KAUNAS UNIVERSITY OF TECHNOLOGY

Faculty of Informatics
Department of Information Systems

Concurrent Programming

Course Work Report – Individual Project

	Student:	Manonmani Natarajan
	Teacher:	Assoc. prof. pract. Guogis Evaldas,
		Lect. Kiudys Eligijus

Kaunas, 2024

Optimization of store locations using gradient descend with data concurrency

1. Description

The aim of this project is to perform optimization using parallelized gradient descend to minimize the overall construction cost for the new stores. The Location (coordinates) of new stores must be selected in such a way that the cost of building the stores is as low as possible. The optimization of the cost is done using gradient descend with data concurrency tools such as multiprocessing. Pool to speed up the optimization process by utilizing additional processors.

2.Task Analysis

- The city is located in a square with x= (-10,10) and y= (-10,10) coordinates.
- There is minimum n>3 existing stores in a city.
- It's planned to build a minimum of m > 3 new stores in a city.
- The location (coordinates) of new stores must be selected in such a way that the cost of building the stores is as low as possible.
- The cost of building store C is evaluated based on its distance to nearby stores D, which is the distance between the new store location and each existing store. and the city boundary B which means the distance from the new store location to the nearest boundary of the city square.
- The formula to found out the value of CD is given below:
  The distance between the store whose coordinates ($\boldsymbol{x1}, \boldsymbol{y1}$) and ($\boldsymbol{x2}, \boldsymbol{y2}$), the price is calculated according to the formula:

$$C(x_1, y_1, x_2, y_2) = \exp(-0.2 \cdot ((x_1 - x_2)^2 + (y_1 - y_2)^2))$$

In The code the formula is implemented as follow

```
def distance_cost(x1, y1, x2, y2):
    return np.exp(-0.2 * ((x1 - x2)**2 + (y1 - y2)**2))
```

Library used:

❖ Numpy- Numerical python is used to work with arrays

- The formula to found out the value of CB is given below:

  The formula to found out the value of CB is given below:

  The distance between the store whose coordinates ($x1$, $y1$), and the price of the nearest city boundary point with coordinates (xr, yr) is calculated according to the formula:

$$C^R(x_1, y_1, x_r, y_r) = \begin{cases} 0, & \textit{if the store is planned to be built within the city limits} \\ \exp(0.25 \cdot ((x_1 - x_r)^2 + (y_1 - y_r)^2)) - 1, & \textit{other cases} \end{cases}$$

In The code the formula is implemented as follow

```
# Boundary cost function
def boundary_cost(x, y):
     dist_x = min(abs(x - X_coordinate[0]), abs(x - X_coordinate[1]))
    dist_y = min(abs(y - Y_coordinate[0]), abs(y - Y_coordinate[1]))
    dist = min(dist_x, dist_y)
    if dist > 0:
        return 1 / (1 + np.exp(-0.25 * dist**2))
    else:
        100
```

Library used:

❖ Numpy- Numerical python is used to work with arrays

- The total cost is calculated as the sum of the distances to these locations (C = CD + CB).

  In The code the formula is implemented as follow

```
def Total_Price(Existing_stores, New_stores):
    total_cost = 0
    for New_store in New_stores:
        total_distance = 0
        for existing in Existing_stores:
            distance = distance_cost(New_store[0], New_store[1], existing[0],
existing[1])
            total_distance += distance
        distance_to_existing_stores = total_distance / len(Existing_stores)
        distance_to_boundary = boundary_cost(New_store[0], New_store[1])
        total_cost += distance_to_existing_stores + distance_to_boundary
    return total_cost
```

