Fully Dynamic Min Cut

그래프의 *Minimum Cut* (최소 컷) 은 그래프가 연결되지 않도록 지워야 하는 간선의 최소 개수를 뜻한다. 간선에 양의 정수 가중치가 붙으면 이를 *Weighted Minimum Cut* 이라고 부르고, 이 때는 그래프가 연결되지 않도록 지워야 하는 간선 가중치 합의 최소를 구해야 한다.

최소 컷을 다른 말로 *Edge Connectivity* 라고 부르기도 한다. 연결된 (Connected) 그래프는 최소 컷이 1 이상인 그래프와 동치이고, 이중 연결 (Biconnected) 된 그래프는 최소 컷이 2 이상인 그래프와 동치이고, 삼중 연결 (Triconnected) 된 그래프는 최소 컷이 3 이상인 그래프와 동치이기 때문이다. 그래프의 연결성 그 자체이니 중요성에 대해서는 언급할 가치가 없다.

이 글에서는 Mikkel Thorup의 STOC'01 논문인 *Fully-dynamic min-cut* 을 설명한다. 제목이 모든 것을 말해 주듯이, 이 논문은 Fully-dynamic graph (간선 추가, 제거) 의 최소 컷이 $polylog(n)$ 이하이면, $\tilde{O}(\sqrt{n})$ 시간에 그 최소 컷을 관리할 수 있다고 주장한다. 최소 컷이 상수 이하인지 아닌지도 많은 관심을 받는 주제이다. Dynamic connectivity에 관련한 가장 중요한 논문인 HdLT01의 key result 중 하나가, 최소 컷이 2 이하인지 여부를 $O(\log^5 n)$ 에 관리하는 것이었다. 논문의 최소 컷이 상수 이하인 경우를 넘어서 $polylog(n)$ 이하인 경우를 다루고, 그 복잡도가 $\tilde{O}(\sqrt{n})$ 이니 탐구해 볼 가치가 충분하다고 할 수 있다. 논문의 내용에 대해서 자세히 알아 보자.

Related researches

논문의 내용이 2001년이기 때문에, 논문이 나온 당시와 현재의 SOTA가 차이가 있다. 이 글에서는 간략하게 이들을 짚고 넘어간다.

- Edge Connectivity가 1 이상인지를 결정하는 알고리즘을 *그래프 탐색* (DFS, BFS) 이라고 부른다.
- Edge Connectivity가 2 이상인지를 결정하는 알고리즘을 *절선* (Bridge) 이라고 부른다.
- Edge Connectivity가 3 이상인지를 결정하는 선형 시간 알고리즘은 1991년 발견되었다.
- Edge Connectivity가 4 이상인지를 결정하는 선형 시간 알고리즘은 2021년 발견되었다.
- Weighted Edge Connectivity를 $O(m \log^2 n)$ 시간에 계산하는 랜덤 알고리즘은 2019년 발견되었다.
- Unweighted Edge Connectivity를 $O(m \log^2 n \log\log^2 n)$ 시간에 계산하는 결정론적 알고리즘은 2017년 발견되었다.
- Weighted Edge Connectivity를 $O(m^{1+o(1)})$ 시간에 계산하는 결정론적 알고리즘은 2021년 발견되었다.
- Dynamic Graph에서 모든 쿼리가 Offline일 때, Edge Connectivity가 고정된 상수 c 이상인지를 판별하는 $O(polylogn)$ 알고리즘은 2020년 발견되었다.
- Dynamic Graph에서 Edge Connectivity가 1 이상인지를 결정하는 amortized $O(\log^2 n)$ 알고리즘은 2001년 발견되었다.
- Dynamic Graph에서 Edge Connectivity가 2 이상인지를 결정하는 amortized $O(\log^2 n \log\log^2 n)$ 알고리즘은 2018년 발견되었다.
- Dynamic Graph에서 Edge Connectivity가 $O(polylog(n))$인 어떤 수 이상인지를 결정하는 worst-case $O(\sqrt{n})$ 알고리즘을 이 글에서 소개한다.

