Densest Subgraph: Supermodularity, Iterative Peeling, Flow

이 글에서는 SODA 2022 논문인 Densest Subgraph: Supermodularity, Iterative Peeling, Flow 를 요약한다. 위 논문은 Densest subgraph problem과, 이 문제의 supermodular set function 일반화 버전의 풀이를 다루는 논문이다. 본인은 최근 Densest subgraph problem에 대해서 최근에 다양한 접근 방법들을 정리하고 있는데, 이 글 역시 그러한 노력의 일환이라고 보면 좋다.

Densest subgraph problem의 중요성에 대해서는 이 글을 참고하면 좋다.

Densest subgraph problem에 대해서는 현재 크게 세 가지 접근 방법이 있다.

- Flow 에 기반한 Exact solver.
- LP 에 기반한 Exact solver.
- Charikar의 2-approximation.

이 논문은 여러 Chapter로 구성되어 있는데, 각 Chapter는 사실 DSG를 다룬다는 것 외에는 크게 유사성이 없다. 하지만 내용 전체를 종합해 보면, 결국 이 논문은 DSG 문제에 대한 모든 접근 법에 대해서 통일된 접근법을 제시하거나, State-of-the-art 알고리즘을 제시한다는 의미가 있다.

- 첫 번째 챕터에서는 Flow에 기반한 Exact solver를 제안한다. 이 Exact solver는 $(1+\epsilon)$-approximation을 계산하는 알고리즘이며, 시간 복잡도상 SOTA이다. 알고리즘이 상당히 단순하고, 특히 기존 Exact solver들이 대다수 복잡한 LP solver에 기반해 있는 것과 다르게 전통적인 combinatorial algorithm을 사용하여 문제를 해결한다는 데 의미가 있다.
- 두 번째 챕터에서는 Charikar의 2-approximation을 supermodular set function에 대해서 일반화하며, 이에 따라 등장하는 여러 일반화된 성질과, supermodular set function에서 발견되는 파라미터들을 정리한다.
- 세 번째 챕터에서는, Charikar의 2-approximation과 LP에 기반한 (MWU-based) exact solver의 연결관계를 멋지게 정리한다. Charikar의 2-approximation은 이론적 bound가 가장 낮지만, 실용적으로는 굉장히 자주 사용되고 가치가 큰 알고리즘이다. LP에 기반한 exact solver들은 이론적으로 $(1+\epsilon)$ approximation을 얻는 가장 메이저한 방법이었으나, 실용적으로는 Charikar의 알고리즘의 단순함을 이길 수 없었다. 이 논문에서는 매트로이드 이론의 Lovasz extension이라는 장치를 사용하여, 사실 이 두 solver의 메커니즘이 유사함을 보이고, Charikar의 간단함과 MWU-based의 이론적 soundness의 장점을 취합한 새로운 알고리즘 *Super-Greedy++* 을 제안한다.

Chapter 1. Flow strikes back for DSG

첫 번째 알고리즘은 플로우 접근에 기반한 간단한 $(1-\epsilon)$ approximation이다. DSG의 $(1-\epsilon)$ approximation을 구하는 기존 접근들은 대부분 플로우보다는 LP의 관점에서 접근하는 경우가 많았다 (Width-Independent MWU 글 참고). 이 논문에서는 이와 대비되는 플로우 기반의 접근을 사용한다. 위 접근의 시간 복잡도는 $1/\epsilon$ 에 최소 제곱이 붙는데, 이 접근의 경우 $1/\epsilon$ 에 near-linear dependency를 가진다는 것 역시 주목할만한 차이점이다.

간선에 $w : E \to [1, \infty)$ 라는 가중치가 붙은 hypergraph $G = (V, E)$가 주어진다고 하자. 정점 부분집합 $U \subseteq V$ 라고 할 때, U 에 의해 induce된 간선 부분집합의 가중치 합은 $f(U) = \sum_{e \subseteq U} w(e)$ 일 것이다. 고로, U 의 density는 $\frac{f(U)}{|U|}$ 라고 할 수 있다. (정점 가중치는 없다고 가정한다. 있어도 해결할 수 있으나 이 글에서는 다루지 않는다.)

