Algorytm Odkrywania Wzorców Kolokacji

Wprowadzenie

Algorytm do odkrywania wzorców kolokacji w danych przestrzennych został zaimplementowany na podstawie artykułu "Discovering colocation patterns from spatial data sets: a general approach" autorstwa Yan Huang, Shashi Shekhar i Hui Xiong. Głównym celem algorytmu jest identyfikacja zbiorów cech przestrzennych (typów obiektów), które często występują blisko siebie w przestrzeni geograficznej.

Klasa `ColocationPattern`

Klasa `ColocataionPattern` reprezentuje jeden wzorzec kolokacji, zawierając cechy przestrzenne wchodzące w skład wzorca, wartość wskaźnika uczestnictwa (participation index) dla wzorca oraz listę instancji kolokacji mu odpowiadających.

Incjalizacja

```
def __init__(
    self,
    types: Tuple[FeatureType],
    participation_index: float,
    instances: List[PatternInstance],
):
```

Parametry:

- `types`: cechy przestrzenne wchodzące w skład wzorca.
- `participation_index`: wskaźnik uczestnictwa dla wzorca.
- `instances`: lista instacji kolokacji odpowiadających wzorcowi.

Opis: Konstruktor inicjalizuje obiekt na podstawie danych odkrytego przez algorytm wzorca.

Metoda `__str__`

```
def __str__(self) -> str:
```

Opis: Zwraca tekstową reprezentację wzorca.

Metoda `to_dict`

```
def to_dict(self) -> Dict[str, object]:
```

Opis:

Konwertuje wzorzec do reprezentacji słownikowej i zwraca ją. Słownik zawiera klucze `types`, `participation_index` oraz `num_instances`.

Klasa `ColocationMiner`

Klasa `ColocationMiner` jest głównym komponentem implementacji, odpowiedzialnym za odkrywanie wzorców kolokacji z danych przestrzennych.

Inicjalizacja

```
def __init__(self, radius: float = 0.005, min_prevalence: float = 0.3):
```

Parametry:

- `radius`: Promień sąsiedztwa (odległość progowa) określający, kiedy dwa obiekty są uznawane za sąsiadujące.
- `min_prevalence`: Minimalny próg wskaźnika uczestnictwa (participation index) - miara częstości występowania wzorca.

Opis: Konstruktor inicjalizuje podstawowe parametry oraz struktury danych wykorzystywane w algorytmie:

- `patterns`: Lista odkrytych wzorców kolokacji.
- `spatial_indices`: Słownik indeksów przestrzennych (KDTree) dla każdego typu obiektu.
- `instance_neighbors`: Słownik przechowujący relacje sąsiedztwa dla wszystkich instancji.
- `participation_ratios`: Słownik przechowujący współczynniki uczestnictwa dla każdego typu w każdym wzorcu.

Metoda `fit`

```
def fit(self, df: pd.DataFrame) -> None:
```

Parametry:

- `df`: DataFrame zawierający dane przestrzenne z kolumnami 'type', 'x', 'y'.

Opis: Główna metoda uruchamiająca proces odkrywania wzorców kolokacji. Proces przebiega w następujących krokach:

1. Przygotowanie danych (dodanie identyfikatorów, grupowanie instancji według typów).
2. Budowa indeksów przestrzennych dla każdego typu obiektu.
3. Obliczenie wszystkich relacji sąsiedztwa.
4. Odkrycie wzorców rozmiaru 2 (pary typów).
5. Iteracyjne odkrywanie coraz większych wzorców (k > 2):
    - Generowanie kandydatów rozmiaru k.
    - Przycinanie kandydatów na podstawie górnych ograniczeń wskaźnika uczestnictwa.
    - Odkrywanie instancji pozostałych kandydatów.

- Obliczanie wskaźników uczestnictwa i filtrowanie według progu.

Metoda `_build_spatial_indices`

```
def _build_spatial_indices(self) -> None:
```

Opis: Tworzy indeksy przestrzenne (KDTree) dla każdego typu obiektu. Dla każdego typu obiektu:

1. Pobiera współrzędne wszystkich instancji tego typu.
2. Buduje drzewo KDTree dla tych współrzędnych.
3. Przechowuje drzewo, identyfikatory instancji i współrzędne w słowniku `spatial_indices`.