Tree Packing Technique

Nash-Williams Theorem에 의해서, 최소 컷이 $2k$ 이상인 그래프에서는 k 개의 Edge-disjoint spanning tree를 찾을 수 있다. 고로 이 중 최소 하나의 스패닝 트리는 최소 컷과의 교집합이 2 이하일 것이다. Minimum Cut을 구하는 효율적인 알고리즘들은 대부분 이러한 류의 정리를 사용한다.

이 중 가장 자주 사용되는 것은 Karger의 Greedy Tree Packing으로, 최소 컷과의 교집합이 2 이하인 트리를 높은 확률로 빠른 시간에 샘플링할 수 있다.

이 논문에서도 Karger의 Greedy Tree Packing을 사용할 것이지만, 최소 컷과의 교집합이 2 이하인 트리를 찾는다 하더라도 이를 Dynamic하게 관리하는 것이 쉽지 않다. 고로 Tree packing을 변형하여서 최소 컷과의 교집합이 1인 트리를 찾는 것을 목표로 한다.

이러한 트리를 알고 있을 때 전체 문제는 자료구조들을 창의적으로 조합하는 식으로 해결되며, 해당 부분이 이 글의 핵심이다. Tree packing에 관한 부분은 technical한 계산이 많고, 이미 알려진 내용들이라서 짧게 소개하고 넘어간다.

Definition

- 그래프 G 의 *Tree packing* $T = \{T_1, T_2, \dots, T_k\}$ 는 스패닝 트리의 집합이다.
- 간선의 *Load* $L^T(e)$ 는 $e \in T_i$ 를 만족하는 i 의 개수이다. Load는 2 이상일 수 있다. 즉, 한 간선이 두 개 이상의 스패닝 트리에 속할 수 있다.
- 간선의 *Relative load* $l^T(e)$ 는 $\frac{L^T(e)}{T}$ 이다.
- Tree packing의 가중치 $packVal(T)$ 는 $1/max_{e \in E(G)} l^T(e)$ 이다.
    - 그래프 G 에서 가능한 $packVal(T)$ 의 최댓값을 τ 라고 한다.
- $V(G)$ 의 파티션 P 에 대해서, 이의 *cut value* 를 $cutVal(P) = \frac{E(G \setminus P)}{P-1}$ 이라 정의한다.
- 그래프의 최소 컷의 크기를 λ 라고 한다.
- 그래프 G 와 스패닝 트리 T 가 주어졌을 때, $G \cup T$ 의 컷 중 T 의 간선을 k 개만 사용하는 컷을 k-Respecting Cut이라고 부른다.

Nash-Williams Theorem (1961) 에 의해서 다음 두 사실이 성립한다.

- **Theorem 1**. $\tau = \min_P cutVal(P)$.
- **Corollary 2**. $\lambda/2 \leq \tau \leq \lambda$.

잘 알려진 내용이니 증명은 생략한다.

어떠한 tree packing $T = \{T_1, T_2, \dots, T_k\}$ 가 **Greedy** 하다는 것은 각 T_i 가 $\{T_1, T_2, \dots, T_{i-1}\}$ 의 load를 가중치로 했을 때 최소 스패닝 트리를 이룬다는 것을 뜻한다. 크기 k 의 Greedy tree packing은 $\tilde{O}(km)$ 시간에 쉽게 계산할 수 있다.

Greedy Tree Packing

들어가기 앞서 Karger의 Greedy Tree Packing을 사용한 Minimum Cut 알고리즘을 복습하자.

Lemma 3. $T \geq 3\lambda \ln m/\epsilon^2$ 를 만족하는 Greedy Tree Packing은 $packVal(T) \geq (1-\epsilon)\tau$ 를 만족한다.

Tree packing의 성질은 우리 글의 Scope가 아니기 때문에 증명은 생략한다.

이를 Corollary 2와 조합하면, $\lambda/2(1-\epsilon) \leq (1-\epsilon)\tau \leq packVal(T) \leq \tau \leq \lambda$ 라는 식을 얻는다. ϵ 이 충분히 크다면, T 의 임의의 스패닝 트리와 최소 컷의 교집합의 기댓값이 2에 수렴하게 된다. 여기서 계산을 조금 더 하면, T 에 있는 스패닝 트리 중 constant fraction의 트리가 최소 컷과 2의 교집합을 갖는다. 고로 여기서 적정한 개수의 트리를 뽑은 후 2-Respecting Min-Cut을 구하는 것이 Karger의 알고리즘이다.