다음과 같은 알고리즘이 존재한다.

Theorem 1.1. $W = \sum_{e \in E} w(e)$ 이고, $p = \sum_{e \in E} |e|$ 라고 하자. (일반 그래프에서 $p = 2|E|$). 또한 $\lambda > 0$ 를 입력으로 주어진 파라미터라고 하자. $O(p \ln(W) \log((m+n)^2/p) 1/\epsilon)$ 시간에

- G 의 densest subgraph가 모두 최소 λ 의 density를 가지거나,
- $(1-\epsilon)\lambda$ 이상의 density를 가진 subgraph를 반환하는

알고리즘이 존재한다. Theorem 1.1은 Densest subgraph problem의 결정 문제 버전이기 때문에, 최대화로 바꾸기 위해서는 모두 알다시피 이분 탐색을 해야 한다. 이분 탐색을 적용할 경우 다음과 같은 Corollary를 얻을 수 있다.

Corollary 1.2. $W = \sum_{e \in E} w(e)$ 이고, $p = \sum_{e \in E} |e|$ 라고 하자. (일반 그래프에서 $p = 2|E|$).
$O(p \ln(W) \log((m+n)^2/p)(\ln(r) + 1/\epsilon))$ 시간에 $(1-\epsilon)$ approximate densest subgraph를 얻을 수 있다.

고로, 단순하게 볼 경우, 고정된 ϵ 에 대해 $\log^2$ 정도에 작동하는 알고리즘이 존재함을 알 수 있다. 이제 아래에서 Theorem 1.1을 증명한다.

1.1. 플로우 모델링

이 알고리즘에서 사용할 플로우 모델링은, 이전에 소개한 Dual LP 모델링 에 기반한다. 위 글에 있는 Dual LP의 정의를 보고 오면 이해에 도움이 되겠지만, 정당성 증명에 꼭 필요한 부분은 아니라서 다시 유도하지는 않는다.

우리가 사용할 플로우 네트워크에는 $|V| + |E| + 2$ 개의 정점이 있다. 각 정점은 다음과 같은 네 종류 중 하나이다.

- 소스 s
- 싱크 t
- 모든 정점 $v \in V$ 에 대해 a_v
- 모든 간선 $e \in E$ 에 대해 a_e

정점들 간에 다음과 같이 간선이 구성되어 있다:

- 모든 간선 $e \in E$ 에 대해 $s \to a_e$, 용량 $w(e)$
- 모든 정점 $v \in V$ 에 대해 $a_v \to t$, 용량 λ
- $v \in e$ 를 만족하는 모든 (정점, 간선) 쌍 $(v, e) \in (V, E)$ 에 대해 $a_e \to a_v$, 용량 ∞

로 구성되어 있다.

모든 정점 부분집합 $U \subseteq V$ 에 대해서, 정점 집합 $\{s\} \cup \{a_v : v \in U\} \cup \{a_e : e \in E, e \subseteq U\}$ 는 가중치가
$W - f(U) + \lambda|U|$ 인 (s,t) 컷을 induce함을 관찰할 수 있다. 이 점과 min-cut max-flow theorem을 종합하면, density λ 이상의 정점 부분집합이 있음은 최대 유량이 W 미만임과 동치임을 알 수 있다.

플로우 네트워크의 Residual graph는 초기 간선의 방향과 현재 방향이 같은지에 따라 *정변*과 *역변*으로 구성되어 있다. 일반적으로 Dinic을 사용하여 최대 유량을 찾을 때, BFS를 통해서 Residual graph 상에서 source에서의 최단 거리 (level) 을 계산하여 정점을 라벨링한다. 여기서도 비슷한 개념을 사용하는데, 여기서 경로의 가중치는 경로 상의 *역변* 의 개수 로 정의된다. 즉, 통상적인 Dinic에서는 모든 간선의 가중치가 1이지만, 정변의 가중치는 0, 역변의 가중치는 1인 그래프의 최단 경로라고 보면 된다. 이후 Chapter 1에서 **거리**라는 말을 사용하면 모두 이 정의임에 유의하라.

이제 정의를 나열한다.

