Balanced Clustering

(2024학년도 겨울학기 Logistics Lab 연구참여)

2025.02.18.

지도 교수: 김병인
지도 사수: 한상일
지도 학생: 정현우

차례

1. 서론

클러스터링(Clustering)은 유사한 특성을 가진 데이터를 그룹으로 묶는 과정으로, 동일한 그룹 내 데이터 간 유사성을 최대한 확보하면서 이질적인 데이터는 별도의 그룹으로 분류하는 것이 핵심이다. 예를 들어, 2차원 좌표 데이터의 경우 유클리드 거리가 가까운 데이터들을 하나의 그룹으로 묶을 수 있다.

클러스터링의 확장된 문제 중 하나로 용량 제한 클러스터링(Capacitated Clustering, Balanced Clustering)이 있다. 문제에서 각 데이터는 특정 수요(demand)를 가지며, 각 클러스터는 정해진 용량(capacity) 제한을 초과하지 않아야 한다. Balanced Clustering을 해결하기 위한 다양한 알고리즘이 기존 연구에서 제안되었으며, 대표적으로 Priority Measure, Neighborhood Search, Neural Network 등의 방법이 있다.

본 연구에서는 사전에 클러스터링된 데이터를 재배치하여 용량 제한을 충족하는 알고리즘을 설계하는 것을 목표로 한다. 실험 데이터는 random 모듈을 이용해 생성된 2차원 좌표 데이터이며, 각 데이터는 일정 범위 내 정수 값을 수요로 갖는다. 모든 클러스터는 동일한 용량 제한을 가지며, 초기 클러스터링은 k-means 알고리즘을 활용해 수행된다. 그러나 k-means 결과는 용량 제한을 보장하지 않기에 이후 Balanced Clustering 알고리즘을 적용하여 이를 조정한다.

본 문제의 formulation은 다음과 같다.

$$Coordinates : ((x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n))$$
$$Demands : (d_1, d_2, d_3, ..., d_n)$$
$$Capacity : C$$

$$x_{ij} = \begin{cases} 1 & \text{if requester i is assigned to cluster j} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Minimize} \quad \sum_{j=1}^{k} \sum_{i=1}^{n} cost_{ij} * x_{ij}$$

$$Subject\ to$$
$$\sum_{j=1}^{k} x_{ij} = 1, i = 1, 2, ..., n$$
$$\sum_{i=1}^{n} d_i x_{ij} \leq C, j = 1, 2, ..., k$$

그림 1. Balanced Clustering formulation

2. 이론적 배경

2.1 K-means

대표적인 클러스터링 알고리즘으로, 라이브러리로 구현이 쉽기에 초기에 군집화된 데이터를 생성하기에 적합하다. Initialization - Assignment - Update - Iteration의 단계를 거치며, 본 연구에서는 scikit-library를 이용하여 구현하였다. 실행 결과를 시각화하면 아래와 같다.

