Побудова Top-k шляхів з персональною субмодулярною максимізацією властивостей(features) POI(точок інтересу)

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 $h_1, h_2 \dots h_n$. Фічею h може бути наприклад кількість чи якість кафе на певній POI. Значення фічі варіюється 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)

CL- 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 ← {i} ∪ C-;
3       compute Gain(C);
4       forall POI j in C do
5               C-j ← C \ {j};
6               P⁻ ← the dominating route in CL-j such that cost(P⁻ → j) is minimum;
                                                                    // prune-1
7               P ← P⁻ → j;
8               if cost(P → y) ≤ b then
9                       Compute UP using Eqn. (13);
10                      if Gain(C) + UP ≥ Gain(topK[k]) then
11                              insert route P into CL; // prune-2
12      UpdateTopK(CL, 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 Ci обчислюємо для нього Gain(C).

4-11: на даних етапах генеруються домінуючі відкриті шляхи CL. 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 CL-j, для чого ми використовуємо pruning-1, який базується на виборі лише того шляху з CL-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 CL.

UP=j=1l-1i_j + il , де i=Gain({i}|Pv)

12: закритий шлях Py для відкритого шляху $\mathcal{P}$ з CLтакий, що Py має найменшу вартість, додається до topK.

13: C використовується в наступному рекурсивному виклику функції разом з префіксом для і 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

l=1pl(ln) =nl=1p(l-1n-1) n((p-1n-1)+(p-2n-1))=n(p-1n)=n(n-p+1)(p-1)! n!(n-p)!

Тому складність при застосуванні першого скорочення буде:

O(n!(p-1)!(n-p)!)

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

O((1-)n!(p-1)!(n-p)!)

Опис даних

Вхідні дані задачі:

Карта точок інтересу(POI Map) складається з двох частин:

- граф G(V, E), де V – множина вершин, E – множина ребер; заданий матрицею суміжності
- список векторів фіч(один вектор для кожної вершини)

запит користувача Q

Спочатку генерувалось число N – кількість вершин шляхом вибору із множини псевдовипадкових чисел на проміжку від *5* до *8*.

Генерація матриці суміжності

Для спрощення було взято повний граф K_n. Для заданого значення N генерувалася матриця суміжності наступним чином. Так як матриця суміжності є симетричною матрицею із нулями на головній діагоналі, генерувалася лише та частина матриці, яка знаходиться над головною діагоналлю. Значення відстаней між вершинами вибирались із множини псевдовипадкових чисел на проміжку від *5 до 100.* Матриця була представлена у вигляді двовимірного масиву розміром (N-1)xl, де l: від N-1 до 1 для кожного n: від 1 до N-1. Наприклад, для графа з трьома вершинами візьмемо таку матрицю суміжності:

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{bmatrix}$$

спрощена матриця виглядатиме так:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & NIL \end{bmatrix}$$

Із прикладу помітно, що значення $A_{i,j}$ оригінальної матриці при $i < j$ перемістилося на позицію $A_{i,j-i}$. Саме за такою формулою відновлювались оригінальні значення матриці. Тобто для всіх j таких що j > i в оригінальній матриці, j^, в спрощеній формі, є рівним j-i. Для значень (i,j), таких що j < i бралось значення (j, i) згідно властивості симетричних матриць. Для (i, j), таких що j=i, - 0.

Генерація веторів фіч

Спочатку генерувалося значення H – число фіч вибором із множини псевдовипадкових чисел на проміжку від *2 до 4* Тоді генерувалося N векторів(кортежів) розміром(довжиною) H.