- For estimating **Gradient** numerically central difference formula is used in this code which is used to find the derivative of a function numericallyThe formula is used to to know how much the total cost changes when we move a store a tiny bit in the x or y direction

```
# Gradient calculation
def calculate_gradient(existing_stores, new_stores, j, k):
    small_val = np.zeros_like(new_stores)
    small_val[j, k] = 1e-5
    cost_with_small_val = Total_Price(existing_stores, new_stores + small_val)
    cost_with_negative_small_val = Total_Price(existing_stores, new_stores -
small_val)
    gradient = (cost_with_small_val - cost_with_negative_small_val) / (2 * 1e-
5)#finite difference approximation
    return gradient
```

In the provided code J will be the index of the new store and k will be the x,y coordinate of the store in the jth index.the finite difference algorithm will move gradually to the inexpensive place.while using calculate gradient function in main starmap is used since gradient claculation has multiple arguments. The starmap method applies the calculate_gradient function in parallel to each of these tuples. -

```
gradients = pool.starmap(
                    calculate_gradient,
                    [(existing_stores, new_stores, j, k) for j in
store_indices for k in range(2)]
                )
```

This code in main method ->new_stores -= learning_rate * grad
Will gradually moves the new store to the inexpensive place with slower learning rate and the learning rate will update for every iteration - # Learning rate decay

```
learning_rate *= 0.99
```

-Learning rate decay is used to gradually adjust the learning rate, usually by lowering it, to facilitate the optimization algorithm's more rapid convergence to a better solution.

And also, the usage of process increased the efficiency of the code as it is less time consuming in each iteration than not using process.

3.Testing And Instructions

The submitted task has been developed using **Visual Studio Code** and is structured as follows:

- data_file.py: This file is used to generate the data for both existing stores and new stores. It creates and manages the dataset in Json format it will be stored in /Data file inside the folder.
- main.py: This is the main script containing the source code that solves the problem. It processes the data generated by data_file.py and executes the core logic for store location optimization
- Data: It contains the list of Json file to Use for the project.

4. Performance analysis:

1.The first dataset- "Manonmani_Natarajan_6_new19.json" which consist of 6 existing store and 19 new store inside the dataset. Let's analyze the performance

```
City size: X=[-10, 10], Y=[-10, 10]
Number of existing stores: 6
Number of new stores: 19
Finished optimization with 1 processes in 6.362969398498535 seconds
Objective value after optimization with 1 processes: 16.33720494589894

Finished optimization with 2 processes in 4.586050987243652 seconds
Objective value after optimization with 2 processes: 16.334401437662308

Finished optimization with 3 processes in 4.2848522663116455 seconds
Objective value after optimization with 3 processes: 16.333778953100378

Finished optimization with 4 processes in 3.2326791286468506 seconds
Objective value after optimization with 4 processes: 16.333641019617676

Finished optimization with 5 processes in 3.1600821018218994 seconds
Objective value after optimization with 5 processes: 16.33361046989741

Finished optimization with 6 processes in 4.088791131973267 seconds
Objective value after optimization with 6 processes: 16.33360370440573

Finished optimization with 7 processes in 4.9751927852630615 seconds
Objective value after optimization with 7 processes: 16.33360220616562
```

```
Finished optimization with 8 processes in 5.316037893295288 seconds
Objective value after optimization with 8 processes: 16.333601874377184
Final objective value: 16.333601874377184
```

- **1 process:** 6.36 seconds
- **2 processes:** 4.59 seconds
- **3 processes:** 4.28 seconds
- **4 processes:** 3.23 seconds
- **5 processes:** 3.16 seconds
- **6 processes:** 4.09 seconds
- **7 processes:** 4.98 seconds
- **8 processes:** 5.32 seconds

It seems that using 5 processes yields the fastest time (3.16 seconds), while 6 to 8 processes have a slower performance. This suggests that there might be some issues associated with managing more processes, reducing the efficiency.