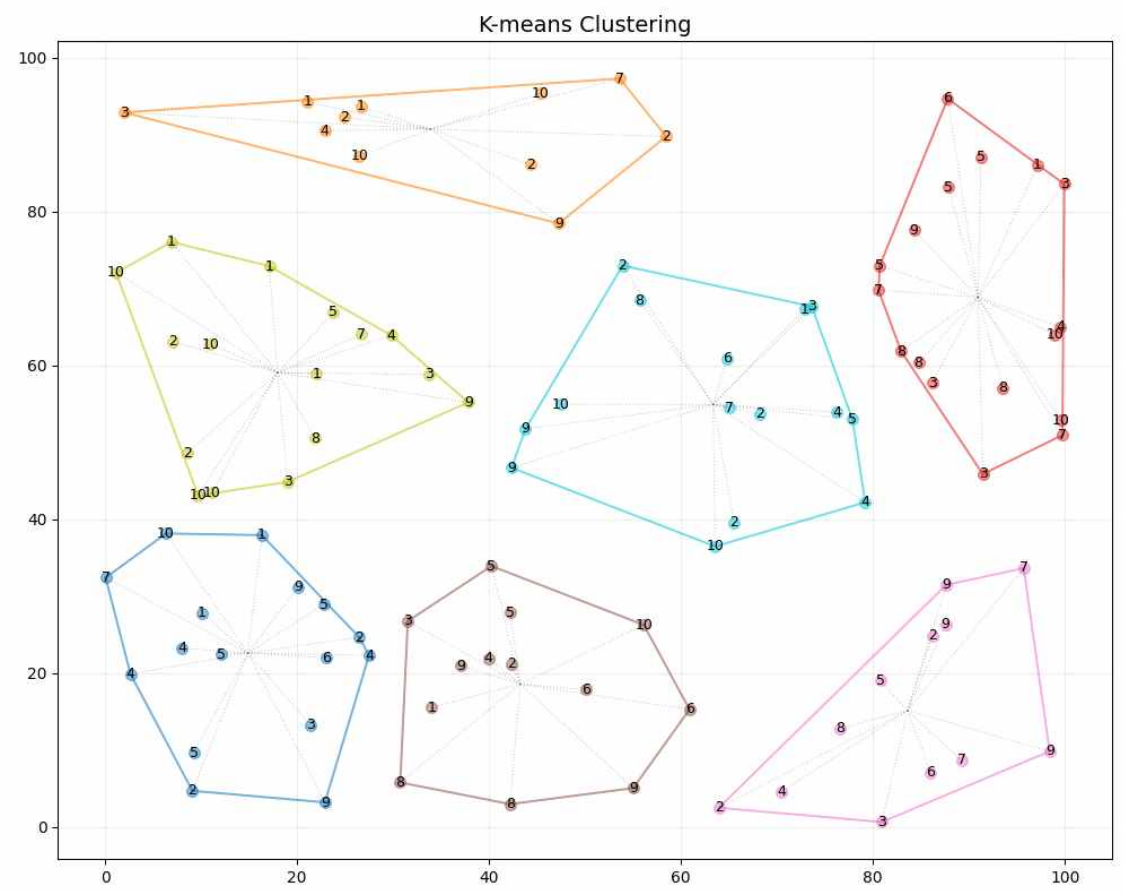

그림 2. K-means 클러스터링 결과

2.2 Silhouette

실루엣 계수(Silhouette Coefficient)는 클러스터링 결과의 품질을 평가하는 지표로, 각 데이터 포인트가 자신이 속한 클러스터 내의 데이터와 얼마나 가깝고, 다른 군집의 데이터와는 얼마나 먼지 나타낸다. 실루엣 계수는 -1에서 1 사이의 값을 가지며, 1에 가까울수록 해당 데이터가 현재 속한 클러스터에 잘 속해 있음을, -1에 가까울수록 잘못된 클러스터에 속해 있음을 나타낸다. 실루엣 계수는 클러스터 내의 평균 거리 a(i)와 가장 가까운 다른 군집과의 평균 거리 b(i)를 이용해 구할 수 있으며, 식은 다음과 같다.

$a(i) = \text{average dissimilarity of } i \text{ to all other objects of } A.$
$d(i, C) = \text{average dissimilarity of } i \text{ to all objects of } C.$
$b(i) = \min_{C \neq A} d(i, C).$

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

$$-1 \leq s(i) \leq 1$$

그림 3. 실루엣 계수

실루엣 계수는 특정 데이터가 적절한 클러스터에 배치되었는지를 평가하는 지표로, 본 연구에서는 재배치할 데이터의 선별 기준으로 활용하였다.

3. 문제 해결 알고리즘

3.1 Recursive Neighborhood Search

첫 번째 알고리즘은 재귀함수를 이용하여 데이터를 반복적으로 재배치하는 방식이다. Algorithm 1에서 이를 자세히 설명하고 있는데, 여기서 X는 각 데이터의 좌표, D는 각 데이터의 수요, C는 클러스터별 용량 제한을 의미한다. 주어진 X, D, C를 바탕으로 K-means 알고리즘은 빠르게 군집화를 수행한다. 군집화가 완료된 후, 각 데이터에 대해 가장 가까운 다른 클러스터를 나타내는 SecondLabels를 계산한다. 이후 재배치 대상이 되는 데이터는 이 SecondLabels을 기준으로 새로운 클러스터로 이동하게 된다.

재배치 대상이 되는 기준은 Silhouette 값을 활용한다. Silhouette 값이 낮을수록 현재 속한 클러스터와의 적합도가 낮다는 의미이며, 이는 곧 해당 데이터를 다른 클러스터로 이동시키는 것이 클러스터링의 품질을 향상할 가능성이 높다는 것을 시사한다. Silhouette은 데이터를 이동할 때마다 재계산한다.

구체적인 이동 과정은 다음과 같다. 클러스터 중 수요 합이 가장 큰 클러스터를 선택, 그 안에서 실루엣이 가장 작은 데이터를 선택한다. 선택한 데이터를 SecondLabels에 해당하는 클러스터로 옮긴다. (SecondLabels에 해당하는 클러스터와 Labels에 해당하는 클러스터를 교환; 데이터가 다시 선택되는 경우 기존의 클러스터로 옮기기 위함) 이때 데이터를 넘겨받은 클러스터의 수요 합이 용량 제한을 넘지 않는다면 Recursive call 중단. 만약 넘는다면 Recursive하게 해당 클러스터에서 실루엣이 가장 작은 데이터를 선택, 다른 클러스터로 넘겨준다. 모든 클러스터가 용량 제한을 만족할 때까지 이와 같은 이동을 반복한다.

그러나 실루엣을 기준으로 재배치할 데이터를 선택할 경우 특정 데이터가 반복적으로 선택되는 사례가 발생할 수 있다. 이를 방지하기 위해 friction과 tabu를 도입하였다. friction