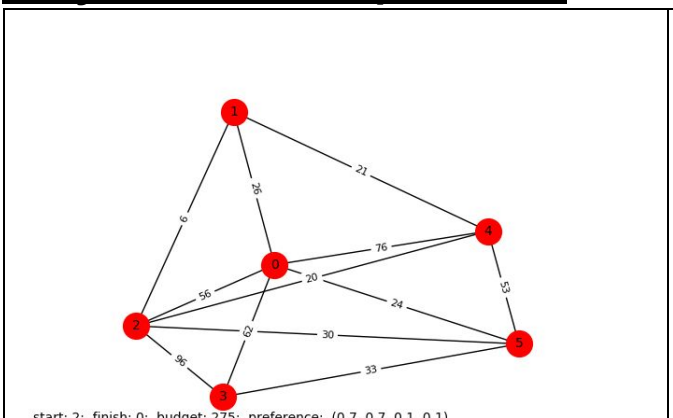
Генерація вектора

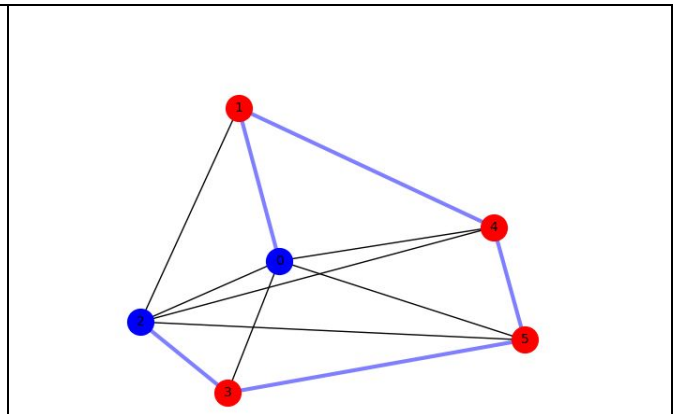
За реальних умов для багатьох POI більшість значень вектора можуть бути рівні 0. Це враховувалося при його генерації. Спочатку формувалася множина значень S від 0.1 до 1.0 із кроком 0.1. До неї додавалось H нулів. Значення фічі генерувалося шляхом псевдовипадкового вибору із згенерованої множини. Якщо отримане значення було відмінним від нуля кількість нулів у множині S подвоювалася. Так само генерувалися наступні значення, з яких формувався вектор.

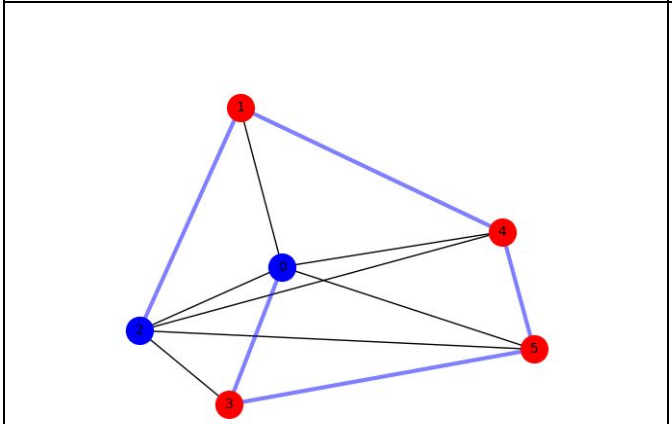
Генерація запиту Q

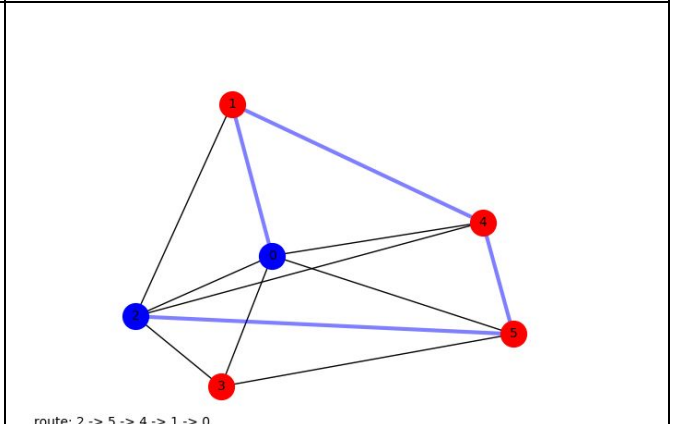
Q = (x, y, b, w, teta, alpha), де x та y – початкова і кінцева точки відповідно(вибирались випадковим чином із множии {0...N-1}); b – бюджет(вибирався випадково але такий що не менший за подвоєне значення ребра найменшої ваги); w – вектор фіч довжиною H, який бажаний для користувача(генерувався шляхом псевдовипадкового вибору чисел); teta – фільтруючий вектор розміром H(вибирався так само як і вектор w); alpha – число необхідне для обрахунку субмодулярної множинної функції(рівне 1).

