Using BART system for expanding AGMs locations

By data science team
Jerry Gonzalez, Chi So and Weijie Yang
MIDS summer 2023 Section 8

Berkeley
UNIVERSITY OF CALIFORNIA

Business Goals

neo4j

- Identify extra locations by analyzing coverage of the BART station map.
- FInd the best route for delivery
- Identify alternative delivery methods

mongoDB

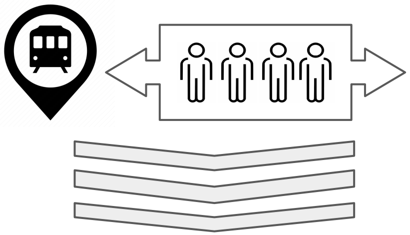
- Real-time delivery analysis
- Possibly using drones to help with deliveries

redis

- Real-time analysis of transportation capacity
- Helps to adjust the transportation strategy

Daily Exit Numbers are considered in Graph Algorithm

Daily Exit Data from 2020/01 to 2023/06



Get the Average Result in 3 yrs

Station	Average Capacity
12th Street	134
16th Street Mission	158
19th Street	122
24th Street Mission	154
Antioch	50
…	…

1.0 Integrate Capacity Data with Travel Times in Graph Algo

Relationship consideration

- Travel Time
- Average Daily Exits

Nature of Data

- Shorter travel time better
- High Exit numbers better

Integrate into Graph Algo

- Derive Community
- Shortest Path

weight = (factor_1 * time) + (factor_2 * (1 /daily_exit_numbers))

2.0 Analyze the business with Exit data after Graph Algo

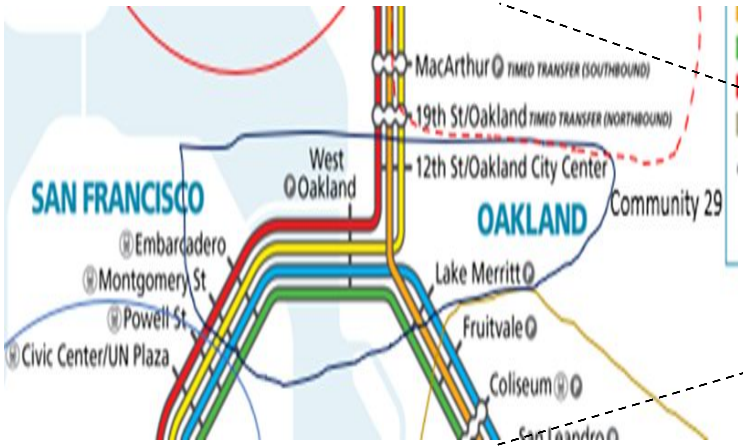
- Daily exit numbers reflect the capacity of BART stations
- Compute the community total exit numbers after Louvain Modularity, contribute to assign extra location in higher level
- Analyze the closeness of stations together with its average daily exits data

