Constructing Top-k Routes with Personalized Submodular Maximization of POI (point of interest) Features

Виконали: Борсук В., Косаревич І.

Постановка задачі

Нам дано карту POI(Points of interest) у вигляді графа G(V, E), де кожне ребро зважене $T_{i,j}$(вагою наприклад може бути час добирання із v_i у v_j) та кожна вершина містить вектор H, який характеризує цю вершину. На вхід подається якись запит користувача Q. На вихід потрібно повернути чергу з пріорітетом, розміром k, різних шляхів (тобто k шляхів P_v), таких, що найбільше відповідають заданому запиту Q(вищий пріорітет буде у того шляху який більше відповідає заданому запиту).

Граф G(V, E) – орієнтований або неорієнтований граф, де V – це множина вершин, E – множина ребер.

Вектор H – вектор розміром n, який складається з фіч $h_{1,}$ h_2 … h_n. Фічею h може бути наприклад кількість чи якість кафе на певній POI. Значення фічі варіюєьстя від 0 до 1.

Запит Q = (x, y, b, w, θ, Φ), де x — початкова, y — кінцева вершина; b — бюджет наявний у користувача, w — вектор побажаня користувача щодо кожної фічі з *H, θ* — вектор, який фільтрує значення на кожній h із H, такий що якщо значення на h менше за відповідне значення із вектора *θ*, то значення на h встановлюється в 0, Φ — монотонна невідємна субмодулярна множинна функція, яка робить певне співвідношення між значенням h на певній POI та іншими POI на шляху P. Обовязковими значеннями є тільки перші три: x, y, b, останні три можуть братися за умовчуванням.

Шлях P — список вершин, який починається із x та закінчується y і характеризується вартістю cost(P_v).

Опис підходу

Нагадаємо, на виході ми хочемо отримати множину із k шляхів, таких, що для кожного P_v cost(P_v) <= b(бюджет). Для кожного шляху ми обраховуємо значення Gain(P_v, Q) = $\Sigma_h w_h \Phi_h(P_v)$, де w_h — значення фічі h шляху P_v, Φ_h — функція (зазначена в описі запиту Q) для певної фічі h. Це значення Gain(P_v, Q) визначає наближеність певного шляху P_v до вимог користувача в Q. Обрані k шляхів повинні мати найвище значення Gain.

Перед застосуванням алгоритму проводиться індексація карти POI для спрощення роботи із нею. Її результат набір вершин V_Q до яких і буде застосовано алгоритм, а також 2 множини FI(Feature Index) та HI(Hop Index).

Алгоритм є варіацією динамічного програмування. Компактним станом називається множина всіх шляхів з однаковим набором вершин але різним порядком їх розташування. Запропонований підхід будує дерево компактних станів, на основі якого вершини якого є шляхами на певному етапі формування(названі відкритими шляхами, тобто такими які починаються в x і містять певну кількість вершин відмінних від y). Розширення шляху відбувається шляхом додавання певної вершини j, яка належить копактному стану і відмінна від y. Причому, на одній ітерації не будується усе дерево, а його певна гілка. Знайдений на певній ітерації шлях перевіряється на відповідність умові cost(P) <= b. Якщо ця умова не виконується, то побудова гілки припиняється. Повноноцінний шлях отримується додаванням до знайденого відкриго шляху вершини y. До того ж на кожній ітерації застосовується так звані скорочувальні стратегії(prunning strategies), які значно зменшують кількість можливих шляхів. В результаті ми отримуємо чергу з пріорітетом із k шляхів. Пріорітетнішим є той шлях, який має вищий Gain($P_{v,}$ Q).

Приклади задач

Даний алгоритм може бути застосований до таких класів задач, як Orienteering Problem, Sequential Location Recommendation, Trajectory Search. Конкретний приклад: ви вперше відвідуєте деяке місто і хочете обійти його найвизначніші місця. Цей алгоритм складе k маршрутів які найбільше задовольняють ваші побажання.

Алгоритм

Позначення:

Q = (x, y, b, w, θ, Φ), де x - початок, y - кінець, b - бюджет, w - вектор фіч, θ - вектор фільтр, Φ - feature aggregation functions.