Ta optymalizacja eliminuje potrzebę wielokrotnego budowania drzew KDTree podczas wykonywania algorytmu.

Metoda `_precompute_all_neighbors`

```
def _precompute_all_neighbors(self) -> None:
```

Opis: Oblicza wszystkie relacje sąsiedztwa z góry (w ramach optymalizacji algorytmu). Dla każdej pary typów obiektów:

1. Używa wcześniej zbudowanych drzew KDTree do znalezienia wszystkich par instancji, które są w odległości mniejszej niż `radius`.
2. Zapisuje relacje sąsiedztwa w macierzy sąsiedztwa `instance_neighbors`.

Metoda `_discover_size_2_patterns`

```
def _discover_size_2_patterns(self) -> List[ColocationPattern]:
```

Opis: Odkrywa wzorce kolokacji rozmiaru 2 (pary typów obiektów) o wskaźniku uczestnictwa powyżej progu. Dla każdej pary typów obiektów, metoda:

1. Identyfikuje instancje obu typów, które uczestniczą we wzorcu (mają sąsiada drugiego typu).
2. Oblicza współczynniki uczestnictwa dla obu typów (jaki procent instancji każdego typu ma sąsiada drugiego typu).
3. Oblicza wskaźnik uczestnictwa jako minimum z tych współczynników.
4. Jeśli wskaźnik jest powyżej progu, tworzy nowy wzorzec kolokacji.
5. Zapisuje współczynniki uczestnictwa dla późniejszego wykorzystania w przycinaniu na podstawie wskaźnika uczestnictwa.

Metoda `_generate_candidates`

```
def _generate_candidates(self, k: int) -> List[Pattern]:
```

Parametry:

- `k`: Rozmiar generowanych kandydatów.

Opis: Generuje kandydatów na wzorce kolokacji rozmiaru k, wykorzystując zasadę apriori: wszystkie podzbiory częstego wzorca muszą być również częste. Metoda:

1. Pobiera wszystkie wzorce rozmiaru k-1 z poprzedniej iteracji.
2. Dla każdej pary wzorców, które mają identyczne k-2 pierwsze typy, łączy je, tworząc potencjalnego kandydata rozmiaru k.
3. Weryfikuje, czy wszystkie podzbiory rozmiaru k-1 potencjalnego kandydata są częstymi wzorcami.
4. Jeśli tak, dodaje kandydata do listy.

Metoda `_discover_frequent_patterns_for_candidates`

```
def _discover_frequent_patterns_for_candidates(self, candidates: List[Pattern]) ->
List[ColocationPattern]:
```

Parametry:

- `candidates`: Lista kandydatów na wzorce do oceny.

Opis: Sprawdza, którzy kandydaci spełniają kryterium minimalnego wskaźnika uczestnictwa. Dla każdego kandydata, metoda:

1. Znajduje wszystkie instancje wzorca za pomocą metody `_find_pattern_instances`.
2. Dla każdego typu w kandydacie, zlicza liczbę instancji, które uczestniczą we wzorcu.
3. Oblicza współczynniki uczestnictwa dla każdego typu.
4. Oblicza wskaźnik uczestnictwa jako minimum z tych współczynników.
5. Jeśli wskaźnik jest powyżej progu, tworzy nowy wzorzec kolokacji.
6. Zapisuje współczynniki uczestnictwa dla późniejszego wykorzystania w przycinaniu.

Metoda `_find_pattern_instances`

```
def _find_pattern_instances(self, pattern_types: Tuple[Pattern]) ->
List[PatternInstance]:
```

Parametry:

- `pattern_types`: Krotka typów tworzących wzorzec.

Opis: Znajduje wszystkie instancje danego wzorca za pomocą podejścia opartego na klikach w grafie sąsiedztwa. Metoda wykorzystuje wcześniej obliczone relacje sąsiedztwa. Jej działanie przebiega następująco:

1. Rozpoczyna od instancji pierwszego typu w wzorcu.
2. Iteracyjnie dodaje jeden typ na raz:

- Dla każdej częściowej instancji, sprawdza, czy może zostać rozszerzona o instancje bieżącego typu.
- Znajduje wszystkie instancje bieżącego typu, które są sąsiadami wszystkich elementów częściowej instancji.
- Tworzy nowe częściowe instancje przez dodanie tych kandydatów.

