Использования Shapley Value для многокритериального ранжирования ребер графа

Ильгам Магданович Латыпов

Научный руководитель: к.т.н. Ю. В. Дорн

16 декабря 2023 г.

Цели исследования

Задачи:

- Дана функция на графе. Например: максимальный поток из вершины А в вершину Б. Нужно научиться выделять подграф наименьшей стоимости, который минимизирует/максимизирует эту функцию.
- Дан набор функций на графе. Нужно научиться выделять подграф, который обеспечивает "оптимальность"по всему набору и при этом имеет небольшую стоимость. "Оптимальность"определим потом.

Идея: Ранжировать ребра по важности с помощью Shapley Value и применить жадный алгоритм для отбор ребер.

Проблемы:

- Точный подсчет Shapley Value – NP задача.
- Жадные алгоритмы типа рюкзака выдают не оптимальный подграф.

про Shapley Value

Введем обозначения: $\mathcal{N}$ – множество игроков. $v : 2^{\mathcal{N}} \to \mathbf{R}$ – функция выигрыша игры. Она показывает какой выигрыш будет, если в игре участвует некоторый набор игроков.

SV(shapley Value) – способ оптимального распределения выигрыша между игроками в коалиционной игре.

$$\Phi_i(v) = \sum_{S \subseteq \mathcal{N} \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} (v(S \cup \{i\}) - v(S))$$

То есть каждый игрок получает матожидание изменения выигрыша при добавлении этого игрока в игру.

Пример: пусть игроки не взаимодействуют между собой, то есть $v(S) = \sum_{i \in S} v(\{i\})$. тогда $\Phi_i(v) = v(\{i\})$.

Как считать SV

Есть множество работ посвященных подсчету SV. Найденные нами работы можно отнести к двум типам:

- Использование вида $v(S)$. В таких работах исследуется функция v и показывается, например, что ненулевых слагаемых полиномиальное количество. И на этом строится полиномиальный алгоритм.
- Методы Монте-Карло. Функция представляется суммой большого числа слагаемых. Так что можно семплировать некоторые слагаемые и получить приближенное значение функции.

• В некоторых работах отмечается, что аппроксимация SV зачастую слишком сильно отличается от реальных значений.

Approximating the SV without Marginal Contributions

Использовали результаты работы **Approximating the SV without Marginal Contributions** для приближенного вычисления SV.

Авторы исследовали вычисление SV для n игроков, при условии, что вычисление v можно запускать не более чем фиксированное число T раз.

В предложенном методе дисперсия SV будет равна $O(ln(n)/(T-n))$. Это значение лучше чем в других рассмотренных работах.

Аггрегация результатов: рюкзак

SV может быть проинтерпретирован как важность ребер для данного v. Из этого соображения появляется желание использовать рюкзак как способ выбрать подграф.

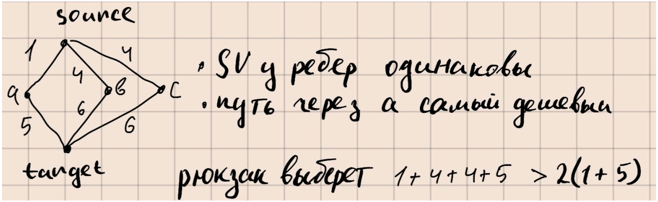
Однако этот метод может выдавать решения слишком далекие от оптимальных:

Обозначим $G(V, E)$ – граф

$$f_G(i,j) = \{\text{len of shorhtest path from i to j}\}$$

и рассмотрим

$$v_{i,j}(S) = [f_G(i,j) == f_{G(V,S)}(i,j)]$$



Алгоритмы отбора ребер

Algorithm 1 greedy deletion

Input : $\mathcal{N}, v$
deleted $\leftarrow \emptyset$ ▷ list of deleted players
$S \leftarrow \mathcal{N}$ ▷ non deleted players
while $v(S) \neq 0$ and $S \neq \emptyset$ **do**
    $i \leftarrow \arg\max_i v(S \backslash \{i\})$
    deleted $\leftarrow$ deleted $\cup \{i\}$
    $S \leftarrow S \backslash \{i\}$;
end while
return deleted