- $c(S)$ = Residual graph 상에서, S에서 $V - S$ 로 나가는 간선의 가중치 합. (일종의 컷 가중치이지만 residual graph 이기 때문에 정확하지는 않다.)
- $level(v) = \$Residual graph 상에서\$s$ 에서 v 로 가는 최단 거리

- $L_i = \{v \mid level(v) = i\}$ 인 정점 집합. 이 때, s 는 빼고 처리한다 (즉 $s \notin L_0$).
- $S_i = L_0 \cup L_1 \cup \ldots \cup L_i$
- $F_i = \{e \in E : a_e \in S_i\}$
- $U_i = \{u \in V : a_v \in S_i\}$

L_i 에 있는 어떠한 정점에서 $L_{i+2} \cup L_{i+3} \cup \ldots$ 로 가는 간선은 존재할 수 없다. 하지만, L_{i+1} 로 가는 간선은 존재할 수 있으며, 이 간선들은 항상 역변이어야 한다. 이러한 역변들을 *forward edge* 라고 하자. 그렇지 않고, 레벨이 같거나 감소하는 간선들은 *backward edge* 라고 하자.

1.2 Theorem 1.1의 증명

h 를 최단 경로 augmenting path의 거리라고 하자. 전략은 다음과 같다.

- h 를 무조건 늘려주는 blocking flow 알고리즘을 사용.
- 만약 이 과정에서 augmenting path가 없어지면 종료.
- h 가 충분히 커지면 ($O(\ln(W)/\epsilon)$), U_i 중 density가 충분히 큰 집합이 존재함을 증명

Lemma 1.3. 어떠한 최대 유량이 아닌 플로우 f에 대해, 모든 augmenting path의 거리가 최소 h 일 경우, 다음을 만족하는 인덱스 i 가 존재한다.

- $c(s + S_i) \leq (W^{1/h} - 1)w(F_i)$ ($w(F_i)$ 는 F_i 에 속하는 간선 가중치 합)

Proof. L_i 에서 L_{i+1} 로 가는 모든 *forward edge* 는 $a_v \to a_e$ 의 꼴을 하고 있다. 또한 이 때 $a_v \to t, s \to a_e$ 는 모두 포화 간선이다. 이러한 간선 e 들에 대해서, $e \in F_{i+1} - F_i$ 가 성립한다. $c(s + S_i)$ 에서 나가는 간선들이 정확히 이러한 forward edge 들이니, a_e 정점 기준으로 플로우가 보존됨을 감안하면 이들의 가중치 합은 $w(F_{i+1} - F_i)$ 이하이다. 고로

$c(s + S_i) \leq w(F_{i+1}) - w(F_i)$

이제 귀류법을 사용하자. 만약 모든 인덱스에 대해 $c(s + S_i) > (W^{1/h} - 1)w(F_i)$ 라면

$w(F_{i+1}) - w(F_i) > W^{1/h}w(F_i) - w(F_i)w(F_{i+1}) > W^{1/h}w(F_i)$

연립하면 $w(F_h) > Ww(F_0)$ 이다. 그런데

- 가정에 의해 $w(F_h) \leq W$ 이다.
- F_0 이 공집합이면 residual graph에서 s 에서 도달할 수 있는 정점이 없다는 것인데, f 가 최대 유량이 아니니 이는 불가능하고, $w(F_0) \geq 1$ 이다.

고로 모순을 보일 수 있다.

Lemma 1.4. 어떠한 최대 유량이 아닌 플로우 f에 대해, 모든 augmenting path의 거리가 최소 h 일 경우, 다음을 만족하는 인덱스 i 가 존재한다.

- $\frac{f(U_i)}{|U_i|} \geq W^{-1/h}\lambda$

Proof. 약간 notation을 바꾼다. $c(x, S)$ 를 플로우 x 에 대한 residual graph에서의 나가는 간선 가중치 합이라고 하자.

