Optimisation de la distribution des vidéos avec Gurobi

1. Contexte et objectif

Le problème consiste à optimiser le placement des vidéos sur des serveurs caches afin de minimiser le temps moyen de streaming pour tous les utilisateurs.

- Chaque vidéo a une taille (MB).
- Chaque cache a une capacité maximale.
- Chaque endpoint (groupe d'utilisateurs) a une latence vers le data center et vers certains caches.
- On connaît le nombre de requêtes pour chaque vidéo par endpoint.

Objectif : maximiser le gain de latence total, c'est-à-dire le temps économisé lorsque les vidéos sont servies depuis un cache plutôt que le data center.

2. Description des données

Élément	Description
V	Nombre de vidéos
E	Nombre d'endpoints
R	Nombre de requêtes
C	Nombre de caches
X	Capacité des caches
video_size[v]	Taille de chaque vidéo v
LD[e]	Latence data center –> endpoint e
connections[e]	Dictionnaire {cache_id: latence} pour endpoint e
requests[r]	Tuple (video, endpoint, nombre_requetes)

---

3. Modèle mathématique exact (MIP)

Variables :

- x[v,c] ∈ {0,1} : vidéo v stockée sur cache c.
- y[r,c] ∈ {0,1} : requête r servie par cache c.

Fonction objectif :

$$\max \sum_{r \in R} \sum_{c \in C} n_r \cdot (LD[e(r)] - L_{e(r),c}) \cdot y_{r,c}$$

où n_r est le nombre de requêtes et L_{e(r),c} la latence vers le cache.

Contraintes :

1. Capacité des caches :

$$\sum_{v} size[v] \cdot x[v,c] \leq X \quad \forall c$$

2. Couplage y ≤ x :

$$y[r,c] \leq x[v(r),c] \quad \forall r,c$$

3. Une requête servie au maximum par un cache :

$$\sum_{c} y[r,c] \leq 1 \quad \forall r$$

4. Contrainte de connexion :

$$y[r,c] = 0 \quad \text{si cache c non connecté à l'endpoint de r}$$

4. Code Python pour charger les données

```python
def read_instance(filename):
    with open(filename, "r") as f:
        data = f.readlines()

    data = [line.strip() for line in data]

    # ----------------------------
    # 1. First line
    # ----------------------------
    V, E, R, C, X = map(int, data[0].split())

    # ----------------------------
    # 2. Video sizes
    # ----------------------------
    video_size = list(map(int, data[1].split()))

    # ----------------------------
    # 3. Endpoints
    # ----------------------------
    idx = 2
    LD = []                     # datacenter latency
    connections = []            # list of dicts: { cache_id : latency }
    for e in range(E):
        ld, K = map(int, data[idx].split())
        idx += 1
        LD.append(ld)
        con = {}
        for _ in range(K):
            c, lc = map(int, data[idx].split())
            con[c] = lc
            idx += 1
        connections.append(con)

    # ----------------------------
    # 4. Requests
    # ----------------------------
    requests = []        # list of tuples (video, endpoint, n_requests)
    for r in range(R):
        v, e, n = map(int, data[idx].split())
        requests.append((v, e, n))
        idx += 1
```

```
    return V, E, R, C, X, video_size, LD, connections, requests


# EXAMPLE USAGE