Tiêu đề bài viết

“Tối ưu hóa lựa chọn điểm bán hàng sử dụng nội suy không gian: Phần 3 - Lựa chọn điểm bán bằng phương pháp tối ưu toán học”

Tổ chức các địa điểm bán hàng hiệu quả bằng cách sử dụng các kỹ thuật phân tích không gian

I - Giới thiệu

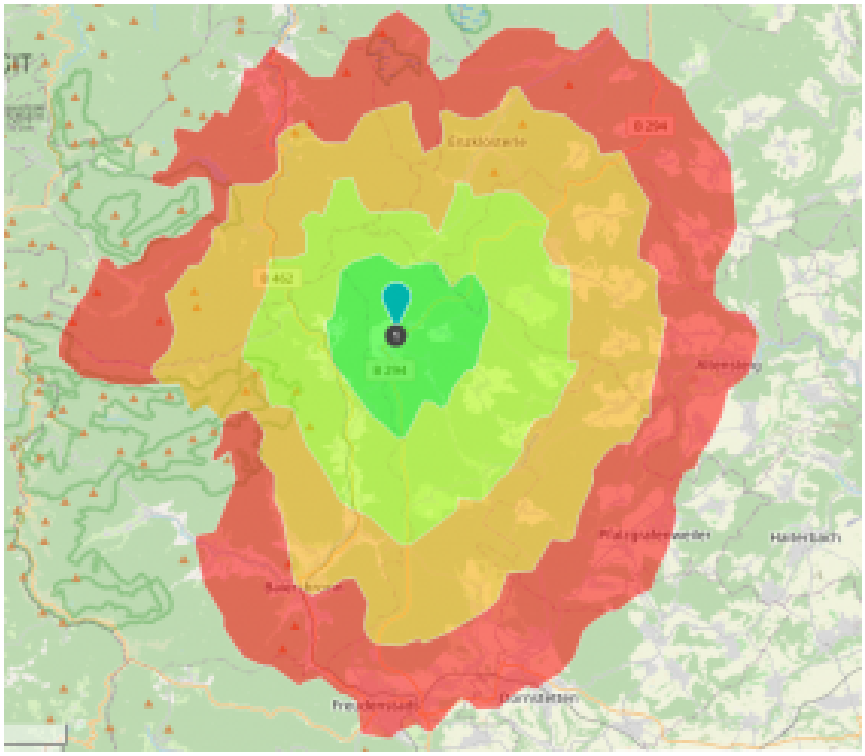
Trong loạt bài viết này, chúng tôi đã đề cập đến các khía cạnh khác nhau của việc áp dụng phân tích không gian để tối ưu hóa lựa chọn điểm bán hàng. Phần 1 đã nêu bật những lợi ích của việc sử dụng phân tích không gian trong việc xác định vị trí điểm bán hàng, giúp doanh nghiệp tiếp cận được nhiều khách hàng tiềm năng hơn. Phần 2 đã đi sâu vào việc xử lý và phân tích dữ liệu không gian, cung cấp cái nhìn chi tiết về các yếu tố dữ liệu đầu vào quan trọng.

Trong bài viết này, chúng tôi sẽ tiếp tục phân tích kỹ thuật và trình bày phương pháp sử dụng phân tích Isochrone – một công cụ mạnh mẽ giúp đánh giá khả năng tiếp cận của các điểm bán hàng dựa trên thời gian di chuyển của khách hàng tiềm năng. Chúng tôi sẽ minh họa cách phương pháp này có thể được áp dụng để lựa chọn các điểm bán hàng tối ưu trong lĩnh vực kinh doanh mặt hàng Mẹ & Bé, khu vực thực nghiệp được giới hạn ở khu vực Hà Đông, Hà Nội.

II - Đánh giá khả năng tiếp cận điểm bán bằng phân tích Isochrone

Isochrone hay còn gọi Vùng tiếp cận được hiểu là gì?

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.

Trong thí nghiệm này, chúng tôi sử dụng API Isochrone từ eKMap API để xác định vùng tiếp cận cho các cơ sở siêu thị Mẹ & Bé tại khu vực Hà Đông, Hà Nội. Hàm **get_isochrone_osm** được xây dựng để tính toán đồng thời các vùng tiếp cận cho các cơ sở này. Isochrone được hiển thị dưới dạng đa giác đại diện cho các khu vực có thể tiếp cận trong **60 phút** di chuyển bằng cách đi bộ từ mỗi cơ sở. Các đa giác này sau đó được sử dụng để xác định số lượng người dân có thể truy cập vào từng cơ sở:

