

TỐI ƯU HÓA ĐIỂM ĐẶT ATM SỬ DỤNG

NỘI SUY KHÔNG GIAN

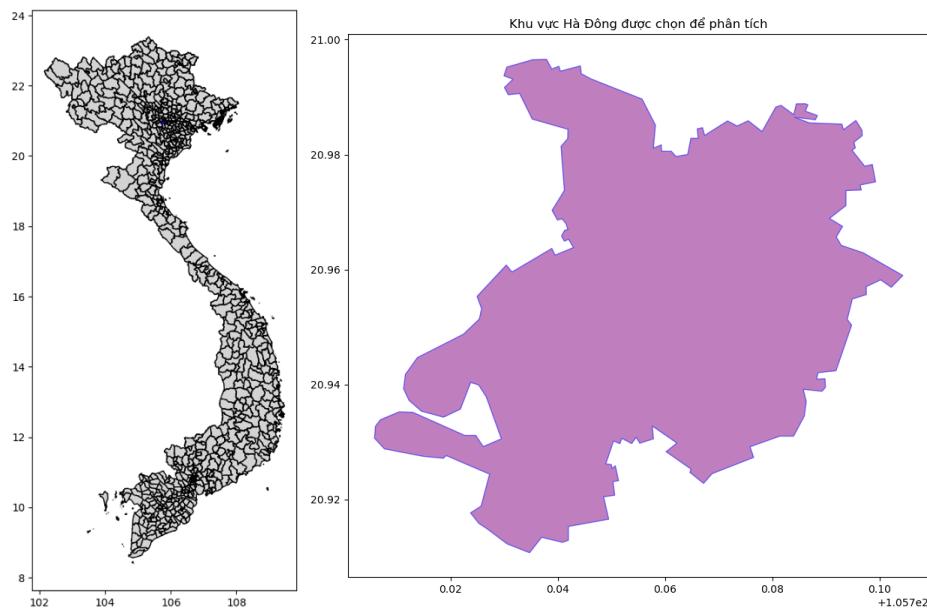
I - Giới thiệu

Mục tiêu của bài toán:

- Đánh giá mức độ bao phủ hiện tại của các điểm ATM hiện có.
- Đề xuất các điểm ATM mới để tăng cường khả năng tiếp cận cho dân cư.
- Tối ưu hóa số lượng điểm ATM có thể mở.

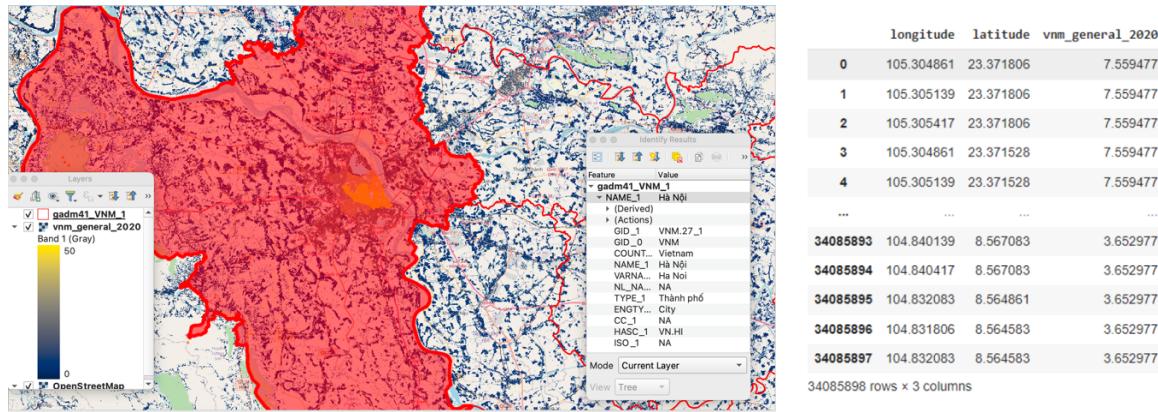
II - Nguồn Dữ liệu

1. *Dữ liệu bản đồ địa giới hành chính Việt Nam (GADM)*: Nguồn dữ liệu bao gồm chi tiết các ranh giới ở Việt Nam chi tiết ở 3 cấp độ thành phố, quận, huyện (thị xã). Nguồn dữ liệu này để giới hạn phạm vi thực hiện của bài toán hiện tại.



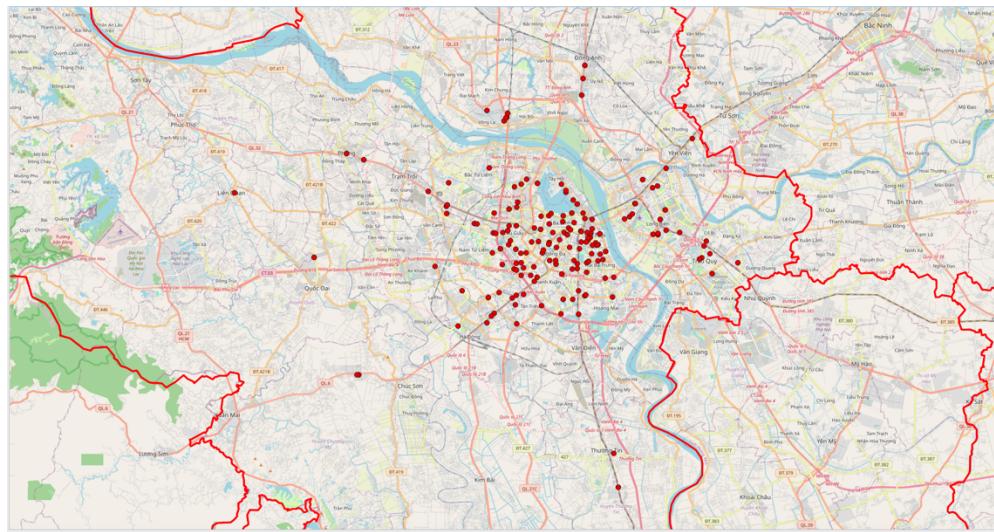
2. *Dữ liệu dân số có mật độ phân giải cao từ Meta và được cấp phép bởi Creative Commons Attribution International (Vietnam: High Resolution Population Density Maps + Demographic Estimates)*

Density Maps + Demographic Estimates): Meta cung cấp dữ liệu dân số dưới dạng bản đồ mật độ dân số chi tiết, giúp phân tích và lập kế hoạch cho các dịch vụ công cộng và các hoạt động phát triển.