은 데이터의 개수와 동일한 길이의 리스트 형태로 저장되며, 특정 데이터가 재배치 대상으로 선택되면 해당 데이터의 friction이 0.1 증가한다. 앞서 설명했듯이 재배치 대상으로 선택하는 기준은 실루엣으로, 실루엣이 낮은 데이터가 우선하여 선택된다. 이때 실루엣만을 기준으로 하는 것이 아닌, 실루엣 + friction을 기준으로 하여 데이터를 선택하면 여러 번 선택된 데이터는 다시 선택될 확률이 줄어든다. 즉 friction은 데이터의 선택 횟수를 반영하는 일종의 가중치이다. tabu는 클러스터링에 적용할 수 있는 여러 메타휴리스틱 중 tabu-search에서 아이디어를 가져왔다. tabu는 queue 자료구조로, 길이가 고정되어 있다. 데이터가 재배치 대상으로 선택되면 해당 데이터는 tabu에 삽입되며, tabu가 가득 차 있을 때 삽입이 이루어지면 가장 먼저 들어왔던 데이터가 삭제된다. tabu는 금지 목록의 역할을 한다. tabu에 들어있는 데이터는 재배치 대상으로 선택될 수 없다. 현재 실루엣이 가장 작은 데이터가 tabu에 들어있다면, 그 다음으로 실루엣이 작은 데이터를 선택하는 방식이다. friction과 tabu를 통해 동일한 데이터의 반복적인 선택을 제한하고, 다양한 방향으로 해를 탐색하게 한다. 다만 tabu(queue)의 길이에 따라 클러스터링의 성능이 달라지는 모습을 확인할 수 있었다. 길이에 따라 클러스터링의 품질뿐만 아니라 feasible한 해에 도달하는지의 여부 역시 달라졌기에, 실제 코드에선 여러 종류의 tabu 길이로 테스트하고, 그중 가장 좋은 품질의 클러스터링 결과를 택했다.

Algorithm 1: Balanced Clustering with Recursive Shift

```
function Initialize(X, D, C):
    K ← ⌈(ΣD) / C⌉
    (L, centroids) ← K-Means(X, K)
    ComputeSecondLabels(X, L)
    ComputeSilhouette(X, L)
    friction[i] ← 0, tabu ← ∅

function RecursiveShift():
    while max(demandSum(k)) > C:
        giver ← argmax(demandSum(k))
        Shift(giver)

function Shift(k):
    if demandSum(k) ≤ C:
        return
    d ← argmin(sil[i]) for i in cluster k
    Move(d → secondLabels[d])
    friction[d] += Δ
    Update silhouette coefficients
    Shift(labels[d])     // Recursive call
```