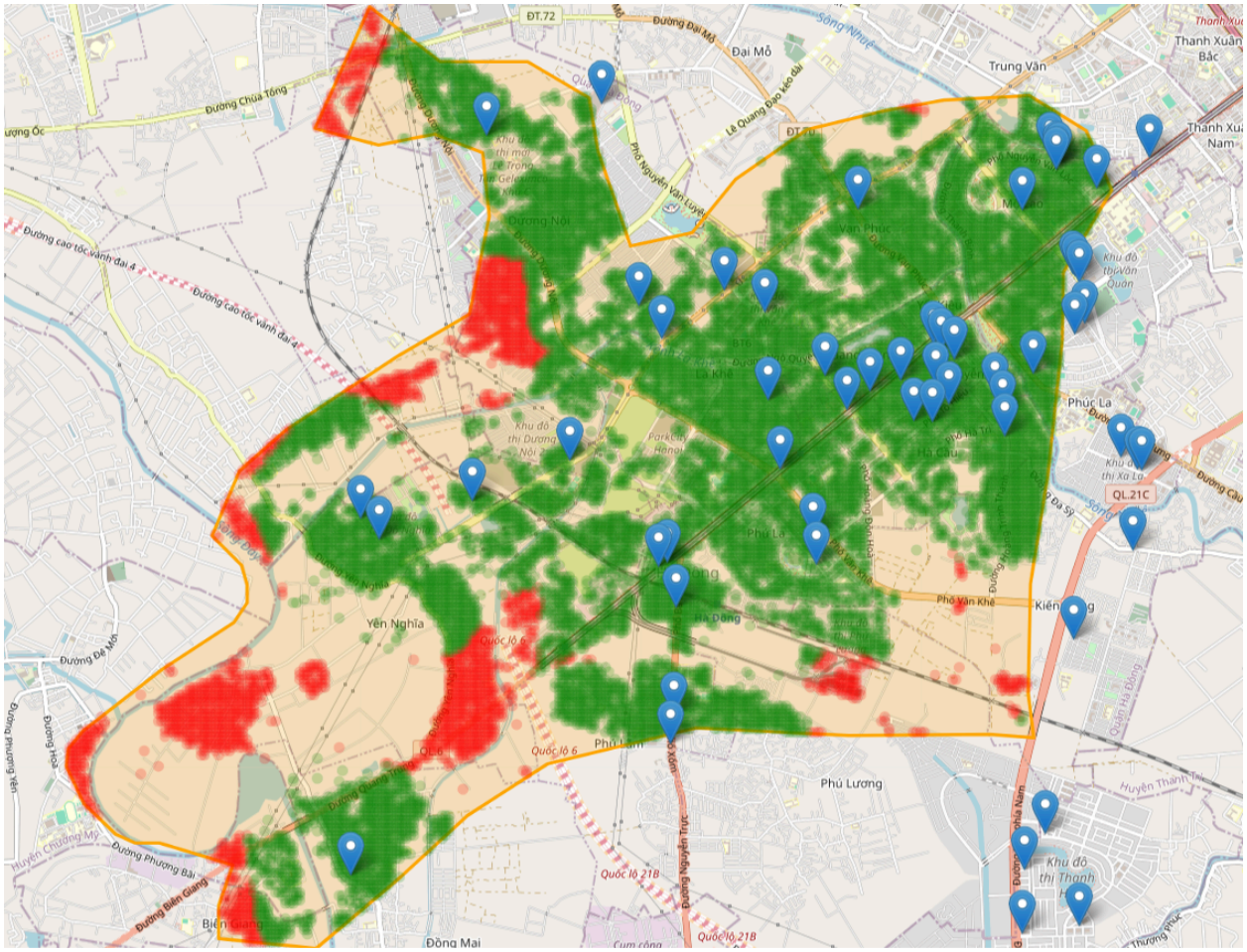
```python
def get_isochrone_osm (each_hosp,travel_time_secs):
  body = {"locations":[[each_hosp.x,each_hosp.y]],"range":[travel_time_secs],"range_type":'time'}
  headers = {
      'Accept': 'application/json, application/geo+json, application/gpx+xml, img/png; charset=utf-8',
      'Authorization': '5b3ce3597851110001cf62486073a1d2472543b7b01af3949d4d5319',
      'Content-Type': 'application/json; charset=utf-8'
  }
  call = requests.post('https://api.openrouteservice.org/v2/isochrones/foot-walking', json=body, headers=headers)
  if(call.status_code==200):
    geom = (json.loads(call.text)['features'][0]['geometry'])
    polygon_geom = Polygon(geom['coordinates'][0])
    return polygon_geom
  else:
    return None
```

Áp dụng tính toán khả năng tiếp cận cho các vị trí siêu thị hiện tại

```
Population with Access: 88.48 %
```

Điều này cho biết khả năng dân số tiếp cận được bằng **88.48%**

Hiển thị khả năng tiếp cận với các cơ sở chăm sóc sức khỏe hiện tại lên bản đồ bằng Folium



- Màu xanh thể hiện người dân có khả năng tiếp cận
- Màu đỏ thể hiện người dân không có khả năng tiếp cận

III - Mục tiêu đặt ra cho bài toán

Trong kế hoạch mở rộng kinh doanh sắp tới, chúng tôi dự định mở thêm 5 địa điểm mới để nâng cao hiệu quả hoạt động và tối ưu hóa chi phí. Để đạt được mục tiêu này, bài toán đặt ra là xác định 5 vị trí tối ưu nhất cho các điểm bán hàng mới với các tiêu chí sau:

- Lựa chọn các vị trí điểm bán 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 điểm bán không vượt quá 15 phút.

- Xem xét các khu vực mà người dân có thể tiếp cận điểm bán bằng cách đi bộ.

IV - Phát triển mô hình

a) Xây dựng bài toán

Ví dụ: Chúng tôi cần một biến số cho mỗi khu vực dân cư để xác định xem điểm bán hàng mở ra có thể phục vụ khu vực đó hay không, và một biến số khác cho mỗi điểm bán hàng để cho biết liệu điểm đó có được mở hay không. Ký hiệu toán học điển hình để biểu diễn mô hình bắt đầu bằng cách đặt tên các tập hợp hỗ trợ các chỉ số của biến và các tham số mô hình được rút ra từ dữ liệu.