Mỗi điểm dữ liệu đại diện cho số lượng dân cư trong 1 ô 30x30m x
(VD: Trong ô đầu tiên có tọa độ tâm là (105.304861, 23.371806) chứa khoảng 7 người)

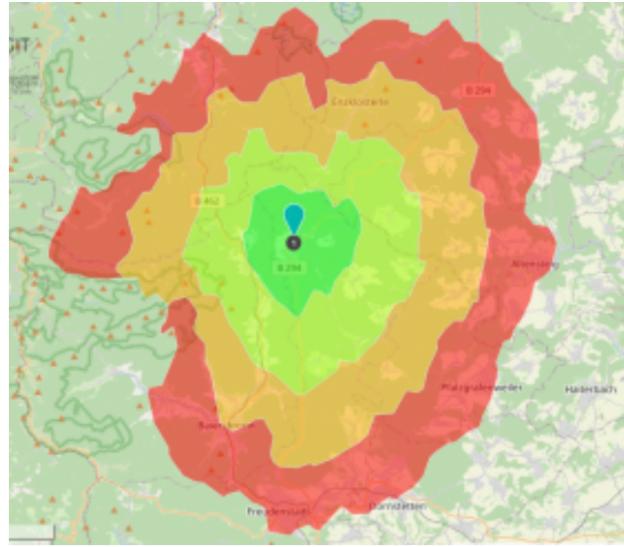
3. *Dữ liệu ATM của Vietcombank trong khu vực Hà Nội.*



III - Phương pháp

1. Đánh giá khả năng tiếp cận điểm bán bằng phân tích Isochrone

Isochrone, hay còn gọi là Vùng tiếp cận, là một khu vực địa lý có thể tiếp cận trong một khoảng thời gian xác định từ một vị trí cụ thể (chẳng hạn như một điểm bán hàng), theo từng phương tiện di chuyển nhất định (như đi bộ, xe đạp, hoặc ô tô).



(Ví dụ các vùng tiếp cận được trực quan lên bản đồ, nguồn từ [OpenRouteService](#))

Phân tích Isochrone cho phép xác định và hình dung các vùng địa lý có thể tiếp cận từ một vị trí ban đầu theo các phương tiện di chuyển trong một khoảng thời gian xác định. Kết quả của phân tích này là các vùng được hiển thị dưới dạng đường viền đa giác trên bản đồ, biểu thị các khu vực có thể tiếp cận.

```
# Isochrone
def get_isochrone(lat, lon, duration, api_key):
    url = "https://api.openrouteservice.org/v2/isochrones/foot-walking"
    headers = {
        'Authorization': api_key
    }
    params = {
        'locations': [(lon, lat)],
        'range': [duration * 60],
        'interval': duration * 60,
        'attributes': ['total_pop']
    }
    response = requests.post(url, headers=headers, json=params)

    # Kiểm tra trạng thái phản hồi
    if response.status_code != 200:
        print("Lỗi API!", response.status_code, response.text)
        return None

    data = response.json()

    # Kiểm tra và chuyển đổi kết quả thành Polygon
    if len(data['features']) > 0 and len(data['features'][0]['geometry']) > 0:
        coordinates = data['features'][0]['geometry']['coordinates'][0]
        polygon_geom = Polygon(coordinates)
        return polygon_geom
    else:
        print("Không thể tạo Isochrone!")
        return None

# Khởi tạo danh sách để lưu isochrones cho các điểm ATM hiện có
isochrone_existing_atm = []

# Vòng lặp qua các điểm ATM hiện có trong 'filtered_data' và tính toán Isochrone
for atm, row in filtered_data.iterrows():
    lat, lon = row['geometry'].y, row['geometry'].x

    # In ra thông báo độ hiện tại đang xử lý
    print(f"Dang tao isochrone cho tọa độ ATM hiện có: (lat: {lat}, lon: {lon})")

    # Gọi hàm tính Isochrone
    isochrone_polygon = get_isochrone(lat, lon, duration=15, api_key=api_key)

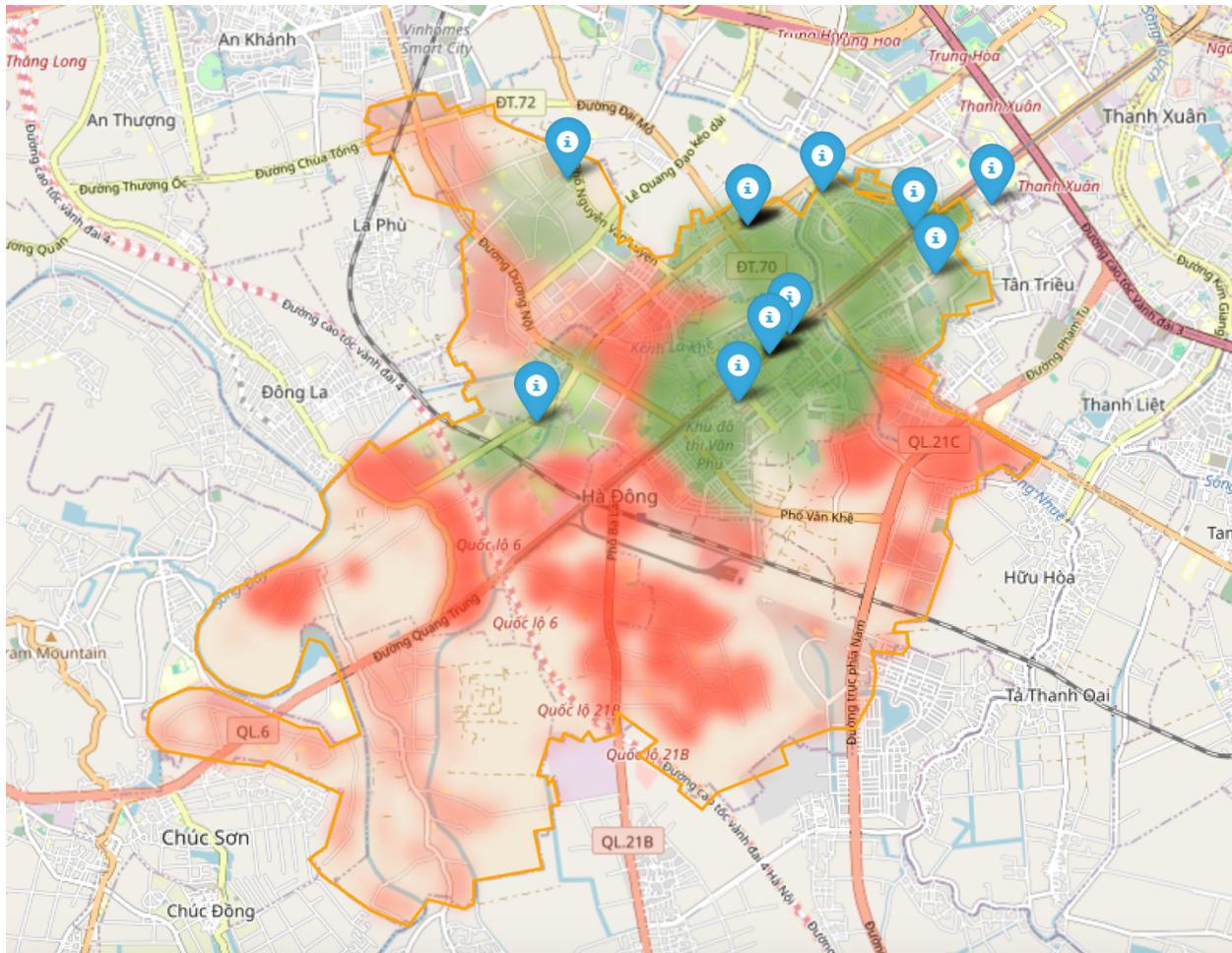
    # Thêm thời gian chờ giữa các yêu cầu để tránh giới hạn API
    time.sleep(20)

    # Kiểm tra kết quả và thêm vào danh sách nếu thành công
    if isochrone_polygon:
        isochrone_existing_atm.append({
            'geometry': isochrone_polygon,
            'id': atm
        })
    print("Thành công")

# Chuyển danh sách Isochrone thành GeoDataFrame
isochrone_existing_atm_gdf = pd.DataFrame(isochrone_existing_atm, crs="EPSG:4326")

# Kiểm tra dữ liệu Isochrone của các ATM hiện có
print(isochrone_existing_atm_gdf)
```