2. Objective Value

The objective value after optimization for each number of processes is as follows:

- **1 process:** 16.3372
- **2 processes:** 16.3344
- **3 processes:** 16.3338
- **4 processes:** 16.3336
- **5 processes:** 16.3336
- **6 processes:** 16.3336
- **7 processes:** 16.3336
- **8 processes:** 16.3336

The objective value stabilizes at approximately 16.3336 from 4 processes onward. This indicates that, after reaching 4 processes, further increasing the number of processes does not significantly improve the optimization result.

The final objective value remains the same across all the process configurations: **16.33**. This means that the number of processes does not impact the quality or result of the optimization; it only affects the time taken for convergence.

2.The second dataset- "Manonmani_Natarajan_3_new6.json" which consist of 3 existing store and 6 new store inside the dataset. Let's analyze the performance

```
City size: X=[-10, 10], Y=[-10, 10]
Number of existing stores: 3
Number of new stores: 6

Finished optimization with 1 processes in 1.8801276683807373 seconds
Objective value after optimization with 1 processes: 5.3264868940896895

Finished optimization with 2 processes in 1.793483018875122 seconds
Objective value after optimization with 2 processes: 5.32524238433094

Finished optimization with 3 processes in 2.169712781906128 seconds
Objective value after optimization with 3 processes: 5.32497166998416
Finished optimization with 4 processes in 2.3480007648468018 seconds
Objective value after optimization with 4 processes: 5.324911955879401

Finished optimization with 5 processes in 2.5534791946411133 seconds
Objective value after optimization with 5 processes: 5.324898743624098

Finished optimization with 6 processes in 2.7392446994781494 seconds
Objective value after optimization with 6 processes: 5.324895818312073

Finished optimization with 7 processes in 2.7837681770324707 seconds
Objective value after optimization with 7 processes: 5.324895170524226

Finished optimization with 8 processes in 2.2380406856536865 seconds
Objective value after optimization with 8 processes: 5.324895027071809
Optimization complete.
Final objective value: 5.324895027071809
```

- **1 process**: 1.88 seconds
- **2 processes**: 1.79 seconds
- **3 processes**: 2.17 seconds
- **4 processes**: 2.35 seconds
- **5 processes**: 2.55 seconds
- **6 processes**: 2.74 seconds
- **7 processes**: 2.78 seconds
- **8 processes**: 2.24 seconds

It seems that using 2 processes yields the fastest time (1.79 seconds), while began to increase as more processes were added. . This suggests that there might be some issues associated with managing more processes, reducing the efficiency.

2. Objective Value

The objective value after optimization for each number of processes is as follows:

- **1 process**: 5.326
- **2 processes**: 5.325
- **3 processes**: 5.3249
- **4 processes**: 5.3249
- **5 processes**: 5.3249
- **6 processes**: 5.3249
- **7 processes**: 5.3249
- **8 processes**: 5.3249

The objective value stabilizes at approximately 5.3249 from 3 processes onward. This indicates that, after reaching 3 processes, further increasing the number of processes does not significantly improve the optimization result.

The final objective value remains the same across all the process configurations: **5.3249.** This means that the number of processes does not impact the quality or result of the optimization; it only affects the time taken for convergence.

3.The Third dataset- "Manonmani_Natarajan_5_new50.json" which consist of 5 existing store and 50 new store inside the dataset. Let's analyze the performance

City size: X=[-10, 10], Y=[-10, 10]
Number of existing stores: 5
Number of new stores: 50

```
 Finished optimization with 1 processes in 164.968585729599 seconds
Objective value after optimization with 1 processes: 44.15132362503392

Finished optimization with 2 processes in 56.43791222572327 seconds
Objective value after optimization with 2 processes: 44.13817412291976

 Finished optimization with 3 processes in 12.830161333084106 seconds
Objective value after optimization with 3 processes: 44.13530862472931

Finished optimization with 4 processes in 11.934938907623291 seconds
 Objective value after optimization with 4 processes: 44.134676331693804

Finished optimization with 5 processes in 18.056466102600098 seconds
```

```
Objective value after optimization with 5 processes: 44.13453642087525

 Finished optimization with 6 processes in 16.86247682571411 seconds
Objective value after optimization with 6 processes: 44.1345054428471

Finished optimization with 7 processes in 15.27865982055664 seconds
Objective value after optimization with 7 processes: 44.13449858297577

 Finished optimization with 8 processes in 15.646395921707153 seconds
Objective value after optimization with 8 processes: 44.13449706385818
Optimization complete. Final objective value: 44.13449706385818
```