Bài toán tối ưu hóa bao gồm:

- I — tập hợp các hộ gia đình
- J — tập hợp các địa điểm điểm bán hàng tiềm năng
- Ji — tập hợp các địa điểm cửa hàng tiềm năng trong khả năng truy cập của hộ gia đình $i \in I$. Lưu ý: $Ji \subseteq J$.

Trong đó

- vi — số người trong hộ gia đình $i \in I$.

Mô hình **maximal covering** như trong bài viết của Church và ReVelle

Mô hình này xác định các biến $\mathbf{z_i}$ cho mỗi khu vực dân cư $\mathbf{i \in I}$ để cho biết liệu khu vực dân cư đó có thể được phục vụ bởi một điểm bán hàng mở tại $\mathbf{j \in J}$ hay không, dẫn đến mô hình hoàn chỉnh như sau:

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

Dòng đầu tiên nêu mục tiêu tối đa hóa tổng số hộ gia đình được phục vụ, trong khi dòng thứ hai (sau _subject to:_) liệt kê ràng buộc đầu tiên: mỗi hộ gia đình chỉ được phục vụ nếu có ít nhất một cửa hàng trong tầm tay mở cửa. Sau đó, số lượng cửa hàng mở sẽ hạn chế việc lựa chọn và cuối cùng, tính chất nhị phân của các biến được sử dụng sẽ được chỉ định.

b) Phương pháp thực hiện

Môi trường thực nghiệm sử dụng Python

Xây dựng thuật toán tối ưu hóa khả năng truy cập trong Python, các bước chính là:

- Nhập các thư viện pyomo, highspy
- Xây dựng lưới địa giới hành chính có tiềm năng đại diện và tính toán khả năng truy cập tối đa
- Xác định mục tiêu tối ưu hóa là tăng khả năng truy cập
- Mô hình hóa bài toán như bài toán tính toán độ phủ tối đa (maximal covering)
- Sử dụng highspy để tính toa
- Phân tích số liệu về khả năng truy cập của các cơ sở giả sử mới

- Hiện thị khả năng truy cập trên bản đồ thông qua Folium

Bước 1. Nhập các thư viện cần thiết

```
import folium as fl
import pandas as pd
import geopandas as gpd
from hdx.api.configuration import Configuration
from hdx.data.resource import Resource
import urllib.request
import requests
import json
import requests
import itertools

from shapely.geometry import Polygon,MultiPolygon
from shapely.ops import unary_union

from gadm import GADMDownloader
import numpy as np

import pyomo.environ as pyo
from gethighs import HiGHS
```

Bước 2. Đọc và xử lý dữ liệu

1. **Đọc các bộ dữ liệu về bản đồ, mật độ dân số, và các địa điểm bán hàng hiện tại**

   Dữ liệu về mật độ dân số khu vực Hà Nội:

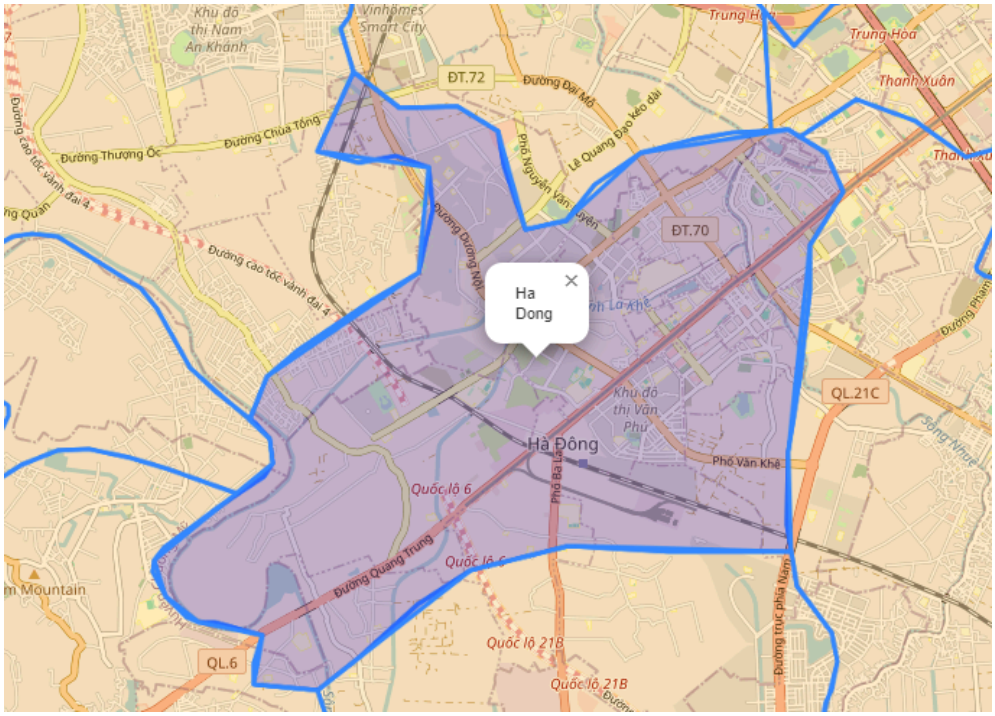
```
population_aoi_gdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 864004 entries, 0 to 864003
Data columns (total 12 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   ID           864004 non-null  int64
 1   xcoord       864004 non-null  float64
 2   ycoord       864004 non-null  float64
 3   population   864004 non-null  float64
 4   geometry     864004 non-null  object
 5   index_right  864004 non-null  int64
 6   shapeName    864004 non-null  object
 7   shapeISO     864004 non-null  object
 8   shapeID      864004 non-null  object
 9   shapeGroup   864004 non-null  object
 10  shapeType    864004 non-null  object
 11  opacity      864004 non-null  float64
dtypes: float64(4), int64(2), object(6)
memory usage: 79.1+ MB
```

