = c(S, V - S)$ is the capacity of the edges leaving S. When S is clear from the context we simply us π and π'.

Suppose D is a demand matrix that is routable in T. We need to show that it is routable in G with congestion $O(h/\alpha)$. The proof works by constructing a series of multicommodity flows that can be stitched together to route all the demands. We first explain how the height h comes into the picture. Since T is a hierarchical decomposition, each edge $e = uv \in E$ can be in upto h clusters along a path from the root until u and v get separated for the first time. We will assume that each cluster S in the tree has its own private copy of each edge $e \in E[S]$ when we compute multicommodity flows in the cluster. We will bound the congestion of routing per cluster and then use the fact that each edge is in at most h clusters to derive the final congestion.

We route the demands bottom up. Let (a, b) be a vertex-pair with demand $D(a, b)$. Let $H = G[S]$ be the cluster which corresponds to the least common ancestor of a and b in T. We call the cluster H the meetup cluster for the pair (a, b). Let $H_1, H_2, \ldots, H_r$ be the children of H in T. Since H is the lca, a and b are both in S, and are in different children of H. Without loss of generality, let $a \in H_1 = G[S_1]$ and $b \in H_2 = G[S_2]$. Consider any cluster $H' = G[S']$ along the path from a to H not including H itself. Such a cluster is called a transition cluster for pair (a, b). a will distribute its $D(a, b)$ flow such that each node $u \in S'$ receives a fraction $\pi_{S'}(u)/\pi_{S'}(S')$ of the flow. Why? Recall that $\pi_{S'}(S') = c(S', V - S')$ is the boundary capacity of the cluster S' and $\pi_{S'}(u)$ is the amount of that capacity incident to u. And all of a's flow has to leave the cluster S' to eventually reach b so we spread the flow in proportion to the boundary capacity. Note that the actual flow from a for the pair (a, b) that reaches u is $D(a, b)\frac{\pi_{S'}(u)}{\pi_{S'}(S')}$. The terminal b also sends its flow up the tree towards the lca H in a similar fashion. For pair (a, b) this stops as H_1 for a and at H_2 for b. It then comes the responsibility of the meetup cluster $H = G[S]$ to ensure that these flows match up in S. Thus, the flow for pair (a, b) is fully satisfied in the clusters of the sub-tree at their least common ancestor which corresponds how it is routed in the tree T.

To complete the description, since the flow is sent bottom up, we need to describe how a cluster $H = G[S]$ with children $H_1, H_2, \ldots, H_r$ maintains the invariant. It has two parts

- For all demand pairs (a, b) such that H is the meetup cluster, H has to ensure that the flow for the pairs which have reached the children's boundary are exchanged in the graph $G[S]$.
- For all demand pairs (a, b) for which H is a transition cluster (meaning that their lca is above H in T), their flow has reached exactly one of the children of H inductively. H needs to

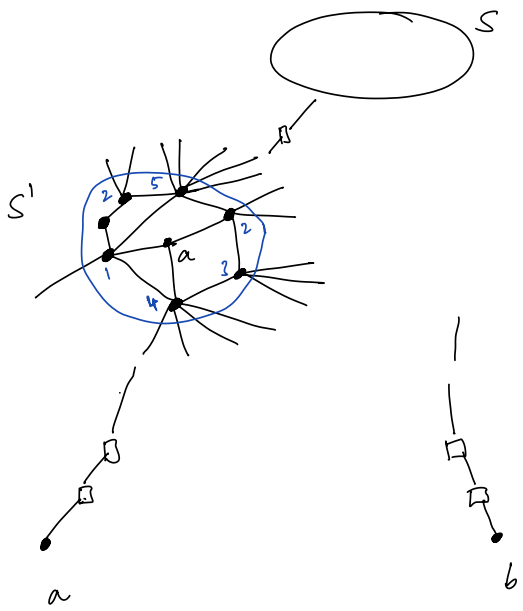

Figure 5: A transition cluster for demand pair (a, b). The cluster S' is on the path from a in T to the lca with b. a distributes $D(a, b)$ to each node $u \in S'$ in proportion to their boundary capacity.

redistribute this flow using edges only in $G[S]$ such that for each such demand pair (a, b) the flow is distributed over nodes in S in proportion to the $\pi_S(u)$ values.