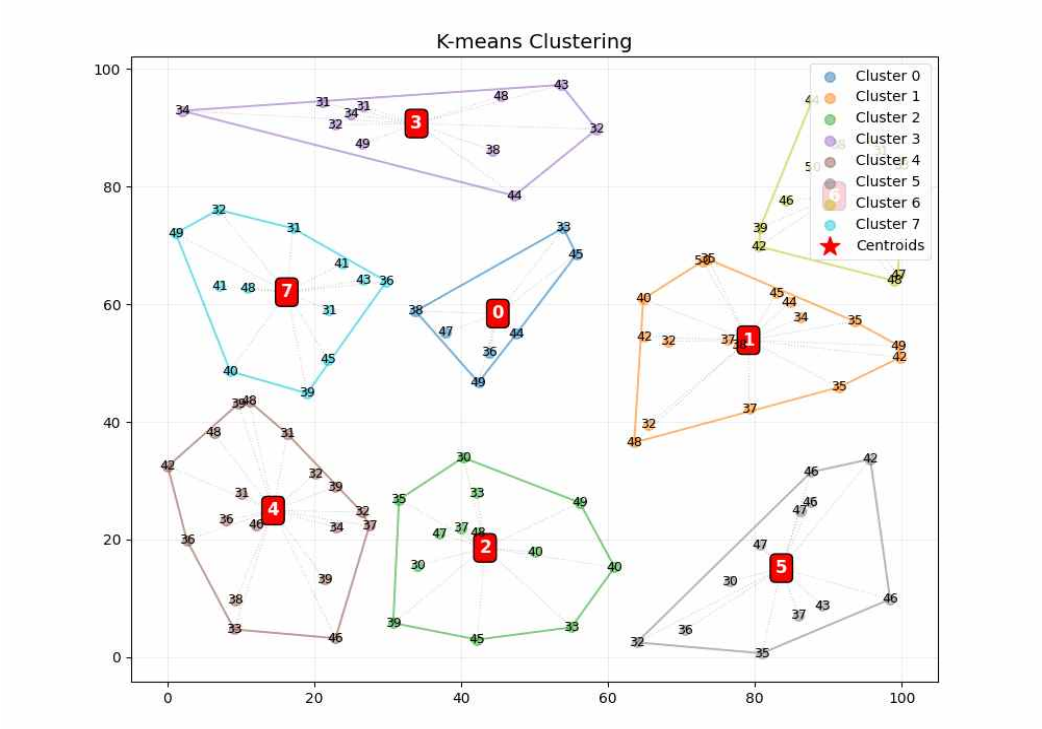

그림 4. samples 100, capacity 500, demand range (30,50)의 dataset

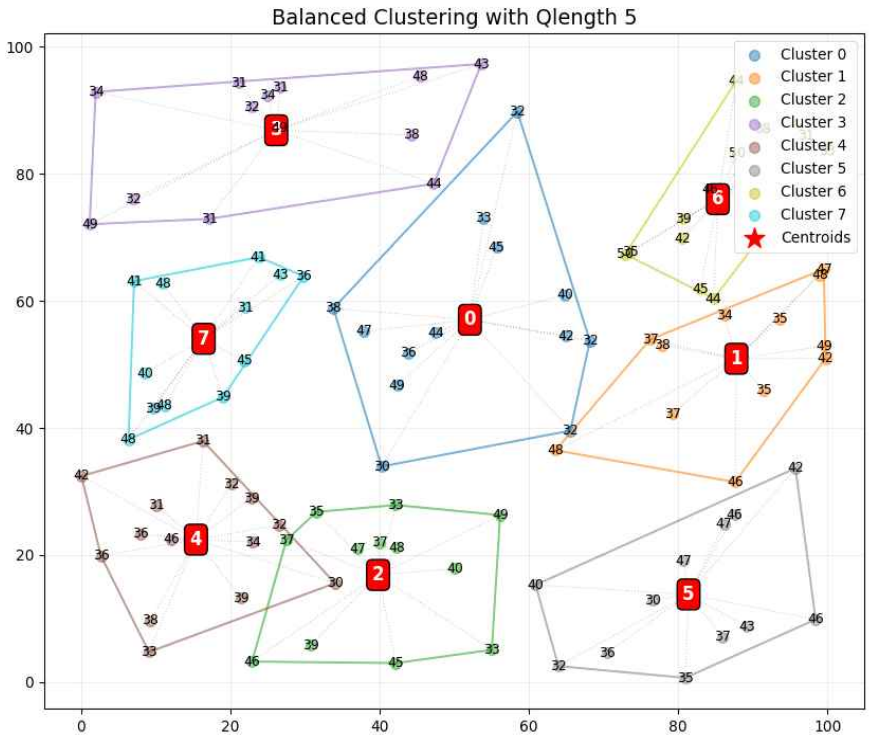

그림 5. Recursive algorithm 적용 결과 (Optimal Tabu Length: 5)

3.2 Network-Based Approach

두 번째 알고리즘의 초기화 방식은 첫 번째와 동일하다. K-means 방식으로 초기화한 이후 용량 제한을 맞추기 위해 데이터를 재배치하는 방식이다. 그러나 첫 번째 알고리즘은 동적으로 재배치할 데이터를 선택하는 반면, 두 번째 알고리즘은 사전에 클러스터 간의 이웃 관계를 네트워크 형식으로 정의한 이후, 네트워크에서 찾아낸 최단 경로를 바탕으로 데이터를 재배치하는 방식이다.

클러스터 네트워크를 조직하는 방식은 다음과 같다. 우선 클러스터 간의 이웃 관계를 정의한다. 클러스터끼리 이웃 관계에 있다는 것은 데이터를 주고 받기에 충분히 가깝다는 의미이며, 이를 판단하기 위해서는 ConvexHull을 활용한다. ConvexHull은 주어진 꼭짓점들을 모두 감싸는 볼록다각형을 반환한다. 두 가지 클러스터가 있을 때, 두 클러스터에 속한 모든 데이터를 감싸는 ConvexHull을 계산한다. 계산한 Hull 내부에 외부의 점들이 포함되지 않는다면 두 클러스터를 이웃 관계로 정의, 포함된다면 이웃 관계가 아니다.

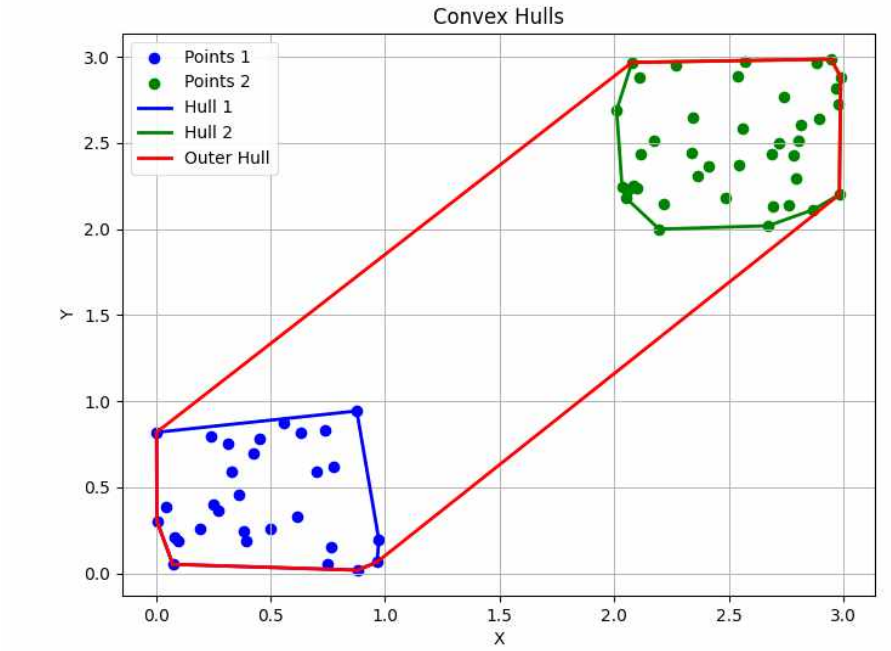

그림 6. ConvexHull을 통해 정의한 이웃 관계의 클러스터

그러나 해당 기준으로만 이웃 관계를 정의할 경우 발생하는 문제는 두 클러스터가 충분히 가까운 거리에 있지만 외부 점들이 포함되는 경우, 외부 점들이 포함되지 않는 hull을 생성할 수 있지만 거리가 먼 경우 등 실질적인 이웃 관계를 판단하기 어렵다는 것이다. 따라서 두 클러스터 간의 거리를 계산한 이후 거리가 기준보다 작으면 ConvexHull과 무관하게 이웃 관계로 확정 짓고, 기준보다 멀다면 이웃 관계에서 배제함으로써 실질적인 이웃 관계를 탐색한다. 이때 '클러스터 간의 거리'를 측정하는 방법은 다양하게 존재하지만, 본 연구에서 선택한 척도는 가장 가까운 데이터 간의 거리인 'Single Linkage'와 가장 먼 데이터 간의 거리인 'Complete Linkage'이다.