Dữ liệu về các điểm bán hàng hiện tại:

```
<class 'pandas.core.frame.DataFrame'>
Index: 68 entries, 42 to 171
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 68 non-null     int64
 1   name               59 non-null     object
 2   geometry           68 non-null     geometry
 3   index_right        68 non-null     int64
 4   shapeName          68 non-null     object
 5   shapeISO           68 non-null     object
 6   shapeID            68 non-null     object
 7   shapeGroup         68 non-null     object
 8   shapeType          68 non-null     object
 9   cachment_area_osm  68 non-null     geometry
 10  cachment_area      68 non-null     geometry
 11  id_with_access     68 non-null     object
 12  pop_with_access    68 non-null     float64
dtypes: float64(1), geometry(3), int64(2), object(7)
memory usage: 7.4+ KB
```

Dữ liệu bản đồ khu vực hành chính được chọn:



Bước 2. Xây dựng hàm tính toán khả năng tiếp cận

Đoạn mã Python sau thực hiện việc tính toán isochrone (vùng thời gian di chuyển) sử dụng eKMap API, cụ thể là với chế độ di chuyển bằng cách đi bộ.

Hàm này giúp tính toán vùng có thể đi bộ trong khoảng thời gian nhất định từ một điểm (ở đây là cửa hàng). Kết quả trả về là một đa giác (polygon) đại diện cho khu vực có thể tiếp cận trong thời gian đó.

```
def get_isochrone_osm (each_hosp,travel_time_secs):
  body = {"locations":[[each_hosp.x,each_hosp.y]],"range":[travel_time_secs],"range_type":'time'}
  headers = {
      'Accept': 'application/json, application/geo+json, application/gpx+xml, img/png; charset=utf-8',
      'Authorization': '5b3ce3597851110001cf62486073a1d2472543b7b01af3949d4d5319',
      'Content-Type': 'application/json; charset=utf-8'
  }
  call = requests.post('https://api.openrouteservice.org/v2/isochrones/foot-walking', json=body, headers=headers)
  if(call.status_code==200):
    geom = (json.loads(call.text)['features'][0]['geometry'])
    polygon_geom = Polygon(geom['coordinates'][0])
    return polygon_geom
  else:
    return None
```

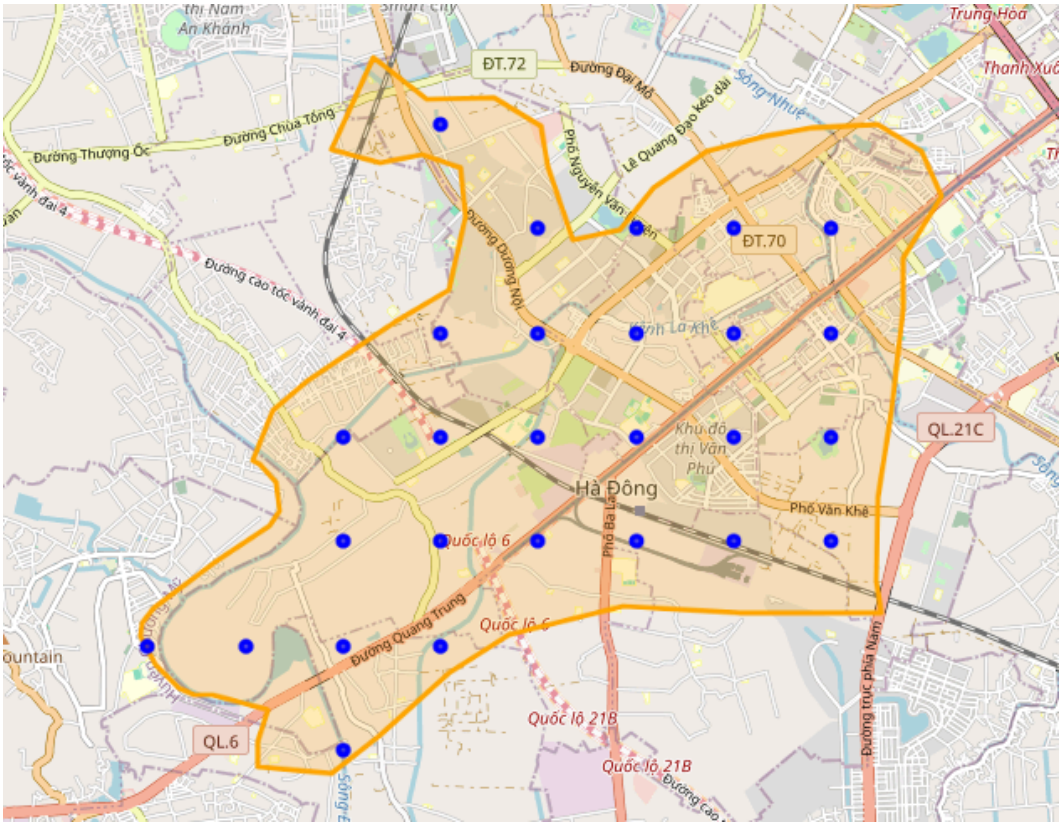
Bước 3. Lưới địa giới hành chính có tiềm năng đại diện