Algorithm 2 SV deletion

Input : $\mathcal{N}, v$
$S \leftarrow \mathcal{N}$ ▷ игроки которые остались
deleted $\leftarrow \emptyset$ ▷ list of deleted players
arr $\leftarrow \arg\mathrm{sort}(SV(v))$ ▷ SV(v) – массив значений SV. Сортируем игроков по увеличению SV
for $i \in$ arr **do**
    if v(S $\backslash \{i\}) > 0$ **then**
        $S \leftarrow S \backslash \{i\}$;
        deleted $\leftarrow$ deleted $\cup \{i\}$;
    end if
end for
return deleted

Рис.: proposed algorithms

Value Function

Нужно построить value function v. Предлагается ее строить в виде произведения функций от базовых метрик.

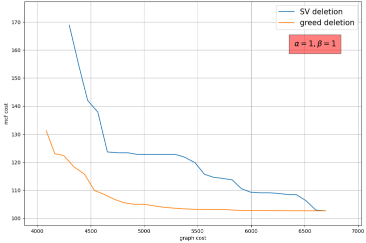
Рассмотрим задачу multicommodity flow. Хотим получить граф подешевле и чтобы стоимость потоков была не большой.

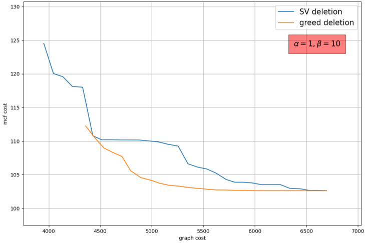
1. graph_cost $\in [0, \text{max\_cost}]$. g(c) $= g(c) = (2 - \frac{c}{\text{max\_cost}})$
2. flow_cost $\in [\text{min\_flow\_cost}, \infty]$. $h(f) = \frac{\text{min\_flow\_cost}}{f}$.

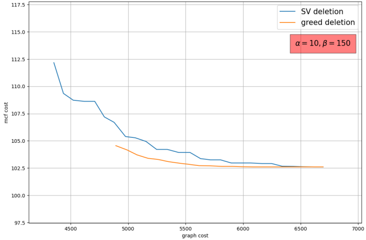
$$v(S) = \left(2 - \frac{c}{\text{max\_cost}}\right)^{\alpha} * \left(\frac{\text{min\_flow\_cost}}{f}\right)^{\beta} \qquad (1)$$

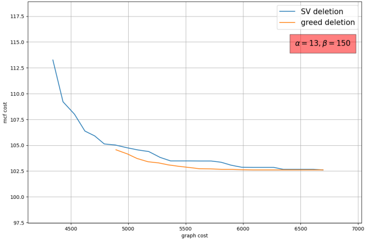
Рассмотрим результаты работы алгоритма при разных параметрах.

Эксперименты









Эксперименты

Теперь рассмотрим три метрики: cost, flow_cost,λ_2
Ведем функции множители, которые представляют эти метрики в v:

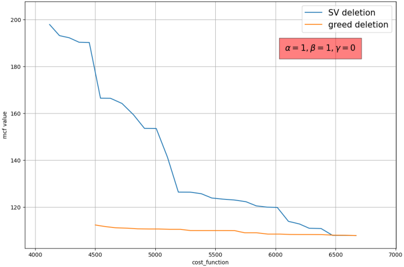
1. graph_cost $\in$ [0, max_cost. g(c) $= g(c) = 1 + (1 - \frac{c}{\text{max\_cost}})$
2. flow_cost $\in [\text{min\_flow\_cost}, \infty]$. $h(f) = \frac{\text{min\_flow\_cost}}{f}$.
3. $\lambda_2 \in [0, 1]$. $r(\lambda_2) = \frac{1+\lambda_2}{2}$