Áp dụng tính toán khả năng tiếp cận cho các vị trí ATM hiện tại (Tính 15 phút từ điểm hiện tại đang xét)



Kết quả cho thấy khả năng tiếp cận hiện tại là 41.38%

Population with Access: 41.38%

2. Xây dựng bài toán

Các tiêu chí đặt ra cho bài toán:

- Lựa chọn các vị trí ATM sao cho có thể tiếp cận được số lượng khách hàng tiềm năng lớn nhất.
- Đảm bảo rằng thời gian di chuyển từ khu vực sinh sống của khách hàng tới ATM không vượt quá 15 phút.
- Xem xét các khu vực mà người dân có thể tiếp cận điểm bán bằng cách đi bộ.

Mô hình hóa bài toán:

I — tập hợp các điểm dữ liệu dân số

J — tập hợp các địa điểm ATM tiềm năng

J_i — tập hợp các địa điểm ATM tiềm năng trong khả năng truy cập của điểm dữ liệu $i \in I$

(Lưu ý: $J_i \subseteq J$).

v_i — số người trong điểm dữ liệu $i \in I$.

Mô hình này xác định các biến z_i cho mỗi điểm dữ liệu $i \in I$ để biết liệu khu vực dân cư đó có thể được phục vụ bởi một điểm bán, còn biến x_j cho các địa điểm ATM tiềm năng để biết ATM đó liệu có được chọn để mở mới hay không.

$$\begin{aligned} & \max \quad \sum_{i \in I} v_i z_i \\ \text{subject to: } & z_i \leq \sum_{j \in J_i} x_j \quad \forall i \in I \\ & \sum_{j \in J} x_j \leq p \\ & x_j \in \{0, 1\} \quad \forall j \in J \\ & z_i \in \{0, 1\} \quad \forall i \in I \end{aligned}$$

Mô hình **maximal covering** như trong [bài viết của Church và ReVelle](#)

Dòng đầu tiên nêu mục tiêu **tối đa hóa tổng số dân cư được phục vụ**.

Dòng thứ hai (sau _subject to:_) liệt kê ràng buộc đầu tiên: **mỗi điểm dữ liệu chỉ được phục vụ nếu có ít nhất một ATM trong tầm tay mở cửa**.

Dòng thứ 3 là ràng buộc **số điểm ATM được chọn mở mới không được vượt quá p** (là số ATM tối đa được mở, cái này sẽ quy định khi chạy thuật toán)

3. Thiết lập bài toán

Bước 1: Nhập các thư viện cần thiết, các bộ dữ liệu để phục vụ bài toán.

```
import pandas as pd
import geopandas as gpd
import requests
from shapely.geometry import Polygon
from shapely.geometry import Point
from shapely.ops import unary_union
import matplotlib.pyplot as plt
import numpy as np
from geopandas.tools import sjoin
import time
from gethighs import HIGHS
import pyomo.environ as pyo
```

Python

Dữ liệu về ranh giới địa chính:

Ranh giới địa chính

```
# Đường dẫn đến tệp Shapefile (ví dụ như "gadm41_VNM_0.shp" hoặc tệp tương ứng bạn muốn phân tích)
file_path = "/Users/fuongfotfet/Desktop/vcb/gadm41_VNM_shp/gadm41_VNM_2.shp" # thay "path_to_your_file" bằng đường dẫn thực tế của tệp

# Đọc dữ liệu từ Shapefile
gdf = gpd.read_file(file_path)

# Lọc khu vực hành chính muốn phân tích, ví dụ: "Hà Đông - Hà Nội"
selected_district = "Hà Đông" # Tên khu vực bạn muốn phân tích
selected_gdf = gdf[gdf['NAME_2'] == selected_district]

# Kiểm tra dữ liệu đã lọc
print(selected_gdf)
```