Lemma 1.3에 의해, $c(f, s + S_i) \leq (W^{1/h} - 1)w(F_i)$ 인 i 가 존재한다. 이 때 $c(0, s + S_i) - c(f, s + S_i) = |f|$ 인데, 이는 모든 플로우가 s 에서 t 로 가는 경로로 분해될 수 있고 (path decomposition), 이 경로는 s 에서 t 로 나가는 횟수가 들어오는 횟수보다 정확히 1 많기 때문에 각 경로가 위 값에 하나씩 기여하기 때문이다. 식을 정리하면

$W > |f|$ (f 가 최대 유량이 아님)

$W > |f| = c(0, s + S_i) - c(f, s + S_i)$ (위에서 설명)

$W > |f| \geq \lambda |U_i| + W - f(U_i) - c(f, s + S_i)$ (그래프 정의상)

$W > |f| \geq \lambda |U_i| + W - f(U_i) - (W^{1/h} - 1) w(F_i)$ (Lemma 1.3))

$W > |f| \geq \lambda |U_i| + W - W^{1/h} w(F_i)$ (F_i 에서 항상 인접한 정점 집합들을 정변으로 도달할 수 있기 때문에, $F_i \subseteq E(U_i)$)

이제 Theorem 1.1을 바로 증명할 수 있다.

Proof of Theorem 1.1. $h_0 = \lceil 2 \ln(W)/\epsilon \rceil + 2$ 로 두고, $h_1 = 2h_0 + 2$ 로 두자. 주어진 플로우 그래프에 대해서, Dinic 과 같은 일반적인 Blocking flow 알고리즘을 돌려서 가장 짧은 augmenting path의 길이가 최소 h_1 이 되도록 하자. (여기서 짧다는 것은 간선의 개수를 뜻하며, 지금까지 써왔던 역변의 개수가 아닌 일반적인 거리 기준이다.) augmenting path는 정변과 역변을 번갈아 사용하기 때문에 이 경우 모둔 augmenting path는 h_0 개 이상의 역변을 사용한다. 고로 Lemma 1.4 에 의하여, 우리가 찾은 플로우는 최대 유량을 갖거나 (W), 다음과 같은 집합을 찾을 수 있다:

$f(U) \geq W^{-1/h_0} \lambda |U| \geq e^{-\epsilon/2} \lambda |U| \geq (1 - \epsilon) \lambda |U|$

Link-cut tree를 사용하면 $O(p \log((m+n)^2/p))$ 시간에 Blocking flow를 사용할 수 있음이 잘 알려져 있다 (Goldberg-Tarjan '87). 고로 전체 알고리즘이 $O(p \ln(W) \log((m+n)^2/p))$ 에 작동하며 이후 U는 쉽게 $O(p)$에 구할 수 있다. ■

Chapter 2. Greedy Peeling Algorithm for DSS

이 챕터에서는 Densest subgraph 문제를 일반적인 submodular function에 대해서 일반화하는 방법을 다루며, 이 과정에서 Greedy peeling algorithm을 잠시 리뷰한다.

Greedy peeling algorithm은 Charikar가 2000년 제안한 Densest subgraph의 2-approximation algorithm이다. 이 알고리즘은 이론 및 실전 모두에서 가치가 있지만, 특히 실제 그래프 데이터 분석에 있어서 실용적 가치가 아주 높은 알고리즘이다. 알고리즘이 간단해서 검증하기 쉽고, 간단하기 때문에 속도도 빠르고, 실제 데이터에서 퍼포먼스가 좋기 때문이다. (다양한 $1+\epsilon$-approx 알고리즘이 그리 간단하지 않다는 것과 대비된다.)

이 단락에서는 Densest subgraph 문제를 일반적인 Densest supermodular subset problem의 형태로 일반화하며 해석하는 방법, 또한 이 과정에서 상기하면 좋은 성질들을 나열한다.

Densest supermodular subset problem (DSS) 는 다음과 같은 성질을 만족하는 subset function $f : 2^V \to \mathbb{R}_{\geq 0}$ 에 대해서 작동한다:

- Supermodular: $f(x \cap y) + f(x \cup y) \geq f(x) + f(y)$ for $x, y \subseteq V$
- Monotone: $f(x) \leq f(y)$ if $x \subseteq y$

이 문제는, 이 함수에 대한 Oracle을 사용하여 $\frac{f(S)}{|S|}$ 를 최대화하는 $S \subseteq V$ 를 찾는 문제이다.