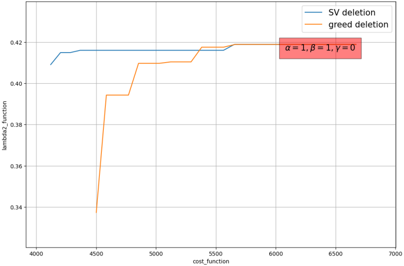
И получим функцию качества:

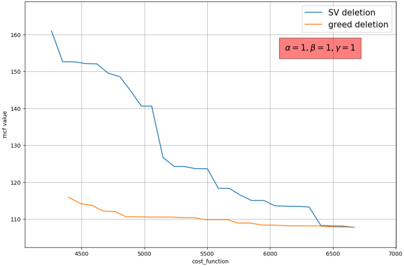
$$v_2(S) = \left(\frac{1}{2}(2 - \frac{c}{\text{max\_cost}})\right)^{\alpha} * \left(\frac{\text{min\_flow\_cost}}{f}\right)^{\beta} * \left(\frac{1+\lambda_2}{2}\right)^{\gamma} \tag{2}$$

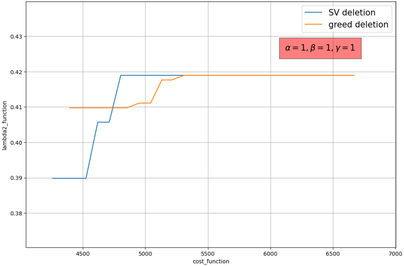
И снова рассмотрим работу алгоритмов при разных параметрах

Эксперименты

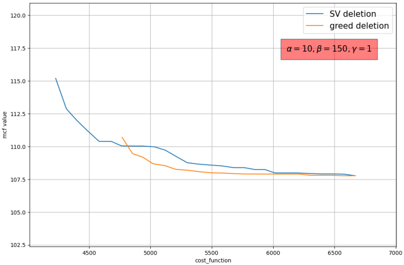


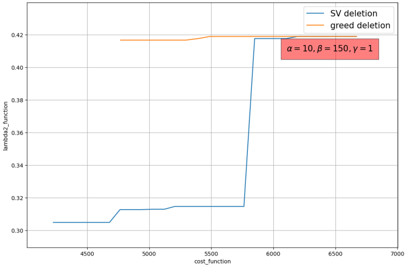


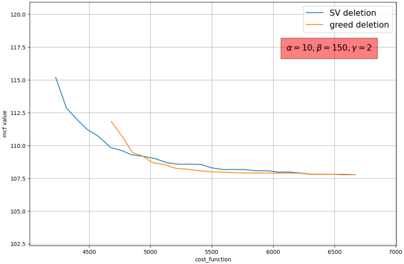


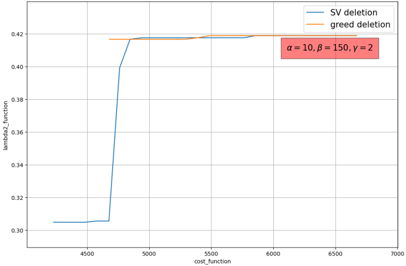


Эксперименты









При возведении в степень функция растягивается по оси y, из-за чего дифференцирующие свойства составляющих value function возрастают. То есть если взять две топологии, такие что $v_1 > v_2$, то при возведении компонент в степень отношение $\frac{v_1}{v_2}$ увеличится, а значит первая топология становится еще более предпочтительной.

Итоги

1. rezults
   - Предложен способ построения value function, который работает в примерах.
   - Результаты оформлены в виде статьи. Код написан с примером применения и закинут на gitlab.
2. next
   - Сделать автоматический подбор коэффициентов.
   - Обосновать почему работает метод.
   - Научиться использовать аддитивность SV по value function.
3. fails
   - были рассмотрены способы аппроксимации SV с помощью нейронок, но там не получили каких-то результатов.
   - Был рассмотрен способ агрегирования результатов построением путей. Но путей может быть слишком много и также другие проблемы.