We show that each of the above two steps can be accomplished in $H = G[S]$ with congestion $O(1/\alpha)$ by using the α-flow-partition-well-linkedness property. It turns out to be reasonably simple because we have set up the definitions appropriately.

Consider the first part. Fix a pair (a, b) for which H is the meetup cluster. As we discussed before say $a \in H_1 = G[S_1]$ and $b \in H_2 = G[S_2]$ where H_1 and H_2 are children of H. a has already sent its flow to vertices in S_1 where $u \in S_1$ has received a flow of $D(a, b)\frac{\pi_{S_1}(u)}{\pi_{S_1}(S_1)}$. Similarly b has sent its flow to S_2 in proportion to π_{S_2}. We need to exchange this flow — we can set up a demand matrix to do this exchange. We won't describe it explicitly but it is not hard to see that there is one which will ensure that the flow is matched up. Similarly, for every pair for which H is the meetup cluster we can exchange the flows that have reached the children clusters by setting up their own demand matrix. Routing the union of these demand matrices in H would suffice to exchange the flow. Can we do it? And what is the congestion incurred? We claim that all these demand matrices can be routed in H with congestion $1/\alpha$ for the following reason. Recall that D is routable in T. Consider any child cluster $H_I = G[S_i]$ of H. Since D is routable in T, the total demand that needs to meetup in H and which originates in H_i is at most the capacity of the edge (H, H_i) in T which has capacity $c(S_i, V - S_i) = \pi_{S_i}(S_i)$, in other words, the boundary capacity of S_i. This means that for each $u \in S_i$, the total flow that has reached u for all the demands that need to be matched up in H is at most $\frac{\pi_{S_i}(u)}{\pi_{S_i}(S_i)} \cdot c(S_i, V - S_i) = \pi_{S_i}(u)$. Thus, for exchanging the meetup flow we have set up many demand matrices, but the total demand from these matrices that originate at $u \in S_i$

is at most $\pi_{S_i}(u)$. This is true for all the vertices in S. But recall that any demand matrix that satisfies this degree condition, by definition of the α-flow-partition-well-linkedness, can be routed in $H = G[S]$ with congestion $1/\alpha$. And we are done! In fact, this is the reason we required this condition on the clusters.

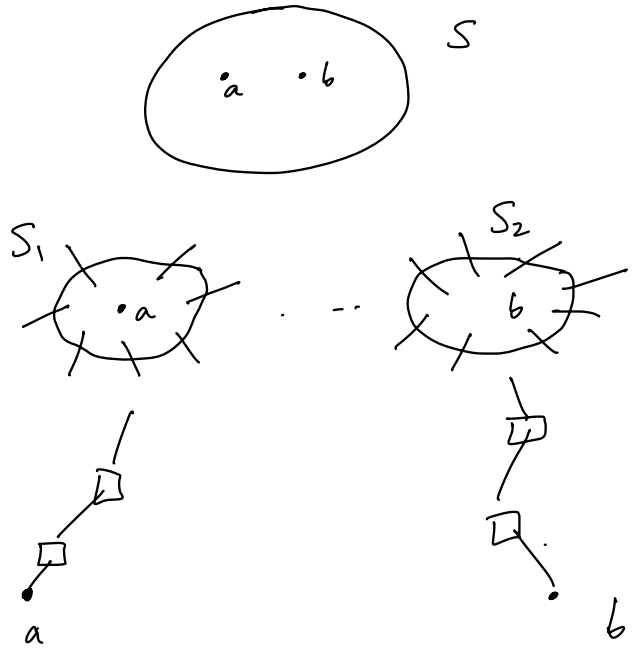

Figure 6: The meetup cluster $H = G[S]$ for pair (a, b) exchanges the flow sent by a to S_1 and sent by b to S_2. It sets up a demand matrix to exchange this flow.