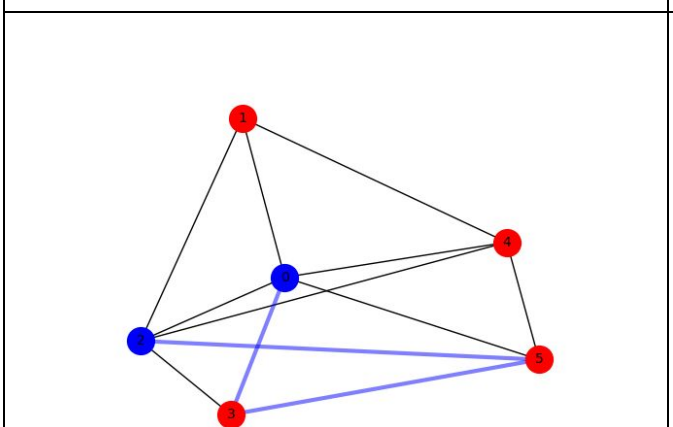
Результати експериментів

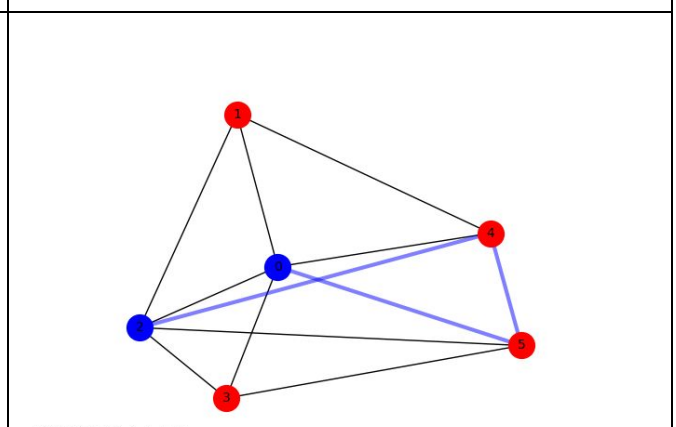






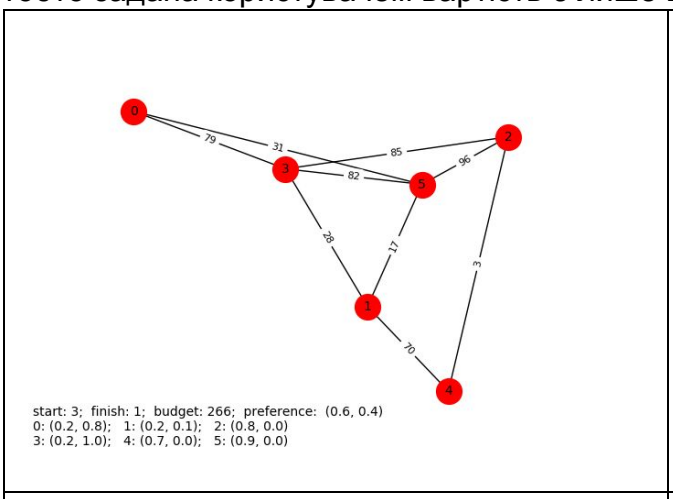


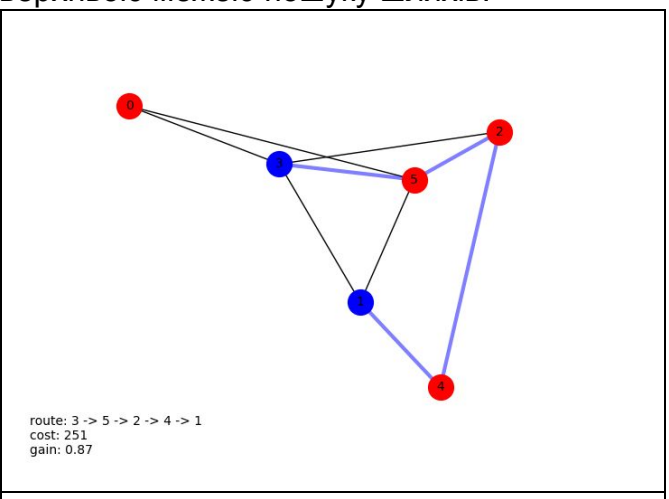


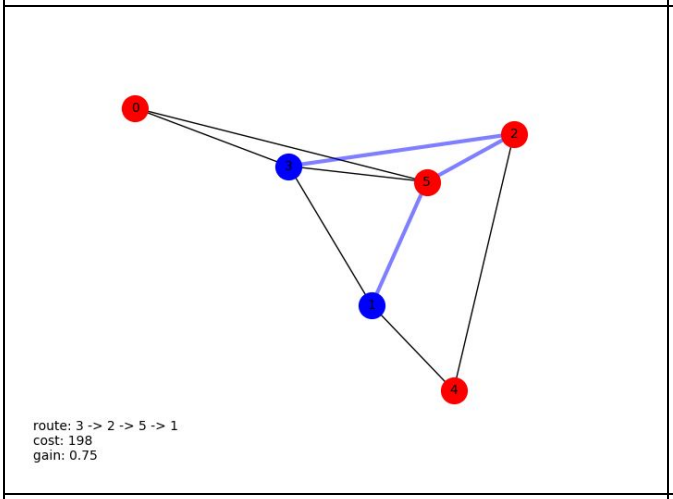


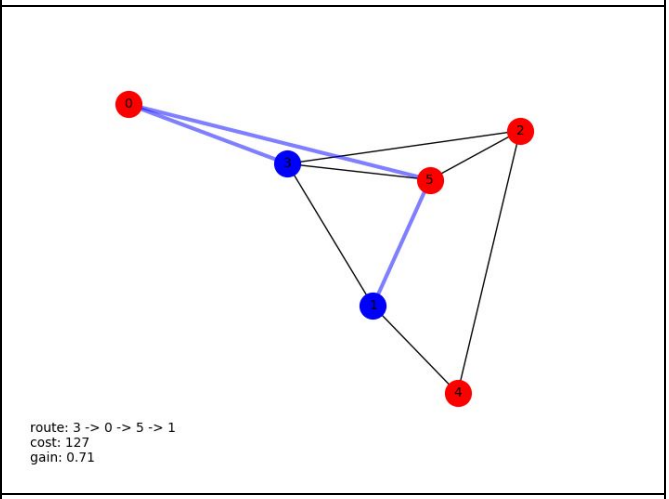
У лівому верхньому куті зображено вхідний граф з його даними та запитом користувача. На інших графах зображено перших 5 найкращих шляхів, які ми отримали за допомогою PACER. Дані шляхи отримані в спадаючому за Gain-ом порядку, тобто найоптимальнішим вважається той, який проходить по вершинах з найбільшим gain-ом та не перевищує дозволену вартість подорожі. Як видно з цих графів,

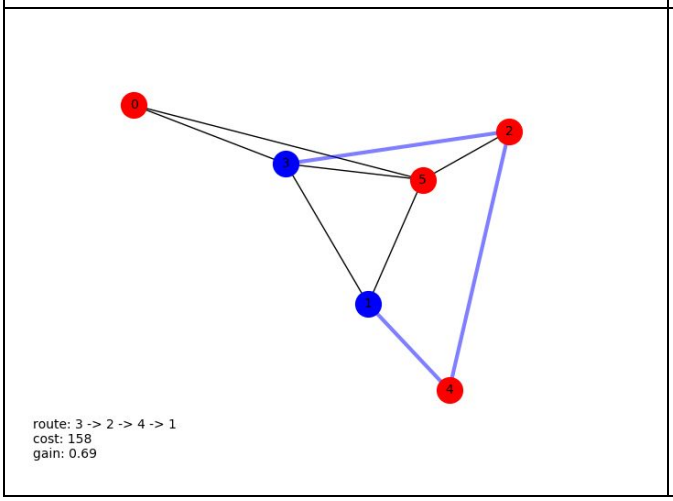
найоптимальніший шлях є не обов’язково найоптимальнішим за вартістю подорожі, тобто задана користувачем вартість є лише верхньою межею пошуку шляхів.

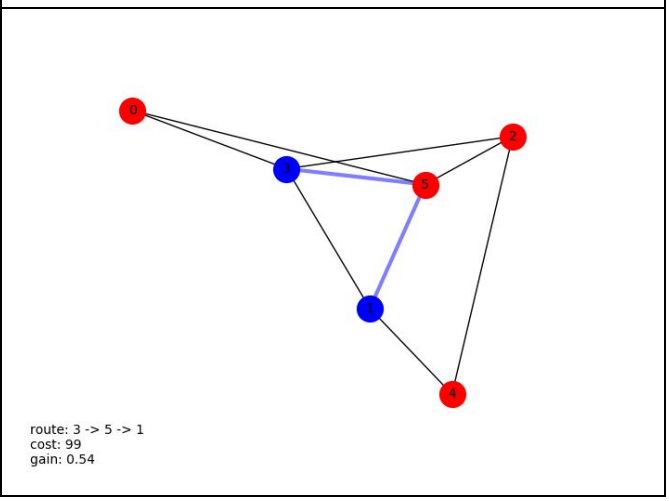












На даному прикладі графу можна побачити, що найоптимальніший шлях є не обов’язково з найменшою вартістю, так само як і у першому прикладі.
Також, усі шляхи пробують обходити граф по вершинах, які мають найбільші значення фіч та відносно найменшу вартість подорожі до них. Таким чином отримуються шляхи, що мають найбільший дохід по фічах та їхня вартість не більші ніж максимальне дозволене значення.

Висновок

В результаті виконання дослідницької роботи ми зрозуміли:

- що таке субмодулярна множинна функція
- що таке евристичні підходи до вирішення задач
- що таке клас складності задачі NP-Hard, і що це дійсно хард(=
- що із написаним псевдокодом алгоритму легше реалізовувати цей алгоритм
- що із UML-діаграмою класів легше реалізовувати проект
- що дослідницьку роботу важливо грамотно структуризувати

В результаті виконання дослідницької роботи ми навчились:

- аналізувати наукову статтю
- розбиратись у роботі невідомого алгоритму
- спільно працювати над одиним проектом на GitHub(зокрема Git Merging)
- створювати ефективну UML-діаграму класів
- “страждати” над відлагодженням програми
- модифіковувати структури даних

В результаті виконання дослідницької роботи для нас залишилось незрозумілим:

- як реалізувати даний алгоритм для кількості вершин графа більшої за 8, через винекнення колізій.
- як знаходити ранк точки необхідний для субмодулярної множинної функції

Посилання на репозиторій в GitHub

https://github.com/borsukvasyl/PACER.git