모든 클러스터 간의 이웃 관계를 파악한 이후에는 Node-Edge로 구성된 그래프로 전체 네트워크를 표현한다. 이웃 관계에 놓인 점들은 간선으로 연결하고, 간선의 가중치는 앞서 계산한 Single Linkage, Complete Linkage와 같은 클러스터 거리이다. 각 클러스터가 곧 노드가 되고, 노드의 위치는 임의로 각 클러스터 중심의 위치와 동일시한다. 결과적으로 생성된 클러스터 네트워크는 아래와 같다.

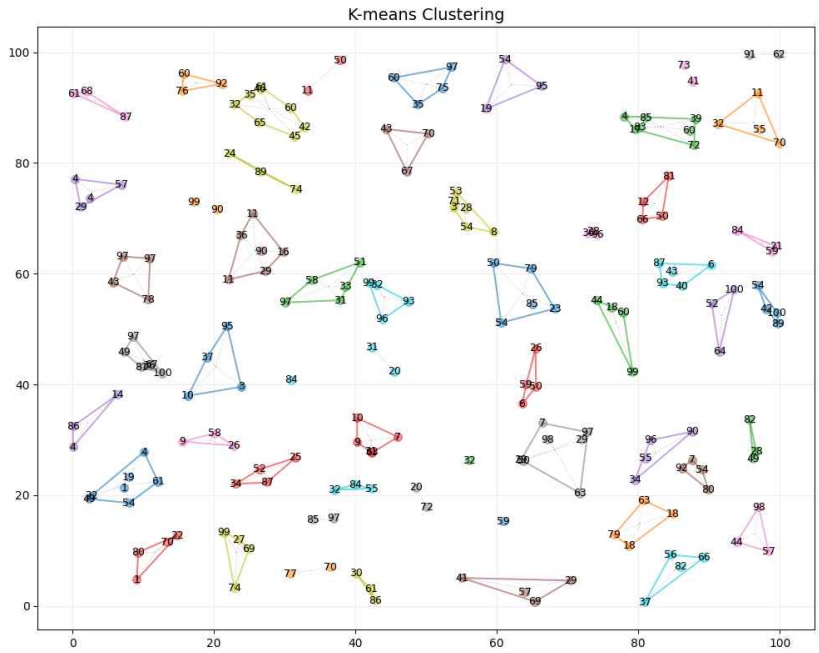

그림 7. 네트워크를 파악하기 이전의 dataset

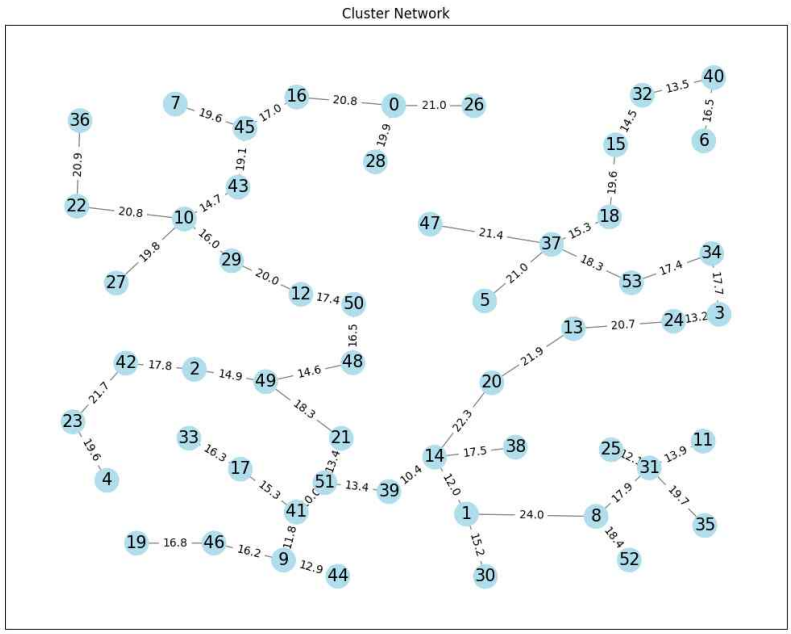

그림 8. 클러스터 네트워크

파악한 클러스터 네트워크를 바탕으로 데이터를 재배치하기 위한 경로를 추적한다. 출발점은 수요 합이 가장 큰 클러스터, 도착점은 수요 합이 가장 작은 클러스터로 선택한다. 출발점에서 도착점까지의 거리는 Dijkstra's algorithm를 통해 최단 거리를 선택한다. 이로써 선택된 경로가 2-15-6-7-10이라면 (번호는 클러스터 번호) 2번 클러스터에서 15번 클러스터로, 15번 클러스터에서 6번 클러스터로 데이터를 옮기는 방식이다. 이때 옮길 데이터는 각 쌍에서 실루엣이 가장 낮은 데이터이다. 이를테면 2번에서 15번으로 데이터를 옮긴다면 2, 15번 클러스터에 속한 데이터만을 가지고 실루엣을 계산한 다음, 2번 클러스터에서 가장 실루엣이 작은 데이터를 15번 클러스터로 옮긴다. 정리하자면, 네트워크를 통해 최단 경로를 파악한 후, 경로에 놓인 클러스터 간에 데이터를 옮김으로써 용량을 조절한다.