Python

```
GID_2 GID_0 COUNTRY GID_1 NAME_1 NL_NAME_1 NAME_2 \
238 VNM.27.11_1 VNM Vietnam VNM.27_1 Hà Nội NA Hà Đông

VARNAME_2 NL_NAME_2 TYPE_2 ENGTTYPE_2 CC_2 HASC_2 \
238 Hà Đông NA Quận Urban District NA VN.ND.HH

geometry
238 POLYGON ((105.71912 20.9277, 105.71861 20.9272...
```



Dữ liệu dân số quận Hà Đông

```

# Đọc dữ liệu dân số Hà Đông
population_data = pd.read_csv("/Users/fuongfotfet/Downloads/vnm_general_2020.csv")

# Đổi tên các cột cho phù hợp (giả sử có các cột 'longitude', 'latitude', và 'population')
population_data.columns = ['longitude', 'latitude', 'population']

# Chuyển đổi thành GeoDataFrame
population_gdf = gpd.GeoDataFrame(
    population_data,
    geometry=gpd.points_from_xy(population_data.longitude, population_data.latitude),
    crs="EPSG:4326"
)

# Tính toán dân số trong khu vực Hà Đông
population_in_hadong = sjoin(population_gdf, selected_gdf, how="inner", predicate="within")

# Kiểm tra dữ liệu sau khi xử lý
print(population_in_hadong.head())

```

[5]

```

...      longitude   latitude  population      geometry \
4520065  105.737639  20.996250  14.904768  POINT (105.73764 20.99625)
4521761  105.733194  20.995972  7.520281  POINT (105.73319 20.99597)
4521762  105.735139  20.995972  7.520281  POINT (105.73514 20.99597)
4521763  105.736528  20.995972  7.520281  POINT (105.73653 20.99597)
4521764  105.737083  20.995972  14.904768  POINT (105.73708 20.99597)

      index_right      GID_2 GID_0 COUNTRY      GID_1 NAME_1 NL_NAME_1 \
4520065          238 VNM.27.11_1 VNM Vietnam VNM.27_1 Hà Nội       NA
4521761          238 VNM.27.11_1 VNM Vietnam VNM.27_1 Hà Nội       NA
4521762          238 VNM.27.11_1 VNM Vietnam VNM.27_1 Hà Nội       NA
4521763          238 VNM.27.11_1 VNM Vietnam VNM.27_1 Hà Nội       NA
4521764          238 VNM.27.11_1 VNM Vietnam VNM.27_1 Hà Nội       NA

      NAME_2 VARNAME_2 NL_NAME_2 TYPE_2      ENGTTYPE_2 CC_2      HASC_2
4520065  Hà Đông  Ha Dong        NA  Quận  Urban District     NA VN.ND.HH
4521761  Hà Đông  Ha Dong        NA  Quận  Urban District     NA VN.ND.HH
4521762  Hà Đông  Ha Dong        NA  Quận  Urban District     NA VN.ND.HH
4521763  Hà Đông  Ha Dong        NA  Quận  Urban District     NA VN.ND.HH
4521764  Hà Đông  Ha Dong        NA  Quận  Urban District     NA VN.ND.HH

```

Python

Dữ liệu các điểm ATM trong quận Hà Đông

ATM_ID	NAME	ADR_01 \
156 10800336	PHENIKAA	TRUONG DAI HOC PHENIKAA, YEN
190 10800397	TOA NHA TSQ MO LAO	CT1 KDT MO LAO
193 10800400	BHXH TP HA NOI	15 CAU DO
236 10800511	KCN PHU NGHIA	KHU CN PHU NGHIA
237 10800512	KCN PHU NGHIA	KHU CN PHU NGHIA
238 10800513	KCN PHU NGHIA	KHU CN PHU NGHIA
239 10800514	10A QUANG TRUNG	10 QUANG TRUNG, P QUANG TRUNG
240 10800515	10A QUANG TRUNG	10 QUANG TRUNG, P QUANG TRUNG
241 10800516	PGD QUANG TRUNG	PGD QUANG TRUNG SO 80 QUANG TR
242 10800517	PGD QUANG TRUNG	PGD QUANG TRUNG SO 80 QUANG TR
244 10800519	DH CNGTVT	DAI HOC CNGTVT, SO 54 PHO TRIE
246 10800522	PGD DAN PHUONG	TTTM TUAN QUYNH THI TRAN PHUNG
247 10800523	TRU SO CN TAY HA NOI	LO HH03 DUONG TO HUU, P.VAN PH
249 10800525	TRU SO CN TAY HA NOI	LO HH03 DUONG TO HUU, P.VAN PH
251 10800527	TRU SO CN TAY HA NOI	LO HH03 DUONG TO HUU, P.VAN PH
257 10800544	SIEU THI COOP-MART	KM 10 DUONG NGUYEN TRAI
258 10800545	PGD VAN QUAN	BT8 LO 44 KHU DO THI VAN QUAN
277 10800609	AEON HA DONG	TANG 1 TTTM AEON HA DONG,TDP H
359 10800530	TSCN TAY HA NOI	LO HH03 TO HUU
361 10800531	PGD QUANG TRUNG VCB THN	80 QUANG TRUNG
	ADR_02	CITY LATITUDE LONGITUDE \
156 HA DONG		HA NOI 20.962822 105.748815
190 HA DONG		HA NOI 20.986552 105.780302
...		
258	PGD	POINT (105.79284 20.97812)
277	Nan	POINT (105.75217 20.98797)
359	TSCN	POINT (105.77214 20.98325)
361	PGD	POINT (105.77445 20.9699)