3. Jeśli na dowolnym etapie nie można znaleźć rozszerzeń, wzorzec nie ma instancji.
4. Ekstrahuje same identyfikatory z końcowych instancji.

Metoda `get_patterns`

```
def get_patterns(self) -> List[ColocationPattern]:
```

Opis: Zwraca wszystkie odkryte wzorce kolokacji posortowane według wskaźnika uczestnictwa (malejąco) i rozmiaru wzorca.

Klasa `ColocationDataset`

Bazowa, abstrakcyjna klasa zbioru danych umożliwiającego wykorzystanie do odkrywania wzorców kolokacji przestrzennych.

Inicjalizacja

```
def __init__(self):
```

Opis: Konstruktor inicjalizuje obiekt oraz domyślnie ustawia atrybut `_data` na `None`.

Metoda `load_data`

```
@abstractmethod
def load_data(self) -> pd.DataFrame:
```

Opis: Abstrakcyjna metoda wczytująca dane ze źródła danych i zwracająca `DataFrame` z wczytanymi danymi.

Klasa `OSMColocationDataset`

Klasa `OSMColocationDataset` jest implementacją zbioru danych do odkrywania kolokacj przestrzennych bazującego na danych o points of interest (POI) dostępnych w bazie danych geograficznych OpenStreetMap.

Inicjalizacja

```
def __init__(self, area: Tuple[float], poi_types: List[str]):
```

Parametry:

- `area`: Krotka zawierające współrzędne bounding box terenu w kolejności minimalna szerokość, minimalna długość, maksymalna szerokość, masymalna długość geograficzna.
- `poi_types`: Lista wybrancyh typów POI dostępnych w OpenStreetMap (możliwe typy można znaleźć pod tym adresem).

Opis: Konstruktor inicjalizuje obiekt oraz atrybuty klasy.

Metoda `load_data`

```
def load_data(self) -> pd.DataFrame:
```

Opis: Metoda wykorzystująca bilbiotekę `overpy` do odpytania bazy danych OSM o położenie POI występujących na obszarze podanym przy inicjalizacji klasy. Dla każdego POI zwracany jest identyfikator `id`, nazwa typu `type` oraz współrzędna szerokości geograficznej `x` oraz długości geograficznej `y`.

Klasa `GBIFColocationDataset`

Klasa `GBIFColocationDataset` jest implementacją zbioru danych do odkrywania kolokacj przestrzennych bazującego na danych o obserwacjach gatunków roślin i zwierząt zanotowanych w bazie danych Global Biodiversity Information Facility (GBIF).

Inicjalizacja

```
def __init__(self,
   area: Tuple[float],
   species_names: List[str],
   min_year: int = 2010,
   limit_per_species: int | None = None,
):
```

Parametry:

- `area`: Krotka zawierające współrzędne bounding box terenu w kolejności minimalna szerokość, minimalna długość, maksymalna szerokość, masymalna długość geograficzna.
- `species_names`: Lista nazw naukowych wybranych gatunków.
- `min_year`: Minimalny rok obserwacji do włączenia w zbiór danych.
- `limit_per_species`: Opcjonalny limit ilości obserwacji dla jednego gatunku w zbiorze.

Opis: Konstruktor inicjalizuje obiekt oraz atrybuty klasy.

Metoda `load_data`

```
def load_data(self) -> pd.DataFrame:
```

Opis: Metoda wywołuje metody `_get_species_key` oraz `_get_all_occurrences` dla każdego z podanych gatunków i zapisuje pobrane obserwacje w DataFrame. Dla obserwacji gatunku zwracany jest identyfikator `id`, nazwa gatunku `type` oraz współrzędna szerokości geograficznej `x` oraz długości geograficznej `y` na której dokonano obserwacji.

Metoda `_get_species_key`

```
def _get_species_key(self, species_name: str) -> int:
```

Parametry:

- `species_name`: Nazwa naukowa gatunku rośliny lub zwierzęcia.

Opis: Metoda wykorzystuje API GBIF do odnalezienia klucza w bazie danych odpowiadającego wybranemu gatunkowi.