이 논문에서도 비슷한 Approach를 사용할 것이다. 하지만 사용하기 앞서, 우리는 Dynamic한 그래프에서 Greedy Tree Packing을 관리할 수 있어야 한다.

Theorem 4. Dynamic Graph에서, 크기 T 의 Greedy Tree Packing은 간선 추가/제거마다 $O(T^2 \log^4 n)$ 시간에 관리할 수 있다.

Proof. 일반성을 잃지 않고 e 를 지운 경우만 고려할 수 있다. 갱신 전 T_i 의 Load를 $l_i(e)$, 갱신 후 T_i 인 T_i' 의 Load를 $l_i'(e)$ 라고 하자. 핵심 관찰은, 간선 추가/제거시 각 T_i 에 대해서 $l_i(e) \neq l_i'(e)$인 간선이 최대 i 개라는 것이다. 이유는:

- 모든 i 에 대해서, $\sum_e l_i(e) = \sum_e l_i'(e) = i(n-1)$ 이다.
- 지워지지 않은 모든 간선 $f \neq e$ 에 대해 $l_i'(f) \geq l_i(f)$ 이다. 증명은 수학적 귀납법을 사용하면 된다. $i = 0$ 일 경우 자명하다. $l_{i-1}'(f) \geq l_{i-1}(f)$ 가 성립한다고 하자.
    - 만약 $f \in T_i - T_i'$ 이라면, 즉 f 가 스패닝 트리에서 제거된다면, $l_{i-1}'(f) > l_{i-1}(f)$ 이다. 그렇지 않다면, 귀납 가정에 의해 모든 간선의 가중치가 그대로거나 증가하는 상황에서, 간선이 T_i' 에서 제거될 이유가 없기 때문이다 (Tie case가 애매한데 Lexicographical order로 MST를 고른다는 식으로 가정하자). 고로 $l_i'(f) \geq l_{i-1}'(f) \geq l_{i-1}(f) + 1 \geq l_i(f)$ 이다.
    - 만약 $f \notin T_i - T_i'$ 이라면, 즉 f 의 상태가 유지되거나 스패닝 트리에 추가된다면, $l_i'(f) - l_{i-1}'(f) \geq l_i(f) - l_{i-1}(f)$ 이다. 고로 $l_i'(f) \geq l_i(f)$이다.
    - 고로 수학적 귀납법에 의해 증명이 완료된다.
- 지워진 간선에서 감소한 Load의 양은 최대 i 이니 남은 간선에서 증가한 양의 합도 최대 i 다. 고로 값이 증가한 간선의 수는 최대 i 개이다.

각 간선 추가/제거 마다 T^2 개의 간선이 바뀌고, 이 과정에서 Dynamic MST를 관리해야 한다. 이는 각 간선 갱신마다 $O(\log^4 n)$ 시간에 관리할 수 있음이 잘 알려져 있다 (HdLT01). Theorem 4가 증명된다. ■

마지막으로, 이 부분의 Main Theorem을 설명한다.

Theorem 5. $T = \omega(\lambda^7 \log^3 m)$ 를 만족하는 Greedy Tree Packing과 1-Respecting하는 Minimum Cut이 존재한다.

Theorem 5의 증명은 우리 글의 scope가 아니며, 생략한다. 이 Theorem의 의외는 2-Respecting Cut이 아니라 1-Respecting Cut만 찾아도 된다는 것이다. 교집합이 1인 트리를 찾는 것은 2보다 어렵다. 훨씬 더 큰 Tree packing을 요구하며, constant fraction이 아니라 existance만 보장한다. 하지만, 이 논문에서는 Polylogarithmic factor를 자유롭게 사용할 수 있으며, Dynamic 2-Respecting Cut이 아니라 1-Respecting Cut을 구함으로써 훨씬 더 많은 시간상 이점을 얻어가기 때문에 의미가 있다.

Main Result: Dynamic 1-Respecting Cut for Trees

이제 이 논문의 Main Theorem을 설명한다.