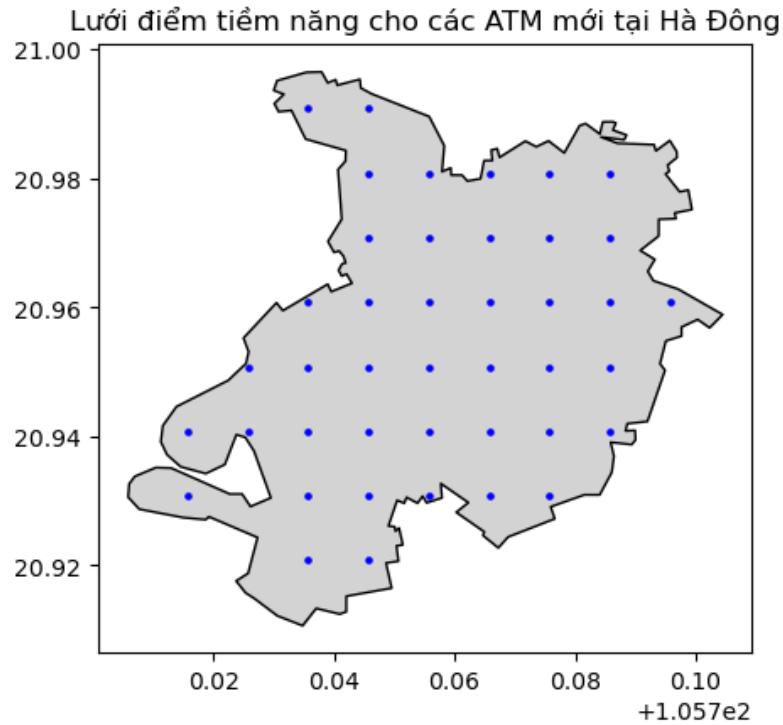
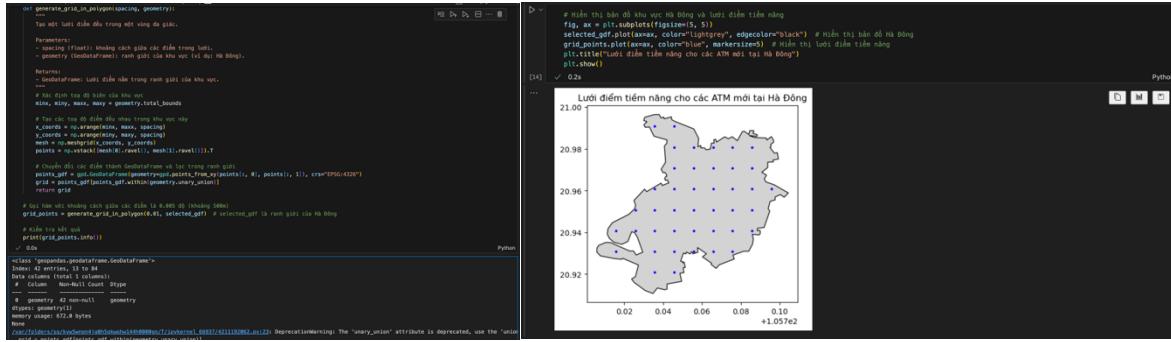
Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings...](#)

Bước 2: Tạo lưới các điểm có tiềm năng đại diện

Trong trường hợp không có đề xuất địa điểm cụ thể cho việc phân tích các vị trí tiềm năng từ các nguồn chính thức, các điểm có thể được ước tính gần đúng thông qua một lưới đại diện trong khu vực mục tiêu.

Một mạng lưới như vậy cung cấp một khuôn khổ khởi đầu để xem xét nơi thành lập các điểm ATM mới nhằm tối đa hóa khả năng tiếp cận cho nhóm dân số chưa được phục vụ đầy đủ.

Hàm này tạo ra một loạt các điểm cách đều nhau - được xác định bởi tham số khoảng cách, tạo ra 42 điểm tiềm năng ở Hà Đông.



Bước 3: Tính toán khả năng tiếp cận cho toàn bộ điểm hiện có và tiềm năng

```

# Isochrone
def get_isochrone(lat, lon, duration, api_key):
    url = "https://api.openrouteservice.org/v2/isochrones/foot-walking"
    headers = {
        'Authorization': api_key
    }
    params = {
        'locations': [[lon, lat]],
        'range': [duration * 60],
        'interval': duration * 60,
        'attributes': ['total_pop']
    }

    response = requests.post(url, headers=headers, json=params)

    # Kiểm tra trạng thái phản hồi
    if response.status_code != 200:
        print("Lỗi API:", response.status_code, response.text)
        return None

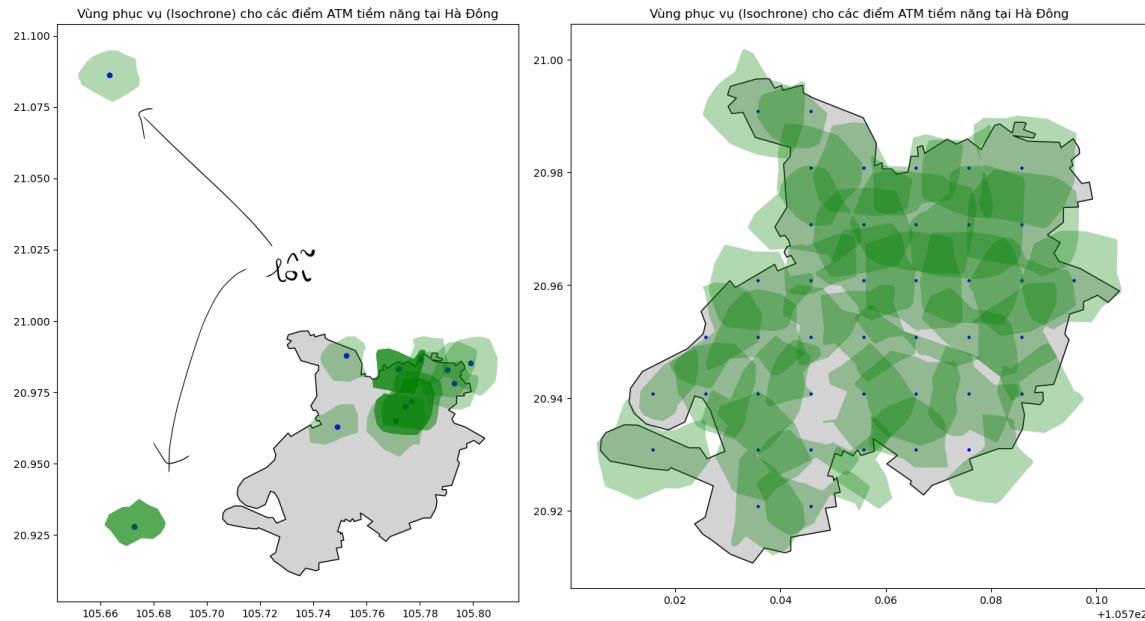
    data = response.json()

    # Kiểm tra và chuyển đổi kết quả thành Polygon
    if 'features' in data and len(data['features']) > 0:
        coordinates = data['features'][0]['geometry']['coordinates'][0]
        polygon_geom = Polygon(coordinates)
        return polygon_geom
    else:
        print("Không thể tạo Isochrone.")
        return None

# Ví dụ sử dụng hàm cho một điểm bán hàng
lat, lon = 21.986552, 106.780302 # Tọa độ của điểm bán hàng
api_key = '5b3ce597851110001cf62487458313b6e8042c8042c9c33' # Thay thế bằng API key của bạn
isochrone = get_isochrone(lat, lon, duration=15, api_key=api_key)
isochrone