Tối ưu hóa việc bố trí các cửa hàng và cửa hàng mới là điều cần thiết để đảm bảo tối ưu chi phí. Điều này thường liên quan đến việc phân tích các vị trí tiềm năng, trong trường hợp không có đề xuất địa điểm cụ thể từ các nguồn chính thức, có thể được ước tính gần đúng thông qua một lưới đại diện trong khu vực mục tiêu.

```python
def generate_grid_in_polygon(
    spacing: float, geometry: MultiPolygon
) -> gpd.GeoDataFrame:
    """
    This Function generates evenly spaced points within the given GeoDataFrame.
    The parameter 'spacing' defines the distance between the points in coordinate units.
    """

    # Get the bounds of the polygon
    minx, miny, maxx, maxy = geometry.bounds

    # Square around the country with the min, max polygon bounds
    # Now generate the entire grid
    x_coords = list(np.arange(np.floor(minx), int(np.ceil(maxx)), spacing))
    y_coords = list(np.arange(np.floor(miny), int(np.ceil(maxy)), spacing))
    mesh = np.meshgrid(x_coords, y_coords)
    grid = gpd.GeoDataFrame(
        data={"longitude": mesh[0].flatten(), "latitude": mesh[1].flatten()},
        geometry=gpd.points_from_xy(mesh[0].flatten(), mesh[1].flatten()),
        crs="EPSG:4326",
    )
    grid = gpd.clip(grid, geometry)
    grid = grid.reset_index(drop=True).reset_index().rename(columns={"index": "ID"})

    return grid
```

Một mạng lưới như vậy cung cấp một khuôn khổ khởi đầu để xem xét nơi thành lập các điểm bán hàng mới nhằm tối đa hóa khả năng tiếp cận cho nhóm dân số chưa được phục vụ đầy đủ. Đoạn mã phác thảo một hàm Python generate_grid_in_polygon được thiết kế để tạo lưới biểu diễn này trong một khu vực địa lý nhất định, được biểu thị dưới dạng đối tượng MultiPolygon .

Hàm này tạo ra một loạt các điểm cách đều nhau—được xác định bởi tham số khoảng cách —trên phạm vi hình học được cung cấp, dẫn đến 443 vị trí tiềm năng ở Hà Nội



Vị trí các điểm tiềm năng trên bản đồ dưới dạng lưới 0,01 độ

Tính toán khả năng tiếp cận cho toàn bộ điểm tiềm năng

Để tối ưu hóa vị trí của các điểm bán hàng tiềm năng, phân tích của chúng tôi bao gồm việc tính toán đồng thời cho từng vị trí được đề xuất, sử dụng cùng thông số đi bộ 15 phút đã được áp dụng trước đây cho các cơ sở hiện có.

Bằng cách sử dụng dữ liệu này, chúng ta quyết định phân khúc dân số nào không được tiếp cận và có thể được phục vụ bởi các cơ sở tiềm năng này. Sau đó, chúng tôi tổng hợp dữ liệu tiếp cận từ các địa điểm hiện có và tiềm năng để tính toán khả năng tiếp cận tối đa có thể nếu tất cả các địa điểm tiềm năng được mở cùng với các cơ sở hiện có.

```
Maximum access attainable with this potential location list: 99.82 %
```

Điều này cho biết khả năng tiếp cận tối đa có thể đạt được với danh sách các vị trí tiềm năng này là **99.82%.**

Bước 4. Xây dựng mô hình tối ưu hóa khả năng tiếp cận

1. Xác định mô hình tối ưu hóa

```
def model_max_covering(w, I, J, JI, p, J_existing):

    assert set(J_existing).issubset(set(J))

    m = pyo.ConcreteModel('MaxCovering')

    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=J_existing)
    m.nof_fixed = pyo.Param(mutable=False, within=pyo.Integers, default=len(J_existing))

    @m.Param(m.I, within=pyo.NonNegativeReals)
    def w(m, i):
        return w[i]

    @m.Param(m.I, within=pyo.Any)
    def JI(m, i):
        return JI.get(i,[])

    m.x = pyo.Var(m.J, within=pyo.Binary)
    m.z = pyo.Var(m.I, within=pyo.Binary)

    @m.Objective(sense=pyo.maximize)
    def covering(m):
        return pyo.quicksum(m.w[i] * m.z[i] for i in m.I)

    @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])

    @m.Constraint()
    def budget(m):
        return pyo.quicksum(m.x[j] for j in m.J) <= m.nof_fixed + m.p

    @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 . Phải mất một số tham số:

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

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

2. Chuẩn bị dữ liệu đầu vào

```
w = population_aoi.set_index('ID').population.to_dict()

J_existing = set(selected_hosp.id)
J_potential = set(potential_locations_150.ID )

J = sorted( J_existing | J_potential )
I = sorted( set(population_aoi.ID) )