V_Q - набір POI (вершин)

FI_Q - словник фіч, які вказують на відсортований по спаданню ваги даної фічі список з кортежами (вершина, вага)

HI_Q - список аершин, де кожна вершина посилається на список з елементами (вершина, відстань до цієї вершини), який посортований за зростанням відстаней

C^- - відкритий шлях (можливий для розширення)

C - compact state (група відкритих шляхів, які містять однакові вершини)

C_L - список відкритих шляхів які містять POI з C

I - набір вершин, які є можливими для розширення C^-

Рекурсивна функція:

PACER(C^-, I)

Required: Q = (x, y, b, w, θ, Φ), V_Q, FI_Q and HI_Q to compute Gain(C) and cost($\mathcal{P}$), and k

Parameters: compact state C^- and the set of POIs **I** for extending C^-

Output: a priority queue topK

1 forall POI i in set **I** in order do

2 $C \leftarrow \{i\} \cup C^-$;

3 compute Gain(C);

4 forall POI j in C do

5 $C^{-j} \leftarrow C \setminus \{j\}$;

6 $\mathcal{P}^- \leftarrow$ the dominating route in C_L^{-j} such that cost($\mathcal{P}^- \rightarrow$ j) is minimum; // prune-1

7 $\mathcal{P} \leftarrow \mathcal{P}^- \rightarrow$ j;

8 if cost($\mathcal{P} \rightarrow$ y) ≤ b then

9 Compute UP using Eqn. (13);

10 if Gain(C) + UP ≥ Gain(topK[k]) then

11 insert route $\mathcal{P}$ into C_L; // prune-2

12 UpdateTopK(C_L, topK);

13 PACER(C, prefix of i in **I**);

Коментарі:

1-3: розширення C^- для кожної вершини *i* з множини вершин *I*, таким чином створюючи новий compact state C і обчислюємо для нього Gain(C).

4-11: на даних етапах генеруються домінуючі відкриті шляхи C_L. А саме, кожна j є C обирається як кінцева POI шляху а інші POI C^{-j} це вже попередньо порахований compact state при попередніх викликах даної функції.

6: вибирається домінуючий шлях $\mathcal{P}^-$ в поточному C_L^{-j}, для чого ми використовуємо pruning-1, який базується на виборі лише того шляху з C_L^{-j}, який є можливим та з мінімальною вартістю.

8-11: якщо вартість отриманого шляху не є більшою за дозволену вартість, то використовуємо pruning-2, щоб перевірити чи сума Gain(C) та оціненої верхньої межі (UP) не є меншою за gain к-того шляху в topK, і якщо так, то $\mathcal{P}$ додається до C_L.

$UP = \sum_{j=1}^{l-1} \delta_{ij} + \lambda\delta_{il}$, де $\delta_i = \Delta Gain(\{i\}|P_v)$

12: закритий шлях $P \rightarrow y$ для відкритого шляху $\mathcal{P}$ з C_L такий, що $P \rightarrow y$ має найменшу вартість, добавляється до topK.

13: C використовується в наступному рекурсивному виклику функції разом з префіксом для i в множині вершин I.

Аналіз складності

Для визначення складності алгоритму потрібно виділити такі дві характеристики, як величина масиву V_Q (масив точок-кандидатів) позначимо її n та максимальна довжина досліджених шляхів p. Завдяки застосуванню першого скорочення(prunning 1) ми маємо на деякому рівні L найбільше n компактних станів, кожен з яких представляє найбільше L шляхів, кожен з яких обраховується лише раз. Найбільша кількість досліджених шляхів на цьому рівні L, згідно підрахунків авторів, в чиїх обчисленнях було використано правило Паскаля про вибір k речей із n речей та згідно факту p<n, складе

$$\sum_{l=1}^{p} l\binom{n}{l} = n\sum_{l=1}^{p}\binom{n-1}{l-1} \approx n\left(\binom{n-1}{p-1}+\binom{n-1}{p-2}\right) = n\binom{n}{p-1} = \frac{n}{(n-p+1)(p-1)!}\ \frac{n!}{(n-p)!}$$

Тому складність при застосуванні першого скорочення буде:

$$O\left(\frac{n!}{(p-1)!(n-p)!}\right)$$

У разі застосування другого скорочення(prunning 2), із певним відсотком γ вилучених шляхів із досліджених алгоритмом, складність складе:

$$O\left((1-\gamma)\frac{n!}{(p-1)!(n-p)!}\right)$$