Densest subgraph problem (DSG) 를 DSS의 인스턴스로 보기 위해서는 $f(S) = (S$ 내부의 간선 개수) 라는 함수를 정의하면 된다. 이 함수는 위에 있는 모든 조건을 만족하기 때문이다. 이 때 몇 가지 파라미터를 정의할 수 있다.

- $f(v|S) = f(S \cup \{v\}) - f(S)$
- $c_f = \max_{S \subseteq V} \frac{\sum_{v \in S} f(v|S-v)}{f(S)}$

먼저, 첫 번째 파라미터는 $G[v|S]$ 에서 v 의 degree와 동일하다. 즉, supermodular function에서의 ***degree*** 의 비유라고 보면 될 것이다.

두 번째 파라미터에 대해서는 추가적으로 다음과 같은 정의를 할 수 있다. supermodular function f 가 r-decomposable 하다는 것은, 이들이 "크기가 작은" supermodular function 의 합으로 표현 가능하다는 것이다. 엄밀히 말해서, 어떠한 양의 정수 m 에 대해서 $V_i \subseteq V$, f_i 가 존재해서

- $|V_i| \leq r$
- $f_i : 2^{V_i} \to \mathbb{R}_+$ 인 supermodular function
- $f(S) = \sum_{i=1}^{m} f_i(S \cap V_i)$ for all $S \subseteq V$.

$G[S]$ 의 간선 개수는 2-decomposable supermodular function이다. 각 간선의 기여도를 따로 따져보면 쉽게 알 수 있다. rank-r hypergraph에 대해서도 같은 이유로 r-decomposable 하다.

Proposition 2.1 모든 r-decomposable supermodular function f 에 대해서 $c_f \leq r$ 이다.

이에 대한 증명은 정의를 따라가면 어렵지 않게 가능하다.

Greedy peeling

Greedy peeling은 다음과 같은 알고리즘이다:

- $v_1 = argmin_{v \in V} f(v|V-v)$ 를 찾은 후, v_1 을 그래프에서 지운다.
- 남은 그래프에서 이를 반복하면, Vertex ordering $\{v_1, v_2, \ldots, v_n\}$ 을 얻을 수 있다.
- 모든 $1 \leq i \leq n$ 에 대해서, $\{v_i, \ldots, v_n\}$ 중 가장 densest한 것을 반환한다.

이 알고리즘은 Charikar의 알고리즘을 supermodular function에 맞게 일반화한 것이다. 고로 유사한 bound 역시 성립한다.

Theorem 2.2 Greedy peeling algorithm은 $\frac{1}{c_f}$-approximation 알고리즘이다.

이 peeling algorithm에 대해서는 다음과 같은 흥미로운 성질 역시 성립한다. 이후 내용과 연관은 없지만 단순 참고를 위해 설명한다. $G = (V, E)$ 의 k-core는, $G[S]$ 의 모든 정점의 차수가 k 이상인 **maximal** 정점 부분집합 $S \subseteq V$ 를 뜻한다. k-core를 찾는 간단한 알고리즘은, 차수가 k 미만인 정점들을 반복해서 제거하는 알고리즘이다. 차수가 가장 낮은 것을 제거하더라도 이것과 같은 효과를 보일 수 있으며, 이를 통해서 k-core가 unique함 역시 보일 수 있다 (논문에서는 trivial하다고 하는데, 무슨 자신감인지 잘 모르겠다. 다만 직관 자체는 충분하다고 생각한다.)

S_1 이 k_1-core 이고 S_2 가 k_2-core 면 $k_1 \leq k_2$ 일때 $S_2 \leq S_1$ 이다. k-core가 unique하기 때문에, 이는 Greedy peeling algorithm에 의해서 자연스럽게 유도할 수 있다. 이 관찰은 $|G[S]|$ 라는 함수 말고도 supermodular function 전체에 대해서 일반화가 가능하고, 이를 정리하면 다음과 같은 Theorem으로 표현할 수 있다.