Metoda `_get_all_occurrences`

```
def _get_all_occurrences(self, species_key: int, species_name: str) ->
List[Dict[str, Any]]:
```

Parametry:

- `species_key`: klucz w bazie GBIF odpowiadający gatunkowi.
- `species_name`: Nazwa naukowa gatunku rośliny lub zwierzęcia.

Opis: Metoda wykorzystuje API GBIF do pobrania wszystkich (lub ograniczonych limitem) obserwacji danego gatunku na obszarze ustawionym podczas inicjalizacji klasy.

Wyniki eksperymentów

Przeprowadzono eksperymenty odkrywania kolokacji przestrzennym na obszarze Warszawy (w przybliżeniu ograniczonym współrzędnymi [52.15, 20.85, 52.35, 21.15]). Dla przykładowego eksperymentu który przeanalizowano w dokumentacji, parameter `radius` klasy `ColocationMiner` został ustawiony na `0.002` (co w przybliżeniu daje sąsiedztwo w postaci okręgu o promieniu ~200m), natomiast próg wskaźnika uczestnictwa `min_prevalence` został ustawiony na `0.3`. Z bazy danych OpenStreetMap pobrano dane o wszystkich dostępnych 136 typach punktów zainteresowania. W takiej konfiguracji algorytm odkrył 151 kolokacji przestrzennych, które zostały przedstawione w poniższej tabeli:

	types	participation_index	num_instances
0	('bench', 'waste_basket')	0.822967	198721
1	('bench', 'bicycle_parking')	0.819588	86724
2	('bicycle_parking', 'waste_basket')	0.744263	45926
3	('atm', 'fast_food')	0.686654	2901

	types	participation_index	num_instances
4	('bench', 'bicycle_parking', 'waste_basket')	0.652976	1040012
5	('atm', 'vending_machine')	0.624488	5936
6	('fast_food', 'vending_machine')	0.618162	6047
7	('cafe', 'fast_food')	0.614978	2526
8	('bicycle_parking', 'restaurant')	0.610575	14150
9	('fast_food', 'restaurant')	0.594902	4595
10	('cafe', 'restaurant')	0.590564	3761
11	('bicycle_parking', 'parking_entrance')	0.588294	11826
12	('restaurant', 'vending_machine')	0.581887	8652
13	('atm', 'pharmacy')	0.581238	838
14	('atm', 'restaurant')	0.575922	2988
15	('cafe', 'vending_machine')	0.574991	4738
16	('bicycle_parking', 'vending_machine')	0.56701	24194
17	('atm', 'cafe')	0.558027	1770
18	('parking_entrance', 'restaurant')	0.532214	3761
19	('fast_food', 'pharmacy')	0.526221	834
20	('bicycle_parking', 'fast_food')	0.522115	10557
21	('bicycle_parking', 'parcel_locker')	0.52145	6420
22	('atm', 'bicycle_parking')	0.514633	8507
23	('atm', 'bank')	0.510638	1112
24	('parcel_locker', 'parking_entrance')	0.505	2679
25	('parking_entrance', 'waste_basket')	0.503908	13253
26	('bar', 'pub')	0.496	312
27	('pharmacy', 'vending_machine')	0.486044	1662
28	('parcel_locker', 'restaurant')	0.47375	1860
29	('dentist', 'doctors')	0.473477	406
30	('cafe', 'doctors')	0.471264	636
31	('restaurant', 'waste_basket')	0.466981	10299
32	('parcel_locker', 'waste_basket')	0.466038	6698
33	('doctors', 'fast_food')	0.459031	961