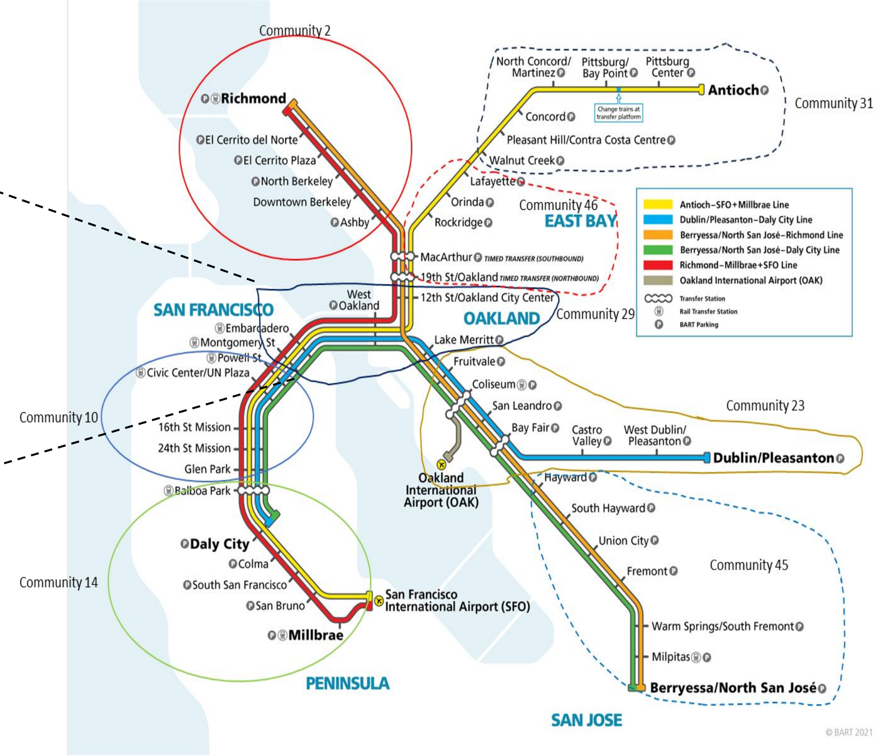
Louvain Modularity on BART map

	name	community	intermediate_community
0	Ashby	2	[15, 2]
1	Downtown Berkeley	2	[15, 2]
2	El Cerrito Plaza	2	[38, 2]
3	El Cerrito del Norte	2	[38, 2]
4	North Berkeley	2	[15, 2]
5	Richmond	2	[38, 2]
6	16th Street Mission	10	[3, 10]
7	24th Street Mission	10	[3, 10]
8	Civic Center	10	[10, 10]
9	Glen Park	10	[3, 10]
10	Powell Street	10	[10, 10]
11	Balboa Park	14	[14, 14]
12	Colma	14	[14, 14]
13	Daly City	14	[14, 14]
14	Millbrae	14	[42, 14]
15	SFO	14	[42, 14]
16	San Bruno	14	[42, 14]
17	South San Francisco	14	[14, 14]

- Applied Louvain modularity algorithm on a simplified version of the BART graph where there is 1 node per station and only 1 relationship each way between connected nodes.
- The resulting Neo4j graph has 50 nodes with 102 relationships.
- The algorithm produces 8 communities or clusters of nodes that are tightly interconnected.

Community map





8 Communities

- Community 2
- Community 10
- Community 14
- Community 23
- Community 29
- Community 31
- Community 45
- Community 46

Berkeley
UNIVERSITY OF CALIFORNIA

Applying the Short Path Algorithm to Weight Function

Adding Daily Exit Numbers in the Weight Function

- **Weight = (factor_1 * time) + (factor_2 * (1 /daily_exit_numbers))**
- **Weighted combination of exit counts and total time**
- **Delivery as many popular stations as possible with 1 single trip**

Couple with Real Time Analysis

- **Delivery to Bart stations in bulk order**
- **Items are first delivered to BART Station and then delivered individually to customers**
- **Real time exit count dynamically changes the algorithm result**
- **Broaden our customer outreach**

From Station	To Station	Weight
West Oakland	Powell Street	24.1
West Oakland	Rockridge	37.9
West Oakland	Walnut Creek	85.6
West Oakland	West Dublin	87.0
West Oakland	Richmond	89.8
West Oakland	South San Franci	100.9
West Oakland	Warm Springs	131.7

Centrality

Insights from Centrality Algorithm

- **Wasserman Faust Closeness:** West Oakland BART lines have highest closeness and heavily connected
- **Betweenness Centrality:** MacArthur, Rockridge, Lake Merritt and 12th Street are heavily passed
- **Avg Exit Number:** West Oakland, Mac Arthur and 12th Street have an average daily capacity near 100

Rank	Station	Closeness	Avg Exit Number
1	yellow West Oakland	0.105979	95
2	green West Oakland	0.105531	95
3	red West Oakland	0.105263	95
4	blue West Oakland	0.104821	95
5	yellow 12th Street	0.103775	133