```

Python



Khả năng tiếp cận tối đa của các điểm ATM tiềm năng: 96.69%

Điều này cho thấy khả năng bao phủ tối đa của quận Hà Đông có thể lên tới 96.69%

Bước 4: Thiết lập mô hình tối ưu hoá

Các hàm tối ưu của bài toán sẽ được đặt vào môi trường Pyomo, sau đó sẽ dùng bộ giải HiGHS để tìm các phương án tối ưu cho bài toán.

Mô hình bài toán:

```

def model_max_covering(w, I, J, JI, p, J_existing):
    # Đảm bảo rằng tất cả các điểm hiện có đều nằm trong tập hợp J
    assert set(J_existing).issubset(set(J))

    # Khởi tạo mô hình
    m = pyo.ConcreteModel("MaxCovering")

    # Khai báo tham số
    m.p = pyo.Param(mutable=True, within=pyo.Integers, default=p)
    m.I = pyo.Set(initialize=I)
    m.J = pyo.Set(initialize=J)
    m.Jfixed = pyo.Set(initialize=list(J_existing))
    m.nof_fixed = pyo.Param(mutable=False, within=pyo.Integers, default=len(J_existing))

    # Khai báo hàm trọng số dân số
    @m.Param(m.I, within=pyo.NonNegativeReals)
    def w(m, i):
        return w[i]

    # Khai báo hàm ánh xạ từ I đến J
    @m.Param(m.I, within=pyo.Any)
    def JI(m, i):
        return JI.get(i, [])

    # Khai báo biến quyết định
    m.x = pyo.Var(m.J, within=pyo.Binary) # Quyết định có mở ATM ở vị trí j hay không
    m.z = pyo.Var(m.I, within=pyo.Binary) # Quyết định khu vực dân cư i có được phục vụ hay không

    # Hàm mục tiêu: tối đa hóa dân số được phục vụ
    @m.Objective(sense=pyo.maximize)
    def covering(m):
        return pyo.quicksum(m.w[i] * m.z[i] for i in m.I)

    # Ràng buộc: Khu vực dân cư chỉ được phục vụ nếu có ít nhất một ATM trong phạm vi
    @m.Constraint(m.I)
    def serve_if_reachable_and_open(m, i):
        return m.z[i] <= pyo.quicksum(m.x[j] for j in m.JI[i])

    # Ràng buộc: Số lượng ATM mở không vượt quá `p` và `nof_fixed`
    @m.Constraint()
    def budget(m):
        return pyo.quicksum(m.x[j] for j in m.J) <= m.nof_fixed + m.p

    # Ràng buộc: Các ATM hiện có phải được mở
    @m.Constraint(m.Jfixed)
    def fix_open(m, j):
        return m.x[j] == 1

    return m

```

Hàm `model_max_covering` xác định mô hình tối ưu hóa toán học bằng Pyomo. Cần chuẩn bị các tham số:

- `w`: Một từ điển chứa số lượng dân số cho mỗi ID điểm dữ liệu
- `I`: Một bộ ID các điểm dữ liệu
- `J`: Một tập hợp ID vị trí ATM tiềm năng
- `JI`: Từ điển ánh xạ ID hộ gia đình tới bộ ID ATM tiềm năng có thể phục vụ chúng
- `p`: Số lượng ATM tối đa được mở

Mô hình thiết lập các biến quyết định `x` (nhị phân, có mở ATM ở vị trí j) và `z` (nhị phân, có phục vụ điểm dữ liệu i hay không). Mục tiêu tối đa hóa tổng dân số được tiếp cận, tuân theo các ràng buộc nhằm đảm bảo các điểm dữ liệu chỉ được đánh dấu là được phục vụ nếu mở một ATM có thể tiếp cận được. Tổng số ATM mở không vượt ngang sách p .