- **1 process**: 164.97 seconds.
- **2 processes**: 56.44 seconds
- **3 processes**: 12.83 seconds.
- **4 processes**: 11.93 seconds.
- **5 processes**: 18.06 seconds
- **6 processes**: 16.86 seconds.
- **7 processes**: 15.28 seconds.
- **8 processes**: 15.65 seconds

It seems that using 4 processes yields the fastest time (11.93 seconds.), while began to increase as more processes were added. Even though after process 7 it starts to decrease but again in process 8 the time increased. This suggests that there might be some issues associated with managing more processes, reducing the efficiency.

2. Objective Value

The objective value after optimization for each number of processes is as follows:

- **1 process**: 44.15
- **2 processes**: 44.14
- **3 processes**: 44.13
- **4 processes**: 44.13
- **5 processes**: 44.134
- **6 processes**: 44.134
- **7 processes**: 44.134
- **8 processes**: 44.134

The objective value stabilizes at approximately 44.13 from 3 processes onward. This indicates that, after reaching 3 processes, further increasing the number of processes does not significantly improve the optimization result. And also after process 4 the time starts increasing so for this dataset it is optimal to have process 4

The final objective value remains the same across all the process configurations: 44.1344**.** This means that the number of processes does not impact the quality or result of the optimization; it only affects the time taken for convergence.

4.The Fourth dataset- "Manonmani_Natarajan_4_new200.json" which consists of 4 existing stores and 200 new stores inside the dataset this is the biggest dataset among all datasets. Let's analyze the performance

City size: X= [-10, 10], Y=[-10, 10]
Number of existing stores: 4
Number of new stores: 200

```
Finished optimization with 1 process in 503.8506157398224 seconds
Objective value after optimization with 1 process: 168.82718776509913

Finished optimization with 2 processes in 281.2114086151123 seconds
Objective value after optimization with 2 processes: 168.78921675142914
Finished optimization with 3 processes in 155.4302477836609 seconds
Objective value after optimization with 3 processes: 168.78081139363917

Finished optimization with 4 processes in 156.99109053611755 seconds
Objective value after optimization with 4 processes: 168.77895022066636

Finished optimization with 5 processes in 159.09712767601013 seconds
Objective value after optimization with 5 processes: 168.77853807127667

Finished optimization with 6 processes in 168.0297989845276 seconds
Objective value after optimization with 6 processes: 168.7784468005873

Finished optimization with 7 processes in 133.50091814994812 seconds
Objective value after optimization with 7 processes: 168.7784265885561

Finished optimization with 8 processes in 143.14665913581848 seconds
Objective value after optimization with 8 processes: 168.77842211256691
Final objective value: 168.77842211256691
```

- **1 process**: 503.85 seconds

- **2 processes**: 281.21 seconds
- **3 processes**: 155.43 seconds
- **4 processes**: 156.99 seconds
- **5 processes**: 159.10 seconds
- **6 processes**: 168.03 seconds
- **7 processes**: 133.50 seconds
- **8 processes**: 143.15 seconds

The optimization time decreases as the number of processes increases, up to 3 processes. After 3 processes, the time stabilizes around **155-168 seconds**. The best performance (i.e., shortest time) was achieved with **3 processes** (155.43 seconds), while **7 processes** also performed well with 133.50 seconds, though the time difference between 3 and 7 processes is not significant.