Analysis and Solutions to Achieve Goals

- Add Extra Locations near West Oakland to alleviate the heavy connections
- Extra locations also contribute to divert the delivery capacity and the passing pressure of the nodes
- Although Rockridge and Orinda are heavily passed by connections, actual capacity reflects that not many people take the routes that pass them

Rank	Station	Betweenness	Avg Exit Number
0	yellow MacArthur	5999.809223	103
1	yellow Rockridge	5509.000000	55
2	orange Lake Merritt	5155.831877	83
3	orange 12th Street	5139.715461	133
4	yellow Orinda	4997.000000	28

MongoDB - Real Time Delivery Analysis & Drones

Real Time Data Analysis for Delivery Business

- **Real-time Data Processing:** MongoDB provides fast read and write operations, making it suitable for real-time data processing, enabling real-time monitoring and analysis of the delivery process.
- **Rich Query Language**: MongoDB's expressive query language allows complex queries and aggregations, enabling the company to derive valuable insights from the data.

Drones Integration, allocations and data analysis

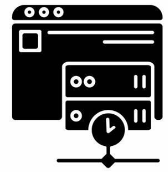
- **Internet of Things (IoT) Support**: MongoDB's support for IoT applications makes it an ideal choice for integrating with drones. This allows seamless data exchange between the drones and the backend system, enabling real-time tracking and management.
- **Scalability**: MongoDB allows to process a vast number of drones, deliveries, and station-related data, enabling to handle increasing data volumes and concurrent users.

MongoDB Codes: Data Model Design and Inserting Data

```python
import pymongo

client = pymongo.MongoClient('mongo_db_connection')
db = client['delivery_system_db']

# Collection for BART Stations
stations_collection = db['stations']
station_data = {
    'name': 'Station A',
    'location': {'type': 'Point', 'coordinates': [longitude, latitude]},
    # Other station-specific data
}
stations_collection.insert_one(station_data)

# Collection for Drones
drones_collection = db['drones']
drone_data = {
    'name': 'Drone 001',
    'status': 'idle',  # or "in transit"
    'currentLocation': {'type': 'Point', 'coordinates': [longitude, latitude]},
    # Other drone-specific data
}
drones_collection.insert_one(drone_data)

# Collection for Deliveries
deliveries_collection = db['deliveries']
delivery_data = {
    'orderId': '12345',
    'packageWeight': 1.5,  # in kg
    'packageSize': 'medium',
    'destination': {'type': 'Point', 'coordinates': [longitude, latitude]},
    'status': 'pending',  # or "in progress" or "completed"
    'assignedDrone': 'Drone 001',
    # Other delivery-specific data
}
deliveries_collection.insert_one(delivery_data)
```

Redis - Caching, Geospatial Indexing to monitor delivery

Caching Function for Delivery Business Analysis



- **Caching:** Redis can be used as a cache to store frequently accessed data, such as delivery routes or frequently requested delivery information. Caching can significantly reduce the response times and improve overall system performance.

Geospatial Indexing for tracking drone locations

- **Geospatial Indexing**: Redis has native support for geospatial indexing, which allows the company to store and query spatial data efficiently. This is essential for tracking drone locations, monitoring delivery destinations, and optimizing drone routes based on real-time conditions.

MongoDB Codes: Tracking Delivery Status

```
import redis

redis_client = redis.StrictRedis(host='redis_host', port=your_redis_port,
decode_responses=True)

# Storing delivery status in Redis
def update_delivery_status(order_id, status):
    redis_client.hset('delivery_status', order_id, status)

# Retrieving delivery status from Redis
def get_delivery_status(order_id):
    return redis_client.hget('delivery_status', order_id)

# Example usage:
order_id = '12345'
status = 'in progress'
update_delivery_status(order_id, status)

# Later, retrieve the status:
delivery_status = get_delivery_status(order_id)
print(delivery_status)
```

Q&A