Theorem 6. 정점 집합을 공유하는 Dynamic Graph G 와 Dynamic Tree T 가 있을 때, $G \cup T$ 의 최소 컷 중 크기가 $polylog(N)$ 이하이며 T 를 1-Respecting하는 *모든 컷*을 간선 추가/제거마다 $\tilde{O}(\sqrt{N})$ 에 관리할 수 있다.

그동안 증명한 Theorem을 모아서 전체 문제를 해결하자.

Theorem 7. Dynamic Graph가 주어질 때, 어떠한 $\lambda \leq polylog(n)$ 에 대해서, Edge Connectivity가 λ 이하일 경우 Edge Connectivity 값을 간선 추가/제거마다 $\tilde{O}(\sqrt{n})$ 에 관리하는 알고리즘이 존재한다.

Proof. λ 가 주어질 때, $\omega(\lambda^7 \log^3 m)$ 크기의 Greedy Tree Packing T 을 관리한다. Theorem 4에 의해, 이는 간선 추가/삭제마다 $O(polylog(n))$ 시간에 관리할 수 있다. Theorem 6에 의해, $O(polylog(n)) \times \tilde{O}(\sqrt{N})$ 시간에 각 T_i 에 대한 1-respecting cut을 관리할 수 있다. Theorem 5에 의해, 관리한 컷 중 Minimum Cut이 존재한다. ■

Proof of the Main Theorem

Main Theorem을 증명하는 데 있어서 Top Tree의 이해가 필수적이므로, 이에 대한 지식을 가정한다. Top Tree가 제공하는 구조는 Dynamic Tree에서 Path와 유사한 성질을 얻는 데 매우 큰 도움이 된다.

아래 Theorem 6을 구체화한 버전을 제시한다.

Theorem 6.1. 정점 집합을 공유하는 Dynamic Graph G 와 Dynamic Tree T 가 있을 때, 모든 $e \in T$ 에 대해서 $cover(e) = min(\sqrt{M}, (\text{e의 서브트리 안과 밖을 잇는 } G \text{ 의 간선 수}))$ 라는 수량을 Top Tree에 관리할 수 있다. 이 때 드는 시간은 간선 추가/제거마다 $\tilde{O}(\sqrt{M})$ 이다.

여기서 알아두면 좋은 것은, $cover(e)$ 의 크기는 $\sqrt{M}$ 까지 관리되지만 우리는 $polylog(N)$ 이하의 컷만 관심이 있다는 점이다. 굳이 $polylog(N)$ 이하라는 조건을 준다고 풀이가 쉬워지지 않기 때문에 이와 같이 Lemma가 형성되었으며, 전체적 흐름과는 상관이 없다.

또한 Theorem 6.1은 Theorem 6보다 약한 결과이다. Dense Graph에서, Theorem 6.1은 Main Theorem을 증명하지 못한다. 이 문제점은 최후에 해결하고, 일단 이후 단락에서는 Theorem 6.1의 증명에 초점을 맞춘다.

Lemma 6.2 (Static Two-level Top Tree). $1 \leq B \leq N$ 을 만족하는 정수 B 에 대해서, 다음 조건을 만족하는 Cluster 부분집합 (이를 *Leaf Cluster* 라고 부른다.) 이 존재하게끔 $O(N)$ 시간에 Top Tree를 구성할 수 있다.

- 루트에서 리프로 가는 모든 경로는 Cluster 부분집합에 속하는 Cluster를 정확히 한 번 만난다.
- 컴포넌트에 속하는 간선이 B 개 이상일 경우, 속하는 Cluster들의 크기는 $[B, 3B]$ 구간에 속한다.
- 그렇지 않을 경우 Cluster 부분집합은 루트 Cluster이다.

본인의 전달 능력 부족으로 Lemma를 어렵게 설명하였으나, 이는 일반적인 Sqrt Decomposition의 요령과 동일하다. 한 마디로, 트리를 크기 $[B, 3B]$ 의 연결 컴포넌트 (Bucket) 로 클러스터링했다고 보면 된다. 다만, 이 클러스터링된 연결 컴포넌트 각각이 Top Tree로 구성되어 있으며, 연결 컴포넌트들도 Top Tree로 묶여서 관리되는 것일 뿐이다.