Proposition 2.3 $f : 2^V \to \mathbb{R}_+$ 가 supermodular function이라고 하고, $v_1, v_2, \ldots, v_n$ 을 f 에 greedy peeling algorithm을 적용했을 때 나온 ordering이라고 하자. 임의의 γ 에 대해서 f 의 γ-core $C(\gamma)$ 는 unique하고, i 를 $f(v_i|S_i - v_i) \geq \gamma$ 가 성립한 최소 i 라고 하면 $C(\gamma) = S_i$ 이다.

어쨌든 전체 내용과는 크게 상관 없는 명제라 증명은 생략한다. (그렇게 어렵지는 않다)

여담으로, 이러한 ***core*** 의 개념은 induced subgraph의 minimum degree를 최대화하는 알고리즘임을 상기하자. 우리의 목표는 average degree를 최대화하는 것인데, minimum degree의 최대화를 통해서 이를 approximate할 수 있다는 것이 흥미롭다.

Chapter 3. Iterative Peeling

일반적인 Greedy peeling algorithm이 반환하는 output은 unique하기 때문에, 가용 가능한 시간과 상관없이 내놓을 수 있는 solution set이 한정적이라는 한계점이 존재한다. 이 알고리즘의 한계를 극복하기 위해 여러 번의 iteration을 통해서 답의 Quality를 반복적으로 증가시키려는 노력인 *Iterative peeling* 알고리즘이 등장했다.

Iterative peeling algorithm (이하 *Greedy++* 라 부른다) 은 Greedy peeling의 간단한 변형인데 알고리즘은 다음과 같다.

- 초기에 $L = \{l_1, l_2, \dots, l_v\}$ 라는 크기 $|V|$ 의 배열을 만든다. (이 L 을 load라고 한다.)
- 다음을 T 번 반복한다
- $L_v + deg(v)$ 를 기준으로 Greedy peeling을 하여 ordering을 뽑는다.
- Ordering의 모든 suffix를 답의 후보로 두고 최적을 찾는다.
- 모든 $v \in V$ 에 대해, L_v 에, ordering에서 해당 정점을 뽑을 "당시의" degree를 더해준다.
- 확인한 $T \times V$ 개의 후보 중 최적을 반환한다.

이는 Supermodular function에 대해서도 일반화할 수 있다 (이하 *Super-Greedy++* 라 부른다). 위에서 사용한 degree의 analogy를 가져오면 된다.

- 초기에 $L = \{l_1, l_2, \dots, l_v\}$ 라는 크기 $|V|$ 의 배열을 만든다. (이 L 을 load라고 한다.)
- 다음을 T 번 반복한다
- $L_v + f(v|S-v)$ 를 기준으로 Greedy peeling을 하여 ordering을 뽑는다.
- Ordering의 모든 suffix를 답의 후보로 두고 최적을 찾는다.
- 모든 $v \in V$ 에 대해, L_v 에, ordering에서 해당 정점을 뽑을 "당시의" $f(v|S-v)$ 를 더해준다.
- 확인한 $T \times V$ 개의 후보 중 최적을 반환한다.

이 알고리즘의 저자들은 알고리즘이 $O(1/\epsilon^2)$ 번 반복하면 $(1-\epsilon)$ approximation으로 수렴한다고 추측하였지만 증명하지는 않았다. 만약에 이 사실이 성립한다면, 이는 Greedy peeling 의 해를 iterative procedure로 refine할 수 있는 방법과 동시에, theoretical한 bound와도 연계시킬 수 있기 때문에 상당히 매력적인 알고리즘이다. 이 단락에서는 이것이, 어느 정도는 사실임을 증명한다.

Theorem 3.1. $T \geq O(\frac{\Delta \ln(n)}{\lambda \epsilon^2})$ 에 대해 *Super-Greedy++* 는 $(1-\epsilon)$-approximation 알고리즘이다. 이 때 $\Delta = max_{v \in V} f(v|V)$ 이다.

이제 이후 내용은 모두 Theorem 3.1을 증명하는 데 사용한다. Theorem 3.1의 증명은 여러 잘 알려지지 않은, technical한 개념들에 강하게 의존한다. 이 글은 이러한 technical한 개념에 특히 초점을 맞추어서, supermodular function을 분석하는 데 사용할 수 있는 다양한 도구들을 익히는 데 집중한다.