IJ_existing = selected_hosp.set_index('id').id_with_access.to_dict()
IJ_potential = potential_locations_150.set_index('ID').id_with_access.to_dict()

IJ = IJ_existing | IJ_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 = sorted(J_existing)
```

Trích xuất các dữ liệu liên quan từ DataFrames

- J_existing : Bộ ID cửa hàng hiện có.
- J_potential : Tập hợp ID vị trí cửa hàng mới tiềm năng
- I1 : Tập hợp tất cả các mã số hộ gia đình.
- IJ_existing : Từ điển ánh xạ ID cửa hàng hiện có tới các nhóm hộ gia đình mà họ có thể phục vụ.

- IJ_potential : Từ điển ánh xạ ID cửa hàng tiềm năng tới các nhóm hộ gia đình mà họ có thể phục vụ .
- JI1 : Bản đồ ngược lại từ hai bản đồ trước, đưa ra các tập hợp cửa hàng tiềm năng cho mỗi hộ gia đình.

Bước 5. Tính toán và hiển thị kết quả

Khởi tạo mô hình và lựa chọn bộ giải pháp

```
model = model_max_covering(w, I, J, JI, 0, J_existing)
solver = HiGHS(time_limit=10, mip_heuristic_effort=0.2, mip_detect_symmetry="on")
```

Sau đó, mô hình này được giải bằng bộ giải HiGHS và trả về ID của các cửa hàng (đã mở) đã chọn.

Cấu hình Solver cho MaxCovering:

Khi giải bài toán MaxCovering với solver như HiGHS, các tham số cấu hình có thể ảnh hưởng đến hiệu suất và chất lượng của giải pháp. Đây là cách các tham số bạn đã sử dụng có thể ảnh hưởng đến bài toán MaxCovering:

- time_limit = 10:
    - Đặt giới hạn thời gian tối đa là 10 giây cho quá trình tìm kiếm giải pháp. Đây là cách để đảm bảo solver không chạy quá lâu.
- mip_heuristic_effort=0.2:
    - Điều chỉnh mức độ nỗ lực dành cho các phương pháp heuristic trong bài toán MIP. Với giá trị 0.2, solver sẽ dùng 20% nỗ lực cho các phương pháp heuristic, giúp cải thiện khả năng tìm ra các giải pháp gần đúng tốt hơn, nhưng có thể làm tăng thời gian giải quyết.
- mip_detect_symmetry="on":
    - Tham số này bật hoặc tắt việc phát hiện đối xứng trong bài toán MIP. Khi phát hiện đối xứng được bật ("on"), solver sẽ

tìm kiếm các đối xứng trong mô hình và loại bỏ chúng để giảm bớt khối lượng tính toán. Việc phát hiện và loại bỏ đối xứng có thể cải thiện hiệu suất giải quyết, đặc biệt trong các bài toán có cấu trúc đối xứng rõ ràng.

Tìm kiếm giải pháp tối ưu

Trong khối mã sau, một danh sách result_list được tạo để lưu trữ kết quả cho các ngân sách khác nhau (số lượng điểm bán hàng sẽ mở).

```
def process_value(each_val):
    model.p = each_val
    solver.solve(model)
    opened_ids = get_selected(model.x)
    selected_new = potential_locations_150[potential_locations_150['ID'].isin(opened_ids)]
    list_ids_access_new = list(selected_new['id_with_access'].values)
    list_ids_access_new = list(itertools.chain.from_iterable(list_ids_access_new))
    list_ids_access = list_ids_access_old + list_ids_access_new
    pop_with_access = population_aoi[population_aoi['ID'].isin(list_ids_access)]
    pop_without_access = population_aoi[~population_aoi['ID'].isin(list_ids_access)]
    pop_percentage = round(pop_with_access['population'].sum()*100/population_aoi['population'].sum(), 2)
    return each_val + len(selected_hosp), pop_percentage

# List of values to process
values_to_process = range(len(potential_locations_150))

result_list = []
with concurrent.futures.ProcessPoolExecutor() as executor:
    # Map the process_value function to each value
    results = list(tqdm(executor.map(process_value, values_to_process), total=len(values_to_process)))
    result_list.extend(results)

print(result_list)
```

- Hàm get_ids được gọi để lấy bộ ID cửa hàng tối ưu để mở
- Dân số có và không có quyền truy cập được tính toán dựa trên các cửa hàng đã chọn
- Tỷ lệ phần trăm dân số có quyền truy cập được tính toán
- Tỷ lệ phần trăm này và ngân sách tương ứng được thêm vào result_list

Kết quả đạt được:

```
selected_new
```

	ID	longitude	latitude	geometry	cachment_area_osm	id_with_access	pop_with_access
0	0	105.95	20.650	POINT (105.95 20.65)	POLYGON ((105.90308 20.64056, 105.9035 20.6400...	[5927723, 5927724, 5927725, 5928528, 5928529, ...	37277.0
1	1	106.05	20.675	POINT (106.05 20.675)	POLYGON ((106.01702 20.7045, 106.01699 20.7043...	[5840716, 5842349, 5843224, 5843225, 5843226, ...	51580.0
6	6	105.85	20.650	POINT (105.85 20.65)	POLYGON ((105.81324 20.66952, 105.81337 20.668...	[5922498, 5923345, 5923346, 5923347, 5924176, ...	41890.0
7	7	105.85	20.675	POINT (105.85 20.675)	POLYGON ((105.80952 20.68801, 105.81144 20.684...	[5856455, 5856472, 5857446, 5857447, 5857463, ...	47302.0
9	9	105.90	20.675	POINT (105.9 20.675)	POLYGON ((105.8715 20.66725, 105.87244 20.6671...	[5860204, 5860205, 5860206, 5861132, 5861133, ...	35966.0

Danh sách tọa độ các điểm tiềm năng sau khi chạy mô hình

Bước 6. Đánh giá kết quả

Phương pháp đánh giá sử dụng Đường cong Pareto:

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

Trực 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 cửa hàng được mở (ngân sách), trong khi trục y biểu thị tỷ lệ dân số được tiếp cận các cơ sở chăm sóc sức khỏe. Khi ngân sách (và do đó, số lượng cửa hàng mở) tăng lên, độ bao phủ cho người dân có xu hướng được cải thiện.

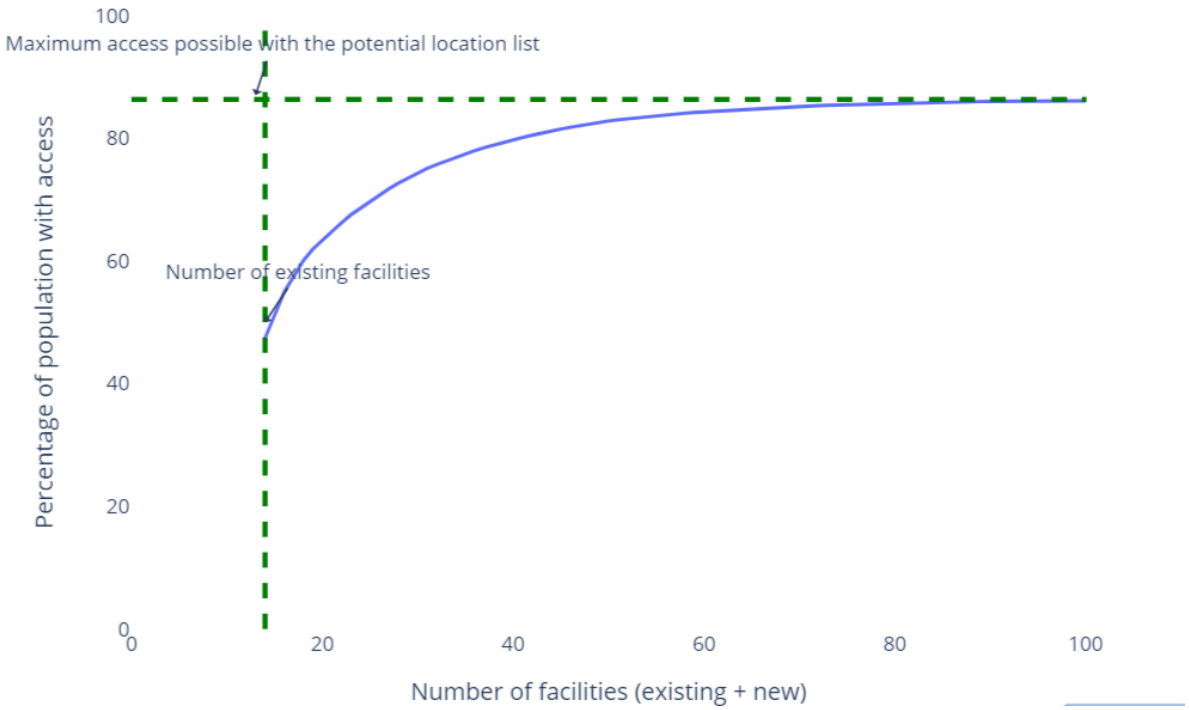
Tuy nhiên, tốc độ cải thiện thường giảm dần, dẫn đến đường cong trở nên phẳng hơn khi có thêm nhiều cửa hàng vượt quá một điểm nhất định. Đường cong này được gọi là đường cong Pareto hoặc biên giới Pareto, vì nó đại diện cho tập hợp các giải pháp không bị chi phối - các giải pháp trong đó việc tăng một mục tiêu (ví dụ: độ bao phủ dân số) chỉ có thể đạt được bằng cách hy sinh mục tiêu kia (ví dụ: tăng dân số). số cửa hàng/ngân sách). 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 cửa hàng 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 cửa hàng 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.

Mã bên dưới sử dụng mô-đun Plotly.express để trực quan hóa đường giới hạn Pareto cho kết quả tối ưu hóa.

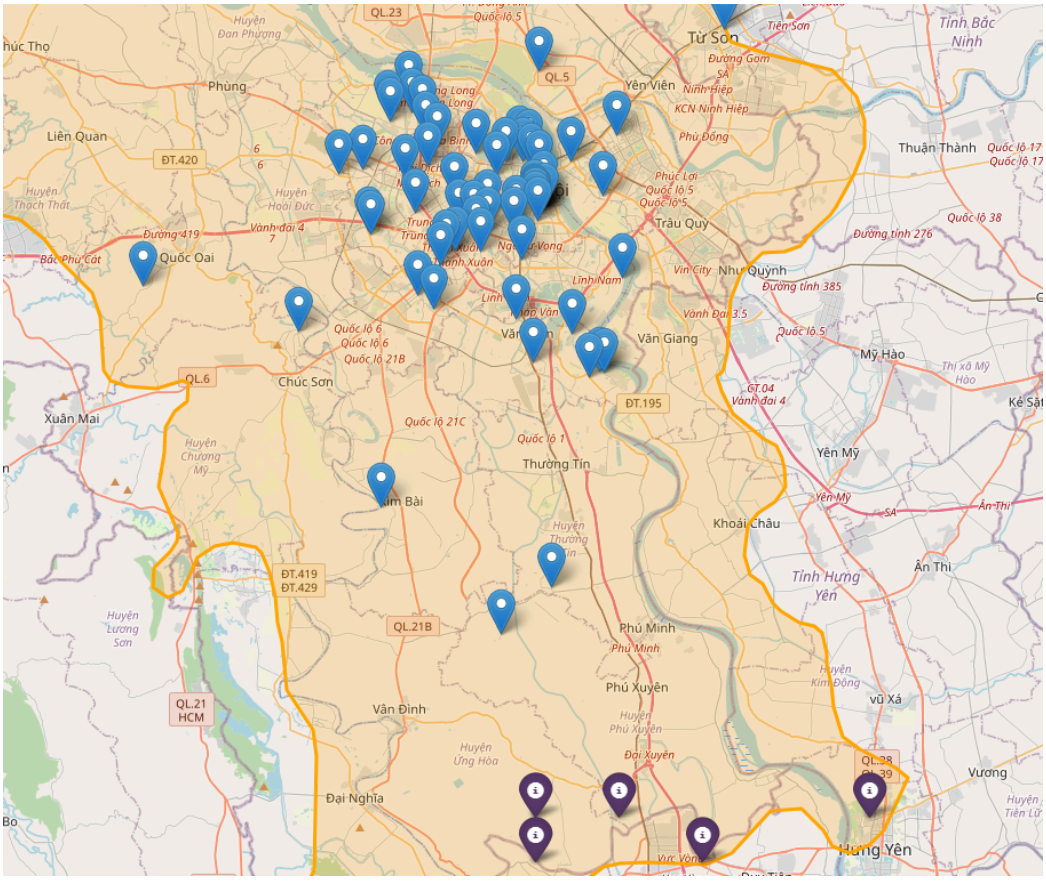
```python
# Extract data for plotting
x_values = [item[0] for item in result_list]
y_values = [item[1] for item in result_list]