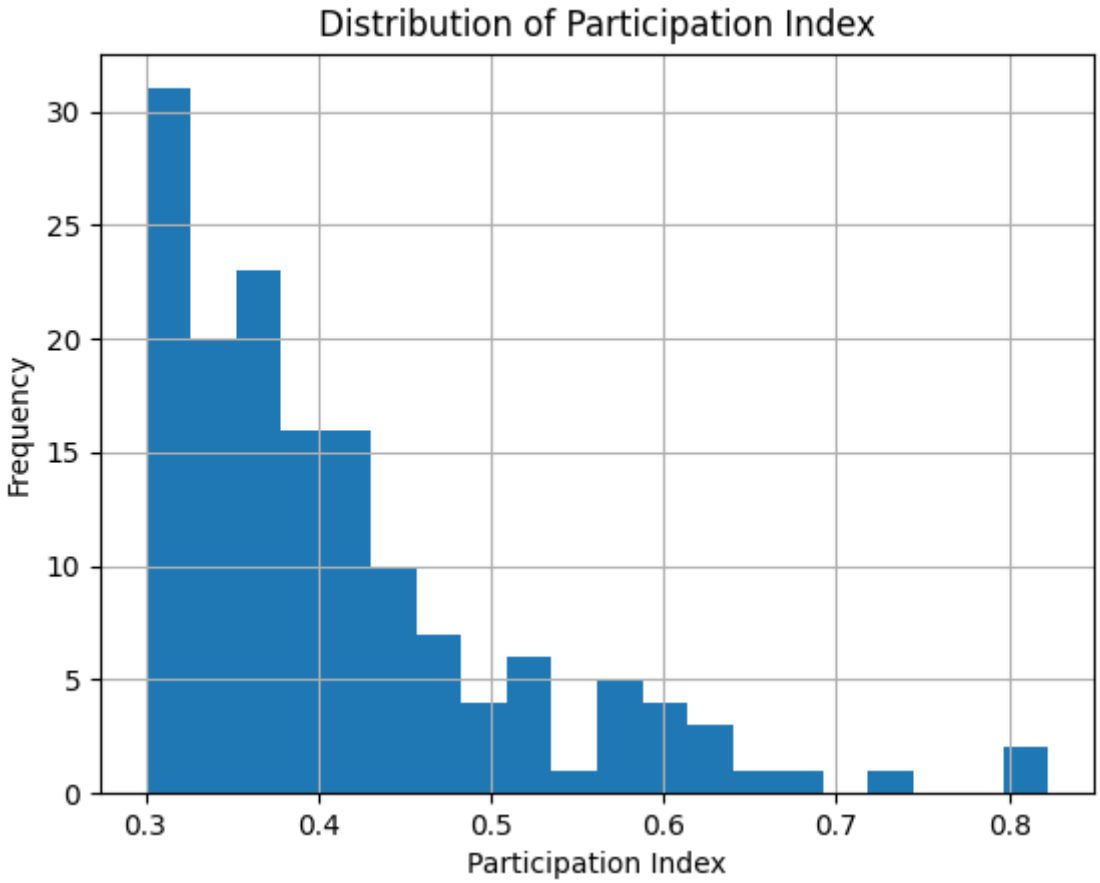
	types	participation_index	num_instances
34	('pharmacy', 'restaurant')	0.457701	1132
35	('bicycle_parking', 'cafe')	0.455105	7494
36	('atm', 'cafe', 'fast_food')	0.454545	7531
37	('atm', 'parcel_locker')	0.45375	1539
38	('atm', 'fast_food', 'vending_machine')	0.449944	18460
39	('vending_machine', 'waste_basket')	0.448248	19729
40	('cafe', 'dentist')	0.444444	539
41	('doctors', 'vending_machine')	0.434686	1972
42	('bench', 'vending_machine')	0.433832	38102
43	('bench', 'parking_entrance')	0.433347	25664
44	('cafe', 'fast_food', 'restaurant')	0.43167	15969
45	('bicycle_parking', 'restaurant', 'waste_basket')	0.430323	79112
46	('cafe', 'pharmacy')	0.428571	562
47	('bicycle_parking', 'parking_entrance', 'waste_basket')	0.42584	89266
48	('cafe', 'restaurant', 'vending_machine')	0.424078	20077
49	('dentist', 'pharmacy')	0.423146	341
50	('atm', 'fast_food', 'restaurant')	0.422451	12125
51	('fast_food', 'restaurant', 'vending_machine')	0.419197	24776
52	('bicycle_parking', 'vending_machine', 'waste_basket')	0.41752	144605
53	('bank', 'fast_food')	0.413216	974
54	('cafe', 'fast_food', 'vending_machine')	0.412728	15290
55	('bench', 'restaurant')	0.411423	18230
56	('bicycle_parking', 'pharmacy')	0.409544	3251
57	('atm', 'waste_basket')	0.409434	7919
58	('atm', 'restaurant', 'vending_machine')	0.407809	18480
59	('atm', 'cafe', 'vending_machine')	0.406029	12430
60	('bank', 'cafe')	0.404853	562
61	('dentist', 'restaurant')	0.400759	1107
62	('doctors', 'restaurant')	0.400217	1244
63	('bar', 'bureau_de_change')	0.399281	207