Proof 6.2. 단순한 Bottom-up 방식으로 트리를 만들 수 있다. $DFS(v)$ 라는 함수는 v 를 루트로 한 서브트리에서 연결 컴포넌트들을 구성한다. 이 때 루트 컴포넌트의 크기는 B 미만이고, 그 외의 컴포넌트의 크기는 $[B, 2B]$ 구간에 속한다.

정점 v 에서는, 각 서브트리를 임의의 순서대로 보면서 루트 컴포넌트를 합쳐준다. 현재 보고 있는 루트 컴포넌트의 크기가 B 이상이 될 경우 해당 컴포넌트를 *완성* 시키고, 빈 루트 컴포넌트를 새로 만든다. 이렇게 작업한 후, v 의 조상으로 가는 간선을 작업한 루트 컴포넌트에 추가한다.

최종적으로 루트를 포함한 컴포넌트를 제외한 모든 컴포넌트의 크기가 $[B, 2B]$ 이다. 루트 컴포넌트의 크기가 B 미만이라면, 자식 중 아무 컴포넌트에 합쳐지면 된다.

이후 각 컴포넌트를 Top Tree로 구성하면, 컴포넌트는 클러스터가 된다. 이 클러스터들도 역시, Top Tree로 구성될 수 있다. ■

Lemma 6.3 (Dynamic Two-level Top Tree). Lemma 6.1의 트리를 $O(B)$ 시간에 Dynamic하게 관리할 수 있다.

Proof 6.3. 간선이 없는 포레스트에 대해서는 자명하게 위와 같은 트리를 구성할 수 있다. 이제 간선이 추가/삭제될 때를 살펴보자.

- 간선이 추가되었을 때는, 간선이 잇는 두 정점이 속하는 임의의 클러스터를 Expose한 후, 간선을 포함한 크기 1의 클러스터를 합쳐준다. 합쳐진 클러스터의 크기가 $3B$ 를 초과했다면, Lemma 6.1을 사용하여 Static하게 reconstruct해준다.
- 간선이 제거되었을 때는, 제거된 간선이 속하는 클러스터에 Soft expose를 통한 Cut을 수행한다. 이제 나뉘어진 컴포넌트에서, 만약에 현재 속하는 클러스터의 크기가 B 미만일 경우, 인접한 클러스터 하나에 붙어서 합쳐준다. 이 클러스터의 크기가 $3B$ 를 초과했다면, Lemma 6.1을 사용하여 Static하게 reconstruct 해준다. ■

Theorem 6.3. 정점 집합을 공유하는 Dynamic Graph G 와 Dynamic Tree T 가 있을 때, 모든 $e \in T$ 에 대해서 $cover(e) = min(\sqrt{M}, (\text{e의 서브트리 안과 밖을 잇는 } G \text{ 의 간선 수}))$ 라는 수량을 Top Tree에 관리할 수 있다. 이 때 드는 시간은 간선 추가/제거마다 $\tilde{O}(\sqrt{M})$ 이다.

T 를 1-Respecting하는 최소 컷은 $min_{e \in T} cover(e) + 1$ 의 크기를 가지기 때문에, 이 값을 Top Tree를 통해서 관리할 수 있다면 전체 문제가 해결된다.

여기서 알아두면 좋은 것은, $cover(e)$ 의 크기는 $\sqrt{M}$ 까지 관리되지만 우리는 $polylog(N)$ 이하의 컷만 관심이 있다는 점이다. 굳이 $polylog(N)$ 이하라는 조건을 준다고 풀이가 쉬워지지 않기 때문에 이와 같이 Lemma가 형성되었으며, 전체적 흐름과는 상관이 없다.

이제 몇가지 Lemma를 제시한다.

Lemma 6.4. 어떠한 Leaf Cluster C 와 C 의 Path $\pi(C)$ 에 대해서, $C \setminus \pi(C)$ 의 임의의 간선을 Cover하는 G 의 간선은, C 의 내부 정점 ($\pi(C)$ 의 끝점이 아닌 C의 정점) 에 한 끝점을 가지고 있다. 또한 이 간선들이 $C \setminus \pi(C)$를 커버하는 방법은 $T \setminus C$ 의 구조와 독립적이다.