Lovasz extensions

Supermodular function은 subset 2^V 에서 $\mathbb{R}_{\geq 0}$ 으로 가는 함수이다. 이를 Continuous하게 일반화한 것을 Lovasz extension이라고 한다. 정확히는, Lovasz extension은 $[0,1]^V \to \mathbb{R}_{\geq 0}$ 으로 가는 함수로, 각 V 개의 입력이 binary인 supermodular function과는 다르게 $[0,1]$ 사이의 real을 입력으로 받는다. 아래 구체적인 정의를 소개한다.

Defintion 3.2. Supermodular function f 가 주어졌을 때, Lovasz extension $\hat{f}(x) = \int_{\tau=0}^{1} f(\{v \in V | x_v \geq \tau\}) d\tau$ 이다.

Lovasz extension을 사용하면 DSS 문제를 매우 compact하게 표현할 수 있다. 다음과 같은 프로그램을 생각해 보자:

> maximize $\hat{f}(x)$ over $x \in \mathbb{R}_{\geq 0}^V$ such that $\sum_{v \in V} x_v \leq 1$

Lemma 3.3 위 프로그램은 DSS를 정확히 표현한다. 다시 말해:

- density λ 인 집합 S 에 대해, 벡터 $x_i = 1/|S|$ for all $i \in S$ 는 위 프로그램의 feasible solution이며 $\hat{f}(x) = \lambda$ 이다.
- 위 프로그램의 feasible solution x 에 대해, $S_\tau = \{v \in V | x_v \geq \tau\}$ 의 density가 최소 $\hat{f}(x)$ 인 $\tau \in [0, 1]$ 이 존재한다.

Proof. 첫 번째 항목은 $\hat{f}(x)$ 의 정의에서 바로 유도된다. 두 번째 항목은 귀루법을 사용한다. x를 고정시킨 후, 모든 집합 S_τ 의 density가 $\hat{f}(x)$ 미만이라고 생각하면, 이를 길이 1의 구간에 대해서 적분한 결과가 $\hat{f}(x)$ 일 수 없다.

Remark. 2022년 1월 과제 Efficient Primal-Dual Algorithms for MapReduce 에 나온 LP Relaxation과의 유사성에 주목하라.

이제 DSS 문제를 Lovasz extension을 사용하여 표현하는 방법을 익혔으니, 이를 푸는 방법에 집중하자.

S_V 를 V 의 모든 원소의 순열의 집합이라고 하자. 어떠한 permutation $\sigma \in S_V$ 에 대해, 이에 대응되는 벡터 $q(\sigma) \in \mathbb{R}^V_{\geq 0}$ 를 다음과 같이 정의하자.

- $q_v(\sigma) = f(v | \{w : w <_\sigma v\})$ ($w <_\sigma v$ 는 순열 상에서 w 가 v 보다 먼저 나옴을 뜻한다.)

이제 다음과 같은 Lemma가 성립한다.

Lemma 3.4. 모든 $x \in [0, 1]^V$ 에 대해, $\hat{f}(x) = min_\sigma(x \cdot q(\sigma))$ 이다. 이에 더해, 이 최솟값은 σ 가 모든 원소를 x_v 가 감소하는 순서로 나열했을 때 얻을 수 있다.

Proof. 논문에서는 citation을 들며 증명을 따로 하지 않았고, 본인도 정말 아쉽게도 이 부분에 대한 증명을 넘어간다. 하지만 이 부분의 증명을 통해서 얻을 수 있는 것이 많은 것 같으니, 관심이 있다면 꼭 한번 알아보길 바란다. 대략 다음과 같은 호흡으로 증명할 수 있다.

- $q(\sigma)$ 가 f 의 base contrapolymatroid의 정점 집합과 일치한다.
- $\hat{f}(x)$ 는 base contrapolymatroid 상 모든 점 p 에 대해 $x \cdot p$ 를 최소화한다.
- $x \cdot q(\sigma)$ 의 최솟값은 위에서 설명한 Greedy algorithm을 통해서 얻을 수 있다.