	types	participation_index	num_instances
64	('atm', 'cafe', 'restaurant')	0.397505	7796
65	('dentist', 'vending_machine')	0.396353	1557
66	('bicycle_parking', 'restaurant', 'vending_machine')	0.396076	74848
67	('dentist', 'fast_food')	0.395595	747
68	('bicycle_parking', 'parking_entrance', 'restaurant')	0.392418	23006
69	('atm', 'toilets')	0.391683	759
70	('bench', 'bicycle_parking', 'vending_machine')	0.391436	195138
71	('fast_food', 'parcel_locker')	0.39	1564
72	('atm', 'fast_food', 'pharmacy')	0.388788	2141
73	('toilets', 'vending_machine')	0.387793	1814
74	('atm', 'bicycle_parking', 'waste_basket')	0.386253	62288
75	('bicycle_parking', 'fast_food', 'restaurant')	0.38294	43043
76	('bicycle_parking', 'doctors')	0.381443	3887
77	('cafe', 'toilets')	0.378033	585
78	('doctors', 'pharmacy')	0.377939	333
79	('bench', 'bicycle_parking', 'parking_entrance')	0.377143	152532
80	('bicycle_parking', 'dentist')	0.375624	3402
81	('fast_food', 'toilets')	0.37533	822
82	('bench', 'bicycle_parking', 'restaurant')	0.37254	124526
83	('atm', 'bicycle_parking', 'restaurant')	0.371965	26207
84	('atm', 'bank', 'fast_food')	0.371806	3952
85	('atm', 'bank', 'cafe')	0.371648	2247
86	('post_box', 'post_office')	0.370909	110
87	('parcel_locker', 'pharmacy')	0.37	718
88	('parcel_locker', 'vending_machine')	0.369375	2704
89	('atm', 'parking_entrance')	0.367825	1744
90	('bench', 'parcel_locker')	0.367513	11115
91	('bicycle_parking', 'parcel_locker', 'waste_basket')	0.366312	32621
92	('bicycle_parking', 'fast_food', 'vending_machine')	0.365979	54174
93	('bicycle_parking', 'cafe', 'restaurant')	0.364317	32528

	types	participation_index	num_instances
94	('bench', 'vending_machine', 'waste_basket')	0.363273	394336
95	('atm', 'dentist')	0.362669	548
96	('atm', 'bicycle_parking', 'vending_machine')	0.362488	50741
97	('fast_food', 'waste_basket')	0.359164	7532
98	('bar', 'cafe')	0.355045	935
99	('atm', 'doctors')	0.352998	625
100	('atm', 'bicycle_parking', 'fast_food')	0.348853	26877
101	('bank', 'vending_machine')	0.343506	1731
102	('bench', 'parking_entrance', 'waste_basket')	0.341833	278043
103	('pharmacy', 'waste_basket')	0.341105	3212
104	('bank', 'pharmacy')	0.339964	361
105	('atm', 'pharmacy', 'vending_machine')	0.337551	3612
106	('bench', 'restaurant', 'waste_basket')	0.337533	153823
107	('bicycle_parking', 'fast_food', 'waste_basket')	0.334501	66192
108	('fast_food', 'parking_entrance')	0.333463	2092
109	('give_box', 'payment_centre')	0.333333	1
110	('love_hotel', 'swingerclub')	0.333333	1
111	('dentist', 'kindergarten')	0.332024	235
112	('cafe', 'parking_entrance')	0.331511	1683
113	('bank', 'cafe', 'fast_food')	0.331278	2209
114	('cafe', 'fast_food', 'restaurant', 'vending_machine')	0.329176	76608
115	('cafe', 'doctors', 'fast_food')	0.328634	2359
116	('bench', 'bicycle_parking', 'vending_machine', 'waste_basket')	0.328508	1700026
117	('fast_food', 'pharmacy', 'vending_machine')	0.328247	3714
118	('post_box', 'vending_machine')	0.327503	995
119	('bank', 'bureau_de_change')	0.326347	177
120	('parking_entrance', 'vending_machine')	0.324483	3530
121	('bicycle_parking', 'cafe', 'vending_machine')	0.323578	42389
122	('atm', 'fast_food', 'restaurant', 'vending_machine')	0.322126	67418