Chuẩn bị dữ liệu đầu vào:

```

w = population_in_hadong.set_index('ID')['population'].to_dict()

# Các điểm ATM hiện có trong khu vực Hà Đông
J_existing = set(filtered_data.index)
# Các điểm tiềm năng trong lưới điểm đã tạo
J_potential = set(grid_points.index)

J = sorted(J_existing | J_potential) # Tất cả các điểm ATM
I = sorted(set(population_in_hadong['ID'])) # Tất cả các điểm dân cư

# Tạo ánh xạ từ ID cửa hàng tới các hộ gia đình có thể tiếp cận
IJ_existing = {
    atm_id: population_in_hadong[population_in_hadong['geometry'].within(isochrone_existing_atm_gdf.loc[atm_id, 'geometry'])].index.tolist()
    for atm_id in J_existing
}

IJ_potential = {
    atm_id: population_in_hadong[population_in_hadong['geometry'].within(isochrone_gdf.loc[atm_id, 'geometry'])].index.tolist()
    for atm_id in J_potential
}

def reverse_mapping(mapping):
    from collections import defaultdict
    aux = defaultdict(set)
    for x, Y in mapping.items():
        for y in Y:
            aux[y].add(x)
    return {y: sorted(aux[y]) for y in sorted(aux.keys())}

JI = reverse_mapping(IJ)

```

- J_existing : Bộ ID cửa hàng hiện có.
- J_potential : Tập hợp ID vị trí cửa hàng mới tiềm năng (trong lưới điểm)
- I : Tập hợp tất cả các ID của các điểm dữ liệu dân cư.
- IJ_existing : Từ điển ánh xạ ID ATM hiện có tới các điểm dữ liệu dân cư được phục vụ.
- IJ_potential : Từ điển ánh xạ ID ATM tiềm năng tới các điểm dữ liệu dân cư được phục vụ.
- JI : Từ điển ngược lại từ hai từ điển trước, ánh xạ mỗi điểm dữ liệu tới các tập hợp ATM tiềm năng

Bước 5: Giải mô hình và tìm các địa điểm ATM tiềm năng mới

```

model = model_max_covering(w, I, J, JI, 3, J_existing)
model

<pyomo.core.base.PyomoModel.ConcreteModel at 0x31a5ab520>

solver = HiGHS(time_limit=10, mip_heuristic_effort=0.2, mip_detect_symmetry="on")
solver.solve(model)

```

Giải mô hình với tham số $p = 3$ (tức mở tối đa 3 điểm ATM)

```

# Danh sách các điểm ATM được chọn
selected_ATMs = [j for j in J if model.x[j].value == 1]

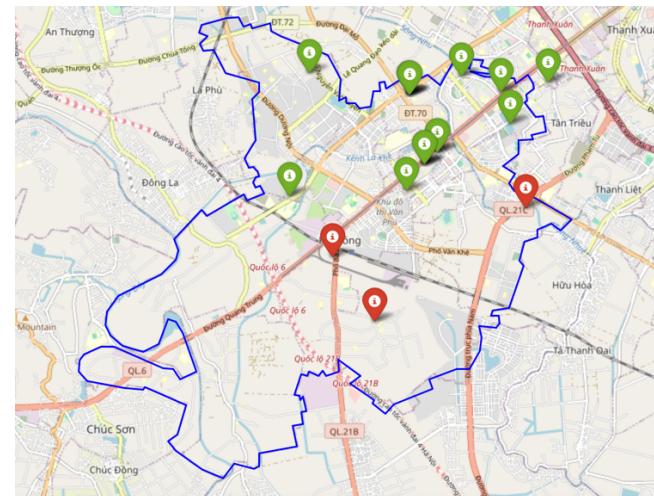
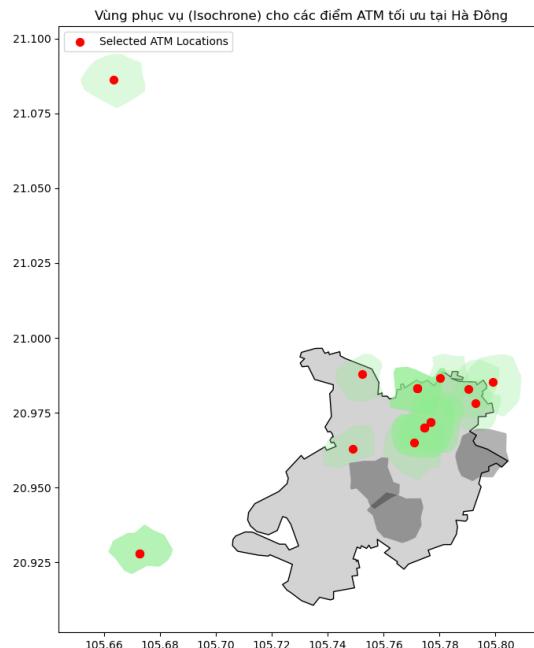
# Danh sách các khu vực (variable) i: Any
served_areas = [i for i in I if model.z[i].value == 1]

print("Các điểm ATM tối ưu để mở:", selected_ATMs)
print("Các khu vực dân cư được phục vụ:", served_areas)

```

Các điểm ATM tối ưu để mở: [36, 45, 59, 156, 190, 193, 236, 237, 238, 239, 240, 241, 242, 244, 246, 247, 249, 251, 257, 258, 277, 359, 361]
Các khu vực dân cư được phục vụ: [4554261, 4554262, 4556023, 4556024, 4556025, 4556026, 4556027, 4556028, 4557728, 4557729, 4557730, 4557731, 4557732, 4557733,

Trích xuất các điểm ATM đã được mô hình chọn, trong đó có 3 điểm mới là 36, 45, 59 đã
các điểm tiềm năng.



(Các vùng xám là vùng bao phủ mới)

Khả năng bao phủ hiện tại của các điểm ATM: 61.93%

Khả năng bao phủ đã được tăng từ 41% lên 61.93%.