경로를 따라 데이터를 재배치했다면 그와 동시에 클러스터 관계도 변화한다. 기존의 이웃 관계가 깨질 수도 있고, 클러스터 간의 거리 역시 변화한다. 이를 반영하기 위해 한 번의 재배치가 이루어질 때마다 경로에 놓인 클러스터 그리고 이와 이웃한 클러스터를 대상으로 네트워크 정보를 수정한다. 클러스터 경로, 이와 함께 변화한 네트워크를 시각화하면 아래와 같다.

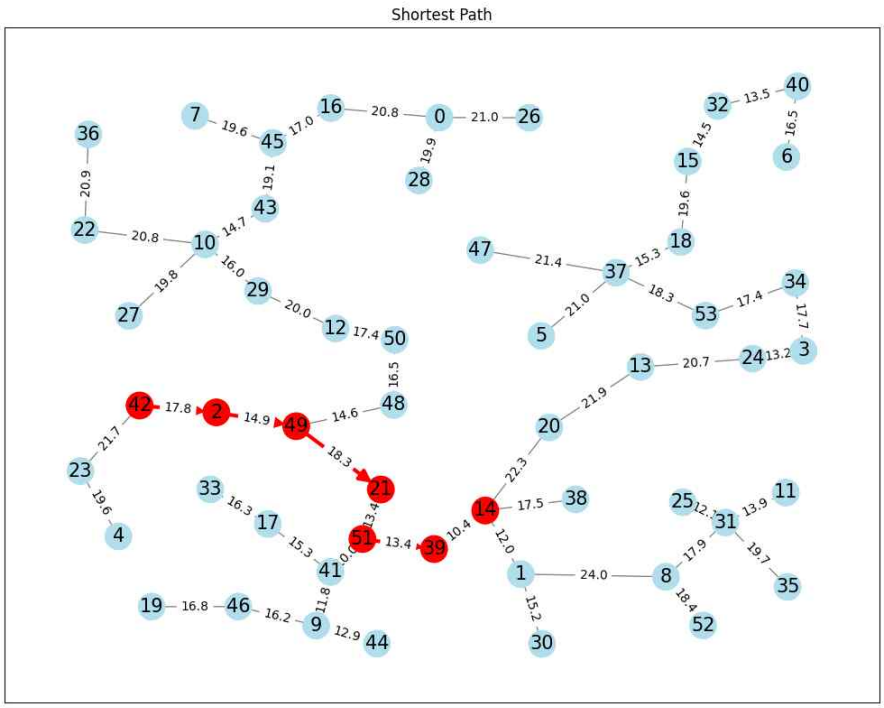

그림 9. Dijkstra 알고리즘을 통해 계산한 클러스터 경로

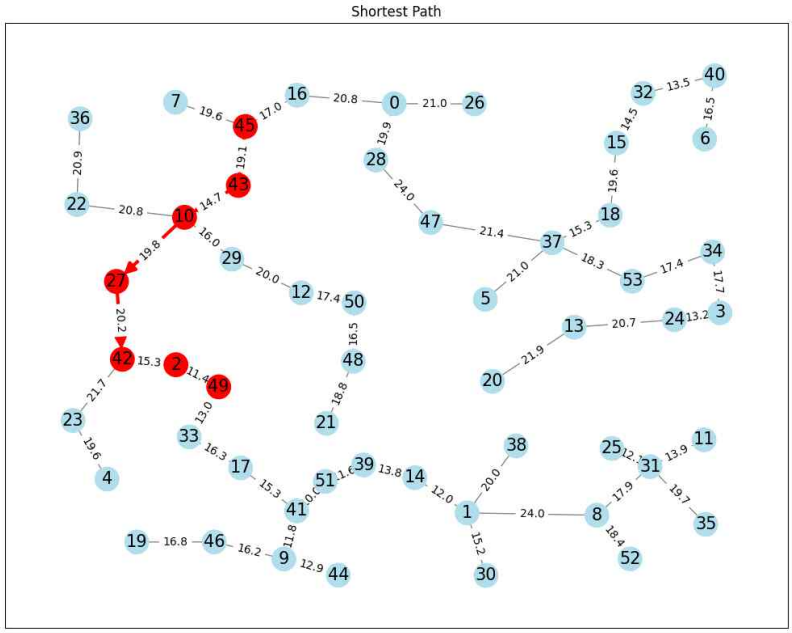

그림 10, 수정된 네트워크와 새로운 경로

알고리즘의 종료 조건은 1번 알고리즘과 마찬가지로 모든 클러스터가 용량 제한을 만족했을 경우이다. 그러나 두 알고리즘 모두 항상 해에 도달하는 것은 아니다. 특정 시점이 되면 똑같은 해를 맴도는 일종의 loop에 빠지게 된다. Loop에 빠지는 이유는 여러 가지일 수 있다. 해의 탐색 방향이 부족해서, 애초에 설정한 클러스터 수가 부족해서일 수도 있다. 이유가 어떻든 해를 찾는데 실패했다면 dummy cluster, 즉 새로운 클러스터를 추가함으로써 해를 찾는다. 계속해서 클러스터를 늘려가다 해를 찾는 시점에 알고리즘은 종료한다.