2. Objective Value

The objective value after optimization for each number of processes is as follows:

- **1 process**: 168.827
- **2 processes**: 168.789
- **3 processes**: 168.780
- **4 processes**: 168.779
- **5 processes**: 168.779
- **6 processes**: 168.778
- **7 processes**: 168.778
- **8 processes**: 168.778

The objective value decreases slightly as the number of processes increases. However, the change is minimal after **3 processes.** The **final objective value** of **168.778** is very close across the different configurations, indicating that adding more processes after 3 does not significantly improve the optimization results in terms of cost (objective value).

5.The Fifth dataset- "Manonmani_Natarajan_19_new100.json" which consists of 19 existing stores and 100 new stores inside the dataset this is the second biggest dataset among all datasets. Let's analyze the performance

City size: X=[-10, 10], Y=[-10, 10]
Number of existing stores: 19
Number of new stores: 100

```
Finished optimization with 1 processes in 315.15213799476624 seconds
Objective value after optimization with 1 processes: 84.78185901495411

Finished optimization with 2 processes in 166.19677662849426 seconds
Objective value after optimization with 2 processes: 84.76701363327344

Finished optimization with 3 processes in 132.4055187702179 seconds
Objective value after optimization with 3 processes: 84.76371774204195

Finished optimization with 4 processes in 117.32468342781067 seconds
Objective value after optimization with 4 processes: 84.76298746353196

Finished optimization with 5 processes in 112.1694974899292 seconds
Objective value after optimization with 5 processes: 84.76282572268997

Finished optimization with 6 processes in 112.01786756515503 seconds
Objective value after optimization with 6 processes: 84.76278990394354

Finished optimization with 7 processes in 113.33680438995361 seconds
Objective value after optimization with 7 processes: 84.76278197177162

Finished optimization with 8 processes in 121.23138308525085 seconds
Objective value after optimization with 8 processes: 84.76278021517572
Final objective value: 84.76278021517572
```

- **1 process**: 315.15 seconds
- **2 processes**: 166.20 seconds (a reduction of ~47%)
- **3 processes**: 132.41 seconds (a reduction of ~20%)
- **4 processes**: 117.32 seconds (a reduction of ~11%)
- **5 processes**: 112.17 seconds (a reduction of ~4%)
- **6 processes**: 112.02 seconds (no significant change from 5 processes)
- **7 processes**: 113.34 seconds (a slight increase compared to 6 processes)
- **8 processes**: 121.23 seconds (a slight increase compared to 7 processes)

The objective value converges to **84.7628** after a certain point (around 4 processes), indicating that increasing the number of processes further doesn't significantly improve the optimization result. This suggests that the optimization process has reached a plateau in terms of minimizing the objective value.

2. Objective Value

The objective value after optimization for each number of processes is as follows:

- **1 process**: 84.78
- **2 processes**: 84.77
- **3 processes**: 84.76
- **4 processes**: 84.76
- **5 processes**: 84.76
- **6 processes**: 84.76
- **7 processes**: 84.76
- **8 processes**: 84.76

The runtime decreases as more processes are used, with a significant reduction when moving from 1 to 2 processes. However, the runtime starts to stabilize between 5 and 6 processes, indicating that further parallelization beyond 5 processes does not provide significant performance improvements and might even cause slight overhead (e.g., 8 processes takes a bit longer than 6).