IV - Phương pháp đánh giá

Tỷ lệ phần trăm dân số có quyền truy cập được tính toán và ngân sách tương ứng được thêm vào result_list.

```
✓ def process_value(each_val):
    # Khởi tạo lại mô hình mỗi lần gọi hàm để đảm bảo tính độc lập
    model = model_max_covering(w, I, J, JI, each_val, J_existing)
    solver.solve(model)

    # Lấy danh sách các ID của ATM đã được chọn mở (cả mới và cũ)
    opened_ids = [j for j in J if model.x[j].value == 1]

    # Lọc các isochrones của ATM mới được chọn mở
    selected_isochrones_new = isochrone_gdf[isochrone_gdf.index.isin(opened_ids)]

    # Lọc các isochrones của ATM cũ đã mở
    selected_isochrones_existing = isochrone_existing_atm_gdf[isochrone_existing_atm_gdf.index.isin(J_existing)]

    # Kết hợp các isochrones của cả ATM mới và ATM cũ
    combined_area = unary_union(list(selected_isochrones_new.geometry) + list(selected_isochrones_existing.geometry))

    # Tìm các khu vực dân cư nằm trong vùng phủ tổng hợp này
    pop_with_access = population_in_hadong[population_in_hadong.within(combined_area)]
    pop_without_access = population_in_hadong[~population_in_hadong.within(combined_area)]

    # Tính phần trăm dân số được bao phủ
    total_population = population_in_hadong['population'].sum()
    pop_percentage = round(pop_with_access['population'].sum() * 100 / total_population, 2)

    return each_val, len(opened_ids), pop_percentage

# Danh sách các giá trị để xử lý
values_to_process = range(len(grid_points))

# Chạy tuần tự để kiểm tra lỗi
result_list = [process_value(val) for val in tqdm(values_to_process)]
✓ 5m 49.9s
```

```
print(result_list)
✓ 0.0s
[(0, 20, 41.38), (1, 21, 50.3), (2, 22, 56.2), (3, 23, 61.93), (4, 24, 66.6), (5, 25, 71.14), (6, 26, 74.11), (7, 27, 76.37), (8, 28, 78.5),
```

Kết quả của lần lượt các số lượng ATM tối đa được mở với số lượng ban đầu có 20 ATM (VD: (0, 20, 41.48) tức là 0 ATM được mở thêm, hiện tại có 20 ATM và mức độ bao phủ 41.48%)

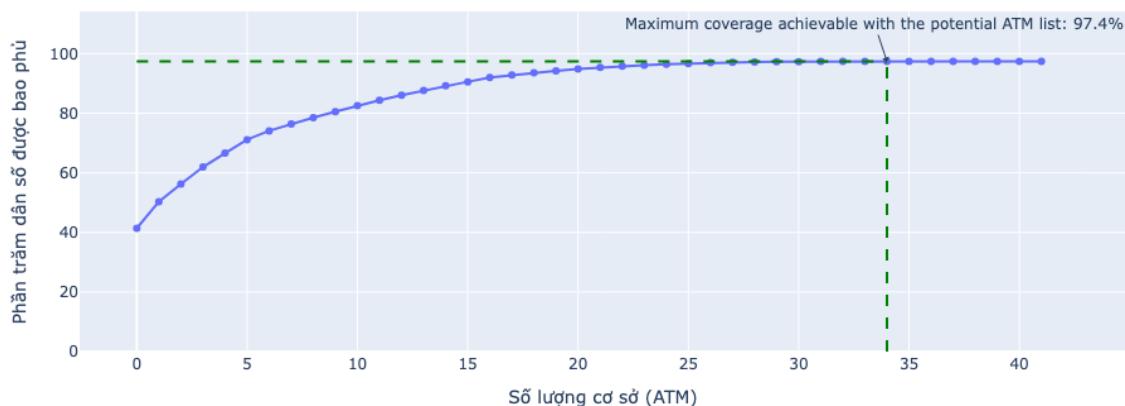
Trục quan hóa đường cong Pareto đề cập đến việc vẽ đồ thị kết quả từ mô hình tối ưu hóa chạy với các ngân sách khác nhau (số lượng cửa hàng sẽ mở) trên biểu đồ 2D. Điều này cho phép hình dung sự cân bằng giữa số lượng cơ sở được mở và mức độ bao phủ dân số tương ứng. Cụ thể, trục x biểu thị số lượng ATM được mở (ngân sách), trong khi trục y biểu thị tỷ lệ dân số được tiếp cận các ATM. Khi ngân sách tăng (và do đó, số lượng cửa hàng mở, độ bao phủ cho người dân có xu hướng được cải thiện).

Tuy nhiên, tốc độ cải thiện thường giảm dần, dẫn đến đường cong trở nên phẳng hơn khi có thêm nhiều cửa hàng vượt quá một điểm nhất định. Đường cong này được gọi là đường cong Pareto hoặc biên giới Pareto, vì nó đại diện cho tập hợp các giải pháp không bị chi phối.

Hình dung đường cong Pareto rất có giá trị trong bối cảnh này vì nó giúp những người ra quyết định hiểu được sự đánh đổi liên quan và đưa ra những lựa chọn sáng suốt. Ví dụ:

1. Nếu đường cong ban đầu có độ dốc lớn, việc bổ sung thêm một vài ATM có thể cải thiện đáng kể phạm vi bao phủ dân số, khiến các khoản đầu tư như vậy có hiệu quả cao.
2. Nếu đường cong phẳng đi nhanh chóng, việc mở thêm ATM ngoài một điểm nhất định có thể không làm tăng đáng kể phạm vi bao phủ, cho thấy lợi nhuận giảm dần.
3. Đường cong có thể tiết lộ "đầu gối" hoặc các điểm mà sự đánh đổi trở nên kém thuận lợi hơn, giúp xác định các điểm cân bằng tiềm năng cho hai mục tiêu.

Đường cong Pareto: Mối quan hệ giữa số lượng cơ sở (ATM) và độ bao phủ dân số



V - Định hướng tiếp theo

Trong các giai đoạn tiếp theo, mô hình sẽ cố gắng tích hợp thêm các ràng buộc liên quan đến bài toán tối ưu ví dụ như:

- Tối thiểu số ATM đối thủ
- Tối đa hóa bao phủ các vùng chứa nhiều các Point of Interests (POIs)
- Mật độ giao thông
- ...

References:

1. **Nguyễn, Văn Dương.** “Tối ưu hóa lựa chọn điểm bán hàng sử dụng nội suy không gian: Phần 2 - Phân tích thăm dò dữ liệu (EDA).”
2. **Krishnan, Parvathy.** “An Open Data-Driven Approach to Optimising Healthcare Facility Locations Using Python.” *Towards Data Science*, June 12, 2024.
<https://towardsdatascience.com/an-open-data-driven-approach-to-optimising-healthcare-facility-locations-using-python-397b3ce38185>.
3. **Janosov, Milan.** “Public Transport Accessibility in Python.” *Towards Data Science*, April 28, 2024. <https://towardsdatascience.com/public-transport-accessibility-in-python-dbdeee99f36f>.