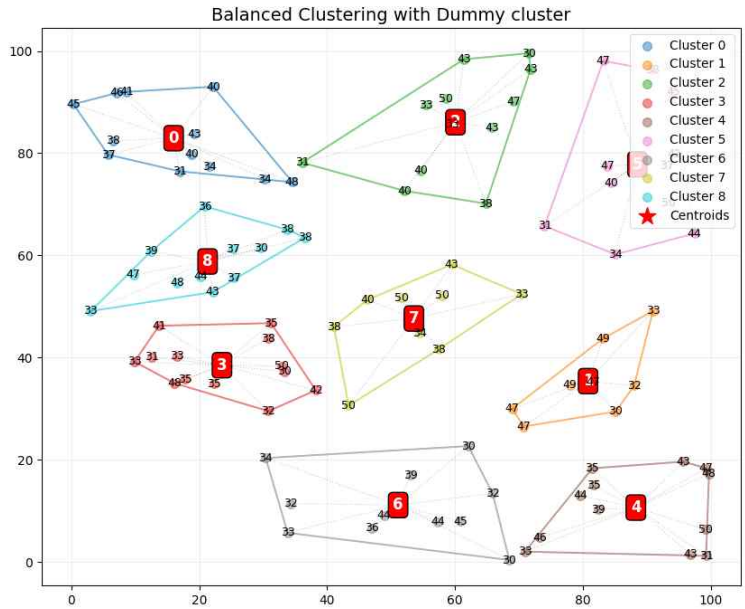

그림 11. Dummy Cluster를 추가함으로써 해를 찾은 경우

Algorithm 2: Balanced Clustering with Cluster network

```
function Initialize(X, demand, capacity):
    Perform K-Means clustering with appropriate number of clusters
    Compute silhouette scores
    Initialize tabu list and cluster network
function RunAlgorithm():
    While clusters do not satisfy capacity:
        Execute transportation algorithm to balance clusters
        If balanced, display results and exit
        Else, add dummy clusters and re-run K-Means
function TransportationAlgorithm():
    While capacity constraints are not met:
        Select source and target clusters based on demand
        Avoid source clusters using tabu search
        Move data along the shortest path
        Recompute centroids and update network
        If no change (cycle detected), add dummy cluster
function UpdatePartialNetwork(affected_clusters):
    Recalculate edges for affected clusters
    Ensure graph connectivity
function TransportData(source, target):
    Find shortest path and move data between clusters
    Update labels, cluster sizes, and demand sums
```

4. 실행 결과

알고리즘의 성능을 평가하기 위해 다양한 Dataset을 만들어 파일로 저장한 후, 성능을 평가하였다. 총 50개의 dataset이 존재하며, 5개의 그룹으로 이루어져 있다. 하나의 그룹에 속하는 dataset은 sample 수, demand range, capacity가 동일하다. dataset의 각 수치는 다음과 같다.

Group 1: {"samples": 50, "demand_range": (5, 20), "clustering_capacity": 100},
Group 2: {"samples": 120, "demand_range": (15, 35), "clustering_capacity": 150},
Group 3: {"samples": 200, "demand_range": (1, 100), "clustering_capacity": 200},
Group 4: {"samples": 1000, "demand_range": (1, 10), "clustering_capacity": 500},
Group 5: {"samples": 300, "demand_range": (25, 60), "clustering_capacity": 350},

실행 결과는 다음과 같다.

	Group 1	Group 2	Group 3	**Group 4**	Group 5
avg_ inertia	8552.087	5667.559	2312.288	**137703.715**	8979.059
avg_ silhouette	0.3075	0.2819	0.2234	**0.3581**	0.2895
avg_std	5.9556	8.5023	23.3792	**21.8451**	16.2927
avg_ computation time	0.2475	6.9738	93.4734	**0.894**	8.3485
avg_dummy_ clusters	0.1	1.8	12.5	**0.1**	2.7

표 3. Single Linkage를 활용한 알고리즘 성능

	Group 1	Group 2	Group 3	Group 4	Group 5
avg_ inertia	**8552.087**	**5747.839**	-	136518.088	8910.019
avg_ silhouette	**0.3075**	**0.2414**	-	0.3513	0.224
avg_std	**5.6018**	**8.9568**	-	25.3794	16.1567
avg_ computation time	**0.1407**	**2.5526**	-	2.2981	45.3965
avg_dummy_ clusters	**0.1**	**1.6**	-	0.2	3.0

표 4. Complete Linkage를 활용한 알고리즘 성능

5. 결론

알고리즘의 성능은 Dataset의 종류에 따라 천차만별이었으며, 특히나 클러스터 용량이 작고 데이터 하나하나가 용량에서 큰 비중을 차지할 때 클러스터링의 난이도는 올라갔다. 데이터의 수와 수요 범위 역시 난이도에 큰 영향을 끼쳤다. 위의 표는 두 번째 알고리즘인 Network-based 알고리즘을 적용한 결과인데, 이 알고리즘은 클러스터의 수가 많을수록, 즉 노드의 수가 많을수록 수정해야 하는 네트워크의 범위도 넓어지기 때문에 클러스터 수가 많은 dataset의 처리 시간이 특히나 길었다. dataset에 따라 추가한 dummy cluster의 수도 차이를 보였으며, 이는 feasible한 해에 도달하기 위해 초기의 클러스터 수에서 몇 개를 추가했는지를 나타내는 수치이다. 같은 dataset이라 할지라도 클러스터 거리의 척

도(Single, Complete Linkage)의 종류에 따라 클러스터링 품질이 달라지는 모습을 확인할 수 있었으며 추후 더 다양한 척도(ex: Ward)를 도입하거나 Dijkstra's algorithm이 아닌 새로운 경로 탐색 방식을 택함으로써 알고리즘의 성능을 향상할 수 있다고 생각한다.