6.The Sixth dataset- "Manonmani_Natarajan_14_new19.json" which consist of 14 existing store and 19 new store inside the dataset. Let's analyze the performance

```
Reading data from Manonmani_Natarajan_14_new19.json...
Data successfully loaded.
City size: X=[-10, 10], Y=[-10, 10]
Number of existing stores: 14
Number of new stores: 19

Finished optimization with 1 processes in 8.603652238845825 seconds
Objective value after optimization with 1 processes: 16.99450488109223

Finished optimization with 2 processes in 6.28527045249939 seconds
Objective value after optimization with 2 processes: 16.994834762580798

Finished optimization with 3 processes in 6.851625680923462 seconds
Objective value after optimization with 3 processes: 16.994893859627126

Finished optimization with 4 processes in 7.091951131820679 seconds
Objective value after optimization with 4 processes: 16.994906236618636
```

```
Finished optimization with 5 processes in 6.903446674346924 seconds
Objective value after optimization with 5 processes: 16.99490894242156

Finished optimization with 6 processes in 7.187093257904053 seconds
Objective value after optimization with 6 processes: 16.994909539901922

Finished optimization with 7 processes in 7.772363901138306 seconds
Objective value after optimization with 7 processes: 16.99490967213041

Finished optimization with 8 processes in 7.981548309326172 seconds
Objective value after optimization with 8 processes: 16.994909701408492
Final objective value: 16.994909701408492
```

- **1 Process:** 8.60 seconds
- **2 Processes:** 6.29 seconds
- **3 Processes:** 6.85 seconds
- **4 Processes:** 7.09 seconds
- **5 Processes:** 6.90 seconds
- **6 Processes:** 7.19 seconds
- **7 Processes:** 7.77 seconds
- **8 Processes:** 7.98 seconds

The runtime doesn't decrease significantly after 2 processes, indicating that the parallelization benefits start diminishing after a certain point. This may be due to factors like overhead from managing multiple processes or the size of the dataset being relatively small..

2. Objective Value

The objective value after optimization for each number of processes is as follows:

- **1 Process:** 16.99450488109223
- **2 Processes:** 16.994834762580798
- **3 Processes:** 16.994893859627126
- **4 Processes:** 16.994906236618636
- **5 Processes:** 16.99490894242156
- **6 Processes:** 16.994909539901922
- **7 Processes:** 16.99490967213041
- **8 Processes:** 16.994909701408492

The slight improvement in the objective value with more processes suggests that the optimization has already converged to a very stable solution. Hence, the parallelization has a minimal effect on improving the final outcome, but it does help in speeding up the computations.

Dataset	No. of Existing Stores	No. of New Stores	Fastest Time (s)	Optimal Process Count	Final Objective Value
Manonmani_Natarajan_6_new19.json	6	19	3.16 (5 processes)	5 processes	16.3336
Manonmani_Natarajan_3_new6.json	3	6	1.79 (2 processes)	2 processes	5.3249
Manonmani_Natarajan_5_new50.json	5	50	11.93 (4 processes)	4 processes	44.1344
Manonmani_Natarajan_4_new200.json	4	200	133.50 (7 processes)	3 processes	168.778
Manonmani_Natarajan_19_new100.json	19	100	112.03	4	84.7628

5.Conclusion

The performance analysis of different datasets with varying process counts shows that optimization time improves with the addition of processes initially but plateaus or worsens beyond a certain point. The objective value remains stable across process configurations, indicating it isn't significantly affected by process count, but optimization time is.

For smaller datasets like "Manonmani_Natarajan_6_new19.json" and "Manonmani_Natarajan_3_new6.json," the time decreased most effectively with 5 and 2 processes, respectively. Adding more processes led to slower performance, likely due to overhead.

For larger datasets like "Manonmani_Natarajan_5_new50.json," 4 processes provided the best performance, and for the largest dataset "Manonmani_Natarajan_4_new200.json," 3 processes were most efficient.

In summary, the most efficient optimization time generally occurs with 3 to 5 processes, depending on dataset size. Adding more processes doesn't always improve results and can cause unnecessary overhead, but an optimal number can speed up convergence.

6.Reference

1. https://www.geeksforgeeks.org/reading-and-writing-json-to-a-file-in-python/

2. https://www.geeksforgeeks.org/numpy-gradient-descent-optimizer-of-neural-networks/
3. https://docs.python.org/3/library/multiprocessing.html
4. https://www.youtube.com/watch?v=raEYl25dgVo