# Create a plotly graph
fig = go.Figure(data=go.Scatter(x=x_values, y=y_values, mode='lines'))

# Update the layout
fig.update_layout(
    title="Pareto Curve",
    xaxis_title="Number of facilities (existing + new)",
    yaxis_title="Percentage of population with access",
    plot_bgcolor='white',
    yaxis=dict(range=[0, 100]),
    xaxis=dict(range=[0, 100]),
    width=1200
)

# Add vertical line and annotation at x=14
fig.add_vline(x=14, line_width=3, line_dash="dash", line_color="green")
fig.add_annotation(
    x=14, y=50,
    text="Number of existing facilities",
    showarrow=True,
    arrowhead=1,
    ax=20,
    ay=-30
)

# Add vertical line and annotation at x=14
fig.add_hline(y=max_access_possible, line_width=3, line_dash="dash", line_color="green")
fig.add_annotation(
    x=13, y=87,
    text="Maximum access possible with the potential location list",
    showarrow=True,
    arrowhead=1,
    ax=10,
    ay=-30
)

# Show the figure
fig.show()
```

pareto_visualization.py được lưu trữ bởi GitHub



V - Kết luận

Mô hình tối ưu hóa của chúng tôi đã xác định một số địa điểm chiến lược cho các điểm bán hàng mới, tăng cường khả năng tiếp cận tổng thể. Với hạn chế ngân sách đã chọn là mở thêm 5 cơ sở , phân tích cho thấy những địa điểm này là những lựa chọn hiệu quả nhất để tăng cường độ bao phủ, tăng tỷ lệ dân số có khả năng tiếp cận từ **88.48%** (chỉ với các cơ sở hiện có) lên **92.04%.**



Với ngân sách mở 5 điểm bán mới, tỷ lệ dân số có thể tiếp cận tăng từ 88.48% lên 92.04%.

Các vị trí tối ưu cho các điểm bán mới được xác định là các điểm chiến lược giúp mở rộng khả năng tiếp cận cho phần lớn khách hàng tiềm năng

VI - Định hướng tiếp theo

Việc đánh giá địa điểm tối ưu không chỉ dựa trên các tiêu chí hiện tại mà còn cần mở rộng để xem xét thêm các yếu tố khác như môi trường kinh doanh và các yếu tố kinh tế - xã hội. Trong các nghiên cứu tiếp theo, chúng tôi dự kiến sẽ tích hợp những yếu tố này vào quá trình đánh giá để nâng cao tính chính xác và hiệu quả của việc lựa chọn địa điểm.

Hiện tại, thực nghiệm của chúng tôi tập trung vào lĩnh vực kinh doanh sản phẩm Mẹ & Bé. Trong tương lai, chúng tôi có kế hoạch mở

rộng nghiên cứu để áp dụng phương pháp đánh giá cho các lĩnh vực khác như ngành hàng tiêu dùng nhanh (FMCG), ngành ăn uống và nhiều lĩnh vực kinh doanh khác, nhằm tạo ra giải pháp tối ưu cho nhiều loại hình kinh doanh.