	types	participation_index	num_instances
123	('bicycle_parking', 'parcel_locker', 'parking_entrance')	0.321583	11541
124	('bar', 'fast_food')	0.319824	1515
125	('atm', 'cafe', 'fast_food', 'restaurant')	0.31833	42888
126	('atm', 'bank', 'vending_machine')	0.317454	7613
127	('atm', 'cafe', 'restaurant', 'vending_machine')	0.317245	43712
128	('car_rental', 'payment_terminal')	0.317073	149
129	('dentist', 'parking_entrance')	0.317064	1217
130	('atm', 'fast_food', 'toilets')	0.3163	3121
131	('atm', 'cafe', 'fast_food', 'vending_machine')	0.314849	47567
132	('bank', 'restaurant')	0.313991	976
133	('bicycle_parking', 'pharmacy', 'waste_basket')	0.313208	20073
134	('bicycle_parking', 'cafe', 'fast_food')	0.312936	25370
135	('bicycle_parking', 'parcel_locker', 'restaurant')	0.312604	11044
136	('pharmacy', 'restaurant', 'vending_machine')	0.312364	4812
137	('bench', 'bicycle_parking', 'restaurant', 'waste_basket')	0.312095	1190434
138	('bench', 'fast_food')	0.310823	12197
139	('bench', 'bicycle_parking', 'parking_entrance', 'waste_basket')	0.309551	2267153
140	('atm', 'bicycle_parking', 'parking_entrance')	0.309112	11740
141	('bank', 'post_office')	0.308696	122
142	('atm', 'bank', 'cafe', 'fast_food')	0.30837	8917
143	('bar', 'restaurant')	0.308026	2206
144	('bar', 'vending_machine')	0.307778	2211
145	('bench', 'bicycle_parking', 'parcel_locker')	0.306947	48698
146	('college', 'school')	0.303951	209
147	('fast_food', 'ice_cream')	0.303084	466
148	('atm', 'cafe', 'pharmacy')	0.300181	1232
149	('bureau_de_change', 'pub')	0.3	61
150	('bureau_de_change', 'taxi')	0.3	44

Wśród 151 odkrytych wzorców 93 składają się z dwóch typów, 49 z trzech i jednynie 9 z czterech punktów zainteresowania.
Poniższy histogram przedstawia rozkład wartości wskaźnika uczestnictwa wśród odkrytych wzorców. Na wykresie można zaobserwować, iż najwięcej odkrytych wzorców ma wartość wskaźnika uczestnictwa niewiele wyższą od progu, i wraz ze wzrotstem tego wskaźnika maleje ilość wzorców.



Z kolei następująca tabela przedstawia 10 typów punktów zainteresowania najczęściej pojawiających się w odkrytych wzorcach:

type	count
bicycle_parking	40
vending_machine	38
fast_food	38
atm	36
restaurant	35
cafe	30
waste_basket	23
bench	18
parking_entrance	17

type	count
pharmacy	17

Dosyć zaskakującym jest wystąpienie parkingu dla rowerów na pierwszym miejscu - chociaż z pewnością Warszawa jest miastem o dobrej infrastrukturze dla rowerów zwłaszcza na tle innych miast w polsce, to jednak parking dla rowerów zdecydownie nie jest obiektem uznawanym za najbardziej niezbędny dla większości społeczeństwa. Nie jest zaskoczniem jednak występowanie automatów z jedzeniem czy bankomatów na wysokich pozycjach, jako iż są to obiekty na ogół potrzebne i łatwe do ustawienia w zasadzie gdziekolwiek. Podobnie ławki oraz kosze na śmieci, które razem tworzą najbardziej częsty wzorzec, zgodnie z oczekiwaniami są obiektami wszechobecnymi w miejskim krajobrazie. Można również zaobserwować wysoką częstość występowania we wzorcach obiektów gastronomicznych.