The second part is also similar. Consider any demand pair (a, b) for which H is a transition cluster. This means that one of a, b is in H and the other is not. Say, $a \in H$ and a has already distributed the flow the pair (a, b) to the boundary of the child H_i of H where $a \in H_i$. a has distributed its flow to each node $u \in S_i$ in proportion to $\pi_{S_i}(u)/\pi_{S_i}(S_i)$. To maintain the invariant, in cluster H we need to distribute a's flow to nodes in S such that each $v \in S$ gets flow in proportion to $\pi_S(v)/\pi_S(S)$. We can set up a demand matrix to exchange this flow. For any node $u \in S_i$ the total demand originating at u is at most $\pi_{S_i}(u)$ since the total demand crossing S_i is at most $c(S_i, V - S_i)$. Also, for each node $v \in S$ the total demand that it receives is at most $\pi_S(v) \leq \pi'_S(v)$. Thus, the union of the demand matrices respect the total bounds imposed by π' at each node of S, and hence by the α-flow-partition-well-linked guarantee, can be routed in S with congestion $1/\alpha$.

For the base case of the induction we see that the flow originates at the singleton leaves which is its own boundary and hence there is nothing to do there. As we argued earlier, the total congestion on an edge edge is the union of the congestions of all the clusters it participates in, and an edge participates in at most h clusters. This finishes the proof of the lemma.

Exercise 4. *Consider a grid graph with unit capacities and the natural recursive decomposition where we decompose each grid into four (almost) equal sized sub-grids. Prove that this yields a*

cut-tree that is $O(\log n)$-approximate.

3 A top-down algorithm for constructing a cut-tree

We now describe a top-down divide and conquer algorithm that starts with V and creates a cut-tree satisfying the conditions of Lemma 6 with $h = O(\log n)$ and $\alpha = \Omega(1/\log^3 n)$. The algorithm does not backtrack. That is, once it partitions a cluster S into its children, it simply recurses on the children. For this to work, the algorithm requires that each cluster satisfy an additional property before it can be successfully partitioned into sub-clusters that satisfy the partition-well-linked property. Following [BKR03] we call this the *precondition*. The entire vertex set V will trivially satisfy this precondition and hence the algorithm can get started. To enable recursion, the algorithm will ensure that when it partitions S into sub-clusters, each of the resulting sub-clusters also satisfies this precondition. Why do we need this precondition? Because it turns out that not every cluster can be successfully partitioned to satisfy the partition-well-linked property. The algorithm will obtain a tree of height $O(\log n)$ by ensuring that each cluster S is decomposed into sub-clusters such that no sub-cluster S_i has more than $2|S|/3$ vertices.

We will use some parameters for formal proofs and to help see the ideas. Let σ denote the upper bound on the flow-cut gap in an n-vertex graph for product multicommodity flow instances. We know $\sigma = O(\log n)$ but we leave it as a parameter. Note that one can get an $O\sigma$ approximation for sparsest cut when needed. We let $\lambda = 64 \cdot \sigma \cdot \log n$ as a parameter. We see that $\lambda = \Theta(\log^2 n)$ for general graphs. We will aim to prove a cut-tree decomposition with $\alpha = \frac{1}{24\sigma\lambda} = \Omega(1/\log^3 n)$. Thus, α is a log factor smaller than λ.

We will assume that capacities are integer valued. The algorithm is polynomial in the sum of the capacities so technically we can only apply it for capacities that are poly-bounded but we will ignore this issue since we are more interested in the proof of the existence for now.

Precondition: Let $H = G[R]$ for $R \subseteq V$ be a cluster. Let $\pi : R \to \mathbb{R}_+$ be boundary capacity weights, that is $\pi(u)$ is the capacity of the edges leaving R that are incident to u. Recall that we want each cluster to satisfy the property that R is boundary well-linked. This will be essentially our precondition but in a precise form. R satisfies the precondition if for all sets U such that $|U| \leq \frac{3}{4}|R|$, we have

$$c(U, R \setminus U) \geq \frac{1}{\lambda}\pi(U).$$

Note that this resembles a cut-well-linkedness condition but it is not quite the same becase the numerator is about π while the denominator is about the cardinality of U. However we will see later that this condition corresponds to sparsity of a related demand matrix. The motivation for this precondition comes from the eventual divide-and-conquer algorithm for constructing a cut-tree which requires each part to be a constant factor smaller than its parent cluster.