6. 연구참여 후기

겨울학기 연구참여를 하게 된 가장 큰 이유는 연구참여란 무엇인지, 어떤 것을 하는지 연구참여 자체에 대한 궁금증이 컸기 때문입니다. 학부 2년 동안 배운 것이 대학원에선 어떻게 활용될지, 공부랑 연구는 어떤 점이 다를지 등등의 궁금증은 진로를 계획함에 있어서 꼭 해결하고 넘어가야 할 것들이라고 생각했습니다. 여러 Lab 중에서도 이곳 Logistics Lab에서 하게 된 계기는 마침 2학기 때 들었던 정보시스템기술 과목 자체가 흥미로웠기 때문도 있지만 알고리즘을 통해 최적의 해에 도달, 물류를 최적화한다는 것이 재미있는 도전 과제처럼 느껴졌기 때문입니다. 실제로 6주간의 연구참여 기간도 Balanced Clustering이라는 과제에 도전하는 기간처럼 느껴졌고, 그것이 곧 이번 연구참여가 재밌다고 느낀 이유이기도 합니다. 나만의 알고리즘을 짜보고, 결과를 확인하고, 결과가 만족스럽지 않다면 다시 도전하고 하는 과정이 즐거웠습니다.

물론 어려웠던 부분도 있었습니다. Assignment처럼 정답이 존재하는 과제도 아니기에 내가 유의미한 방향으로 나아가고 있는가에 대한 의문이 계속해서 들었습니다. 이런 방식으로 문제를 해결하는 것이 맞나? 불필요한, 무의미한 코드를 짜고 있는 것은 아닌가? 이런 의문이 들기도 하였지만 결국 중요한 건 이런 경험 자체라고 생각합니다. 언제까지 가이드라인만 따라가며 살 수는 없고, 스스로 헤쳐 나가야 하는 시기가 올 것이기 때문입니다.

결국엔 좋았던 점, 배워가는 점이 많았던 연구참여 기간이라고 생각합니다. 자유롭게 라이브러리도 사용하고, 알고리즘도 짜보고 하며 python에 많이 익숙해졌고, 참고 문헌을 어떻게 활용할지도 새롭게 배웠습니다. 저한테 직접 하신 말씀은 아니고 주워들은 말이지만, 먼저 본인이 문제 해결 방식을 생각하고 참고 문헌을 찾아보라는 말씀이 기억에 남습니다. 참고 문헌을 먼저 찾아보면 생각이 그 안에 갇힌다는 이야기, 정말 공감이 갔습니다.

조금 대책 없이 연구참여를 하고 싶다고 찾아갔음에도 선뜻 받아주시고 Balanced Clustering이라는 연구 주제를 제시해 주신 김병인 교수님, 매주 피드백과 함께 연구의 방향성을 잡아주시고 연구참여 전반을 이끌어 주신 한상일 사수님께 감사드립니다. 또 참여할 기회가 생긴다면 다음엔 조금 더 부지런히 살면서 유의미한 연구 결과를 가져와 보도록 하겠습니다. 감사합니다.

7. 참고자료

1. Barreto, Sérgio, Carlos Ferreira, José Paixão, and Beatriz Sousa Santos. “Using Clustering Analysis in a Capacitated Location-Routing Problem.” European Journal of Operational Research179, no. 3 (June 2007): 968–77. https://doi.org/10.1016/j.ejor.2005.06.074.

2. Falkner, Jonas K., and Lars Schmidt-Thieme. “Neural Capacitated Clustering.” arXiv, May 19, 2023. https://doi.org/10.48550/arXiv.2302.05134.

3. França, P.m., N. M. Sosa, and V. Pureza. “An Adaptive Tabu Search Algorithm for the Capacitated Clustering Problem.” International Transactions in Operational Research6, no. 6 (1999): 665–78. https://doi.org/10.1111/j.1475-3995.1999.tb00180.x.

4. Geetha, S, G Poonthalir, and P T Vanathi. “Improved K-Means Algorithm for Capacitated Clustering Problem,” n.d.

5. Levin, M. Sh. “Capacitated Clustering Problem.” Journal of Communications Technology and Electronics, July 29, 2024. https://doi.org/10.1134/S1064226924700086.

6. Mulvey, John M., and Michael P. Beck. “Solving Capacitated Clustering Problems.” European Journal of Operational Research18, no. 3 (December 1984): 339–48. https://doi.org/10.1016/0377-2217(84)90155-3.

7. Rousseeuw, Peter J. “Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis.” Journal of Computational and Applied Mathematics20 (November 1987): 53–65. https://doi.org/10.1016/0377-0427(87)90125-7.