이제 우리는 각 클러스터 C 에 대해서 Cluster path 상에 없는 모든 에지들에 대한 covering을 관리한다. C 에 incident한 모든 간선에 대해서, 클러스터 내부의 cover를 Top Tree를 통해서 관리한다. 이렇게 되면 경로에 대한 값이 관리가 잘 되지 않는데, 만약 어떠한 간선의 두 양 끝점이 모두 C 에 속한다면, $\pi(C)$ 상에서 이 간선에 대한 cover는 관리해 준다. 즉, 우리가 관리하고 있지 않는 값은 C 상에 0개 혹은 1개의 끝점을 가지는 간선이 $\pi(C)$ 에서 cover하는 양이다. 여기까지의 값을 각 리프에 대해서 관리하는 것은 $\sqrt{M}$ 시간에 쉽게 할 수 있으며, 초기화 시간 역시 동일하다.

리프가 아닌 클러스터의 경우에는, 두 자식 클러스터에서 값을 합쳐줘야 한다. 이 경우, 대부분의 케이스의 경우에는 단순히 자식 클러스터에서 들어오는 값을 합쳐주면 된다. 이 중, 자식 클러스터 (A, B) 중 하나가 path cluster인 경우가 문제가 된다. $\pi(A)$ 의 값이 바뀌기 때문이다.

이를 위해, 각 클러스터 쌍에 대해서 $E(A, B)$ 를 A, B 사이를 잇는 non-tree edges들의 집합이라고 하자. 이 집합 자체를 관리하는 것 역시, $\sqrt{M} \log N$ 시간에 쉽게 가능하다. 간선의 추가 / 제거에 영향을 받는 노드가 최대 $\log N$ 개이니 각 노드에 대해서 $\sqrt{M}$ 시간을 사용하면 되기 때문이다. 단순한 풀이는, 모든 Top Tree의 자식 노드 쌍 (A, B)에 대해서, $E(A, B)$ 집합 상에 있는 모든 간선에 대해서, $\pi(A), \pi(B)$ 구간을 이 간선이 커버하는 양을 관리하면 된다.

이를 최적화하기 위해서는, $E(A, B)$ 집합에서 중요한 $\sqrt{M}$ 개의 간선만을 관리할 것이다. $\sqrt{M}$ 이하의 컷에만 관심이 있다고 하면, $E(A, B)$ 집합에서 $\pi(A)$ 과의 교집합을 최대화하거나, $\pi(B)$ 와의 교집합을 최대화하는 최대 $\sqrt{M}$ 개의 간선만을 관리하면 된다는 것을 관찰하자. 이 정보는 $E(A, B)$ 를 재귀적으로 구성하는 과정에서 Order가 맞도록 관리할 수 있다. 이렇게 되면 매 업데이트마다 $\sqrt{M} \log N$ 개의 Increment/Decrement 연산이 필요하니, 전체 명제가 증명된다. ■

또한, 실제 Theorem을 증명하기 위해서는 $\sqrt{M}$ 이 아니라 $\sqrt{N}$ 이라는 Time bound가 필요하다. 이는 다음과 같은 잘 알려진 테크닉을 사용하면 된다.

Theorem 6.5 (Eppstein's Sparsification). 정점 집합을 공유하는 Dynamic Graph G 와 Dynamic Tree T 가 있을 때, $G \cup T$ 의 최소 컷 중 크기가 $polylog(N)$ 이하이며 T 를 1-Respecting하는 *모든 컷*을 간선 추가/제거마다 $\tilde{O}(\sqrt{N})$ 에 관리할 수 있다.

Proof. G 의 간선 집합을 임의로 $O(N)$ 크기의 조각으로 쪼갠 후, 각 조각을 Binary tree의 형태로 합쳐준다. 리프 노드에 대해서는 Theorem 6.4을 사용하여 $polylogN$ 이하의 최소 컷을 관리하자. 조각의 크기가 작기 때문에 $\tilde{O}(\sqrt{N})$ 에 관리할 수 있다.

Theorem 6.5에 의해 Theorem 6이 바로 증명된다. ■