It is not feasible to efficiently check whether a given cluster satisfies the precondition so we will use approximation algorithms via flow-cut gap to indirectly guarantee that the precondition is satisfied.

3.1 Ensuring precondition via well-linked style decomposition

During the algorithm we need to ensure that the sub-clusters that are created satisfy the precondition for the recursion. As we remarked above we cannot directly guarantee it. For this we will use a decomposition procedure called AssurePreCondition that will decompose *any* given cluster R such that the resulting pieces after the decomposition satisfy the precondition. We state the lemma and postpone the proof for later.

Lemma 7. *Let $R \subseteq V$ be any cluster. There is an efficient algorithm that decomposes R into sub-clusters $R_1, R_2, \ldots, R_p$ for some $p \geq 1$ such that the following two properties hold:*

- *each R_i satisfies the precondition, and*
- $\sum_{i=1}^{p} c(\delta(R_i)) \leq 2c(\delta(R))$.

The algorithm to achieve the above is very similar to the expander decomposition procedure that we saw earlier. Recall that in expander decomposition we wished to decompose a given graph into pieces such that each piece has good conductance while cutting as few edges as possible. In expander decomposition we were able to show that we can ensure that each piece has conductance $\Omega(\frac{1}{\sigma \log n})$ (via an $O(\sigma)$-approximation algorithm for sparsest-cut based on flow-cut gap) while cutting only a constant fraction of edges of the graph. In the preceding lemma, the increase in the number cut edges is upper bounded by $c(\delta(R)/2$ (since the sum counts each such newly cut edge twice). But we are only interested in each piece being boundary-well-linked so this is similar to well-linked decomposition with initial weight equal to $\pi_R(u)$.

AssurePreCondition(G, R)

1. Computer sparsest cut in $H = G[R]$ with demands $d(u, v) = \frac{\pi(u)}{|R|}$ for each *ordered* pair u, v using flow-cut gap based approximation algorithm. Let (A, B) be the output of the algorithm and ψ the sparsity of the cut.
2. If $\psi > \frac{4\sigma}{\lambda}$ then return R as the single piece
3. Else recurse on A and B and return the union of their decompositions.

We postpone the proof of Lemma 7 to Section 3.3.

3.2 The Partition Algorithm

Now we are ready to describe the main algorithm Partition that takes any connected cluster $H = G[S]$ with $|S| \geq 2$ that satisfies the precondition and returns a partion of H into sub-clusters $S_1, S_2, \ldots, S_p$ for some $p > 1$ such that

- $|S_i| \leq 2|S|/3$ for each sub-cluster S_i
- each S_i satisfies the precondition
- S satisfies the α-flow-partition-well-linkedness property with respect to the decomposition.

Suppose we have such an algorithm. Then the top-down recursive algorithm is straightforward. Notice that V trivially satisfies the precondition because it has no outgoing edges. Thus the algorithm can get started. In each step it ensures that the sizes of the sub-clusters goes down by a constant factor. They satisfy the precondition to enable recursion. And the cluster itself satisfies the desired partition-well-linkedness property with respect to the decomposition.

An important remark here the following. Given a cluster S and decomposition into sub-clusters we can check efficiently whether it satisfies the partition-well-linkedness property — see Remark 3.

How should PARTITION work? Suppose $R = V$ and G has good conductance. Then, as we argued, the right decomposition is to simply take all the singleton vertices and it satisfies the desired properties. Thus, the algorithm starts with the trivial decomposition of R into singletons and the nice aspect of this starting decomposition is that the first two properties are automatically satisfied. We can then check if the third property is also satisfied. If it is then we got lucky and we are done! Of course this is too optimistic. What the algorithm actually does is to maintain a current partition that satisfies the first two properties. Let this partition be $\mathcal{P} = \{H_1, H_2, \ldots, H_p\}$. It then checks if S satisfies the desired third property for this partition. If it does we are done. Otherwise it finds a sparse cut (A, B) of S since the partition-well-linkedness property failed. It then uses the partition (A, B) to compute another partition $\mathcal{P}'$ such once again $\mathcal{P}'$ satisfies the first two properties. How does it make progress? It does so by ensuring that the total number of inter-cluster edges in $\mathcal{P}'$ is strictly less than the inter-cluster edges in $\mathcal{P}$. This is the crux of the proof. Thus, the algorithm can repeat this process and it is guaranteed to terminate with a partition that satisfies all three properties. The algorithm is only guaranteed to reduce the number of inter cluster edges by one so this is the reason for the dependence of the running time on $\sum_e c(e)$.