아마 [Convex Analysis and Optimization with Submodular

Functions: a Tutorial](https://hal.archives-ouvertes.fr/hal-00527714/document) 의 앞 부분에서 대략 이런 내용을 설명을 하는 것 같다.

하여튼 이제는 LP로 표현할 수 있다. $\lambda^\star$ 를 DSS의 최적 값이라고 하고, 다음과 같은 LP를 정의하자:

> minimize $\sum_{v \in V} x_v$ over $x \geq 0$ such that $x \cdot q(\sigma) \geq \lambda^\star$ for all $\sigma \in S_V$

Dual은

> maximize $\lambda^\star \sum_{\sigma \in S_V} y_\sigma$ over $y : S_V \to \mathbb{R}_{\geq 0}$ such that $\sum_{\sigma \in S_V} y_\sigma q_v(\sigma) \leq 1$ for all $v \in V$

Solving the dual with MWU

이제 마지막으로, 이 Dual을 Width-independent MWU로 해결하고, 이 과정이 *Super-Greedy++* 알고리즘과 대응됨을 보인다. Width-Independent MWU에 대해서는 Introduction to Width-Independent MWU 글에서 설명하기 때문에, 이 글에서는 따로 이 테크닉과 그 정당성에 대해서 설명하지 않는다.

MWU 프레임워크를 위 Dual LP에 적용하면, 대략 다음과 같은 pseudocode가 나온다. 이 프로시저를 *Order-Packing-MWU* 라고 부른다.

- 인자로 $f : 2^V \to \mathbb{R}_{\geq 0}, \epsilon \in (0,1), T \in \mathbb{N}, \alpha \geq 1$ 을 받는다.
- 초기 값을 $w^{(0)} = \mathbb{1}_V, y^{(0)} = \mathbb{0}_{S_V}, \eta = \frac{\ln(n)}{\epsilon}$
- $t = 1, 2, \ldots, T$ 에 대해...
- $\sigma^{(t)}$ 를 $w^{(t)} \cdot q(\sigma) \leq \alpha min_{\sigma' \in S_V} w^{(t)} q(\sigma')$ 가 만족되는 임의의 σ 로 정의.
- $y^{(t)} = y^{(t-1)} + \frac{1}{\lambda^\star T} \mathbb{1}_{\sigma^{(t)}}$
- 모든 $v \in V$ 에 대해, $w_v^{(t+1)} = w_v^{(t)} \times e^{\frac{\eta}{\lambda^\star T} q_v(\sigma^{(t)})} = e^{\eta \sum_\sigma y_\sigma^{(t)} q_v(\sigma)}$

Lemma 3.5. $T \geq O(\Delta \ln(n) / \lambda^\star \epsilon^2)$ 이고, $\alpha \geq 1$ 인 경우, *Order-Packing-MWU* 는 다음을 보장한다. (증명은 Width-Independent MWU의 통상적인 증명과 동일)

- $\beta^{-1} y^{(T)}$ 는 Dual의 β^{-1}-approximate 해이다. 이 때 $\beta = (1+\epsilon)\alpha$ 이다.
- 어떠한 $t \in [T]$ 에 대해, $\beta w^{(t)} / (1 \cdot w^{(t)})$ 는 Primal LP의 β-approximate 해이다.

이제 *Super-Greedy++* 와 *Order-Packing MWU* 가 동치임을 보인다.

Lemma 3.6 *Super-Greedy++* 는 $\alpha = e^\epsilon$ 인 *Order-Packing-MWU* 로, 다음과 같은 대응 관계를 가진다:

- 모든 $v \in V$ 에 대해, Super-Greedy의 $l_v^{(t)}$ 는 $\lambda^\star T \sum_\sigma y_\sigma^{(t)} q_v(\sigma)$에 대응된다. 즉 $w^{(t)} = e^{(\eta / \lambda^\star T) l_v^{(t)}}$ 이다.
- Super-Greedy의 $\pi^{(t)}$ 는 $\sigma^{(t)}$ 에 대응된다.

증명은 특별히 자명하지 않지만 수식 전개가 너무 길다. 여기서는 생략한다.