Now we need to describe how the algorithm computes the new partition $\mathcal{P}'$ from $\mathcal{P}$. We develop some intuition before giving a formal description. Recall that (A, B) is a sparse cut for the partition-well-linkedness condition. It means that the number of edges crossing A is too small in comparision to $\pi'(A)$. What is $\pi'(A)$? It is the total number of intercluster edges in the current decomposition and the total number of external edges. Since R satisfied the precondition it is already boundary-well-linked with parameter $1/\lambda$. Thus the reason we have a sparse cut is because there are too many intercluster edges inside A due the current partition $\mathcal{P}$. Suppose we got lucky and A is the union of some sub-collection of clusters from $\mathcal{P}$. Then it makes sense to merge all the clusters in A and make A the new cluster. This reduces the number of intercluster edges. The new cluster A may not satisfy the precondition but we can use ASSUREPRECONDITION on A to partition it into clusters that satisfy the precondition (we introduce new intercluster edges in this process but not too many). The main issue is that A may not be a clean union of current clusters. The algorithm employs a simple heuristic to find a union of existing clusters. It takes the union of all clusters that have at least 3/4 of their vertices inside A. In [BKR03] this procedure is called $round(A)$. The main technical claim is to show that this works. It may not be clear at this stage that that U^* is non-empty. The formal algorithm is described below.

Partition(G, R)

1. Assumes that R satisfies the precondition.
2. Initialize $\mathcal{P} = \{\{v\} \mid v \in R\}$
3. Repeat

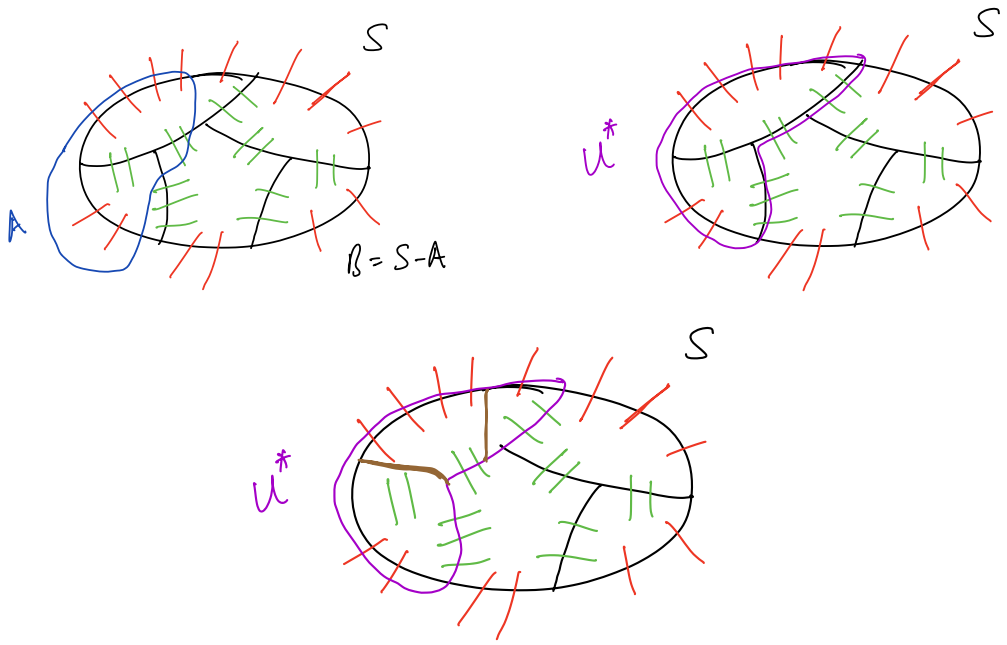

Figure 7: PARTITION finds a sparse cut (A, B) if the current decomposition does not satisfy the desired flow-well-linkedness property. It coverts A to a set U^* a union of existing clusters. It then uses ASSUREPRECONDITION on U^* to find new clusters that satisfy the precondition.

(a) Check if R satisfies α-flow-partition-well-linkedness with respect to $\mathcal{P}$ via flow computation.

(b) If it is satisfied then Break.

(c) Otherwise let (A, B) be a sparse cut such that sparsity is at most $\sigma\alpha$ and with $|A| \leq |B|$.

(d) $U^* = \emptyset$

(e) for each $R_i \in \mathcal{P}$ do

- if $(|R_i \cap A| > 3/4|R_i|)$ then
  $U^* \leftarrow U^* \cup R_i$ and $\mathcal{P} \leftarrow \mathcal{P} - \{R_i\}$

(f) $\mathcal{Q} \leftarrow AssurePreCondition(G, U^*)$.

(g) $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{Q} - U^*$

4. Return $\mathcal{P}$

We see that that algorithm starts with the partition consisting of the singletons and either terminates or updates the partition. We claim that it maintains the first two properties for the partition at the start of each iteration. And clearly when it terminates we see that it satisfies the third property as well since it explicitly checks for it. The initial partition satisfies the first two properties so we show that this is maintained. If the algorithm does not terminate, it computes U^* by taking the union of some of the existing clusters and then uses ASSUREPRECONDITION to cluster it.

We first show that $|U^*| \leq 2|R|/3$. The following is an easy exercise since $|A| \leq |R|/2$ and we only include in U^* clusters that have large intersection with A (and they satisfy the first property previously).

Claim 8. *Suppose $\mathcal{P}$ fails the partition-well-linkedness property. Let U^* be the union of clusters that have large intersection with A. Then $|U^*| \leq 2|R|/3$.*

We run AssurePrecondition on U^* so the resulting sub-clusters satisfy the precondition and their size can only decrease. The clusters which are not in U^* are not touched and they already satisfied the properties. Thus, we see that the algorithm maintains the invariant for $\mathcal{P}$ that each cluster is not too big and that it satisfies the precondition.

The main technical lemma that ensures termination is the following which relates the sparsity of the cut (A, B) with the sparsity of the cut U^*. In the lemma below we use $\pi' : R \to \mathbb{R}_+$ to be the weights associated with the partition $\mathcal{P}$ when we compute the sparse cut (A, B).

Lemma 9.

$$\frac{c(\delta_G(U^*))}{\pi'(U^*)} \leq 4\lambda \cdot \frac{c(A, B)}{\pi'(A)}.$$

In other words, converting A to a union of the existing clusters costs a factor 4λ in the sparsity. We will prove the preceding lemma in Section 3.4.

Since (A, B) is a sparse cut with sparsity at most $\sigma\alpha$,

$$\frac{c(A, B)}{\pi'(A)} \leq \sigma\alpha \leq \frac{1}{24\lambda}.$$

Combined with the preceding lemma we have

$$c(\delta_G(U^*)) \leq \pi'(U^*)/6.$$

Note that π' is with respect to the current partition $\mathcal{P}$.

We will use this prove termination of the algorithm.

Lemma 10. *The algorithm* Partition *terminates in time polynomial in $|R|$ and the maximum integer edge capacity of the edges.*

Proof. Let $E(\mathcal{P}) = \{uv \in E \mid u, v \in R \text{ and } u, v \text{ in diff clusters}\}$ be the set of inter cluster edges induced by the partition $\mathcal{P}$. We observe that $\pi'(R) = 2c(E(\mathcal{P})) + c(\delta_G(R))$ since we count each inter-cluster edge twice and the edges leaving R to outside once. Let $\mathcal{P}'$ be the partition at the end of the iteration. It suffies to prove that the total capacity of $E(\mathcal{P}')$ is strictly less than that of $E(\mathcal{P})$.

How does $\mathcal{P}'$ differ from $\mathcal{P}$? We remove all clusters of $\mathcal{P}$ contained inside U^* and add clusters obtained by running AssurePreCondition on U^*; recall Q is the output of this algorithm. From Lemma 7, we have $\sum_{S' \in Q} c(\delta_G(S')) \leq 2c(\delta_G(U^*))$. Let $\pi'' : R \to \mathbb{R}_+$ be the new weights induced by the partition $\mathcal{P}'$. We can see that $\pi''(R) = \pi'(R) - \pi'(U^*) + \sum_{S' \in Q} c(\delta_G(S'))$ and since $\pi'(U^*) \geq 6c(\delta_G(U^*))$, we have $\pi"(R) < \pi'(R)$ since $c(\delta_G(U^*)) > 0$. Since $\pi"(R) = 2c(E(\mathcal{P}')) + c(\delta_G(R)$ we see that $E(\mathcal{P}') < E(\mathcal{P})$. Thus the number of inter-cluster edges reduces in each iteration. ■

3.3 Proof outline for Lemma 7

Let R be a cluster with weights $\pi : R \to \mathbb{R}_+$ corresponding to the boundary capacity of vertices. The algorithm AssurePrecondition is similar to expander/well-linked decomposition but not exactly the same due to the nature of the condition. The algorithm sets up a demand matrix D which we explain since it is not a product multicommodity flow. For each vertex $u \in R$ it creates a demand $D(u, v)$ for each v where $D(u, v) = \frac{\pi(u)}{|R|}$. In other words, u wants to distribute $\pi(u)$ uniformly among all vertices. Thus the final demand matrix is the union of these demand matrices, one for each u. Our goal is relate the sparsest cut for this demand matrix to the precondition so that we can guarantee that R will satisfy the precondition if there is no sparse cut with respect to D.

Let ϕ be the sparsity according to D. Then, via the flow-cut gap we can find a cut (A, B) such that its sparsity (according to D) is at most $\sigma \cdot \phi$.

Claim 11. *Suppose R does not satisfy the precondition. Then $\phi \leq \frac{4}{\lambda}$.*

Proof. If R does not satisfy the precondition then there is a partition $(U, R - U)$ of R such that

$$c(U, R - U) \leq \frac{1}{\lambda}\pi(U)$$

Consider the demand according to D crossing this partition. It is $\pi(U)\frac{|R-U|}{|R|} + \pi(R-U)\frac{U}{R}$ but this is greater than $\pi(U)/4$ since $|R-U| \geq \frac{|R|}{4}$. Thus the sparsity of this cut is at most $4c(U, R-U)/\pi(U) \leq 4/\lambda$ by our assumption. Hence $\phi \leq 4/\lambda$ since it is the min sparsity among all cuts for D. ■

The preceding claim ensures that the algorithm AssurePreCondition guarantees that each cluster in the final partition satisfies the precondition (because of the termination condition and the recursion). The main issue is why it does not cut too many edges in creating the partition. The analysis is similar to the one we did for expander decomposition but it is a bit more involved. We will try and write a recursion to understand the total number of edges cut.

Suppose R satisfies the min-sparsity condition we return it and don't cut any edges so that is the easy case. Suppose the algorithm finds a partition of R into (A, B) such that the sparsity of the cut, $\psi < \frac{4\sigma}{\lambda} < \frac{1}{16 \log n}$. The algorithm recurses on A and B. The total demand crossing (A, B) is $\pi(A)\frac{|B|}{|R|} + \pi(B)\frac{|A|}{|R|} \leq \pi(A) + \pi(B) = \pi(R)$. Thus,

$$c(A, B) \leq \frac{1}{16 \log n}\pi(R).$$

Thus, the total edges cut is at most $\pi(R)$. Unlike previous expander decomposition recursion analysis, these new cut-edges create two new problems whose total size in terms of outgoing edges is slightly larger than the original number of outgoing edges. Nevertheless the $\log n$ factor is sufficiently large that this increase can also be absorbed. One has to do the analysis a bit carefully.

3.4 Proof outline for Lemma 9

TODO

References

[BKR03] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 24–33, 2003.

[HHR03] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 34–43, 2003.

[Rac02] Harald Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52. IEEE, 2002.

[RS14] Harald Räcke and Chintan Shah. Improved guarantees for tree cut sparsifiers. In *Algorithms-ESA 2014: 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings 21*, pages 774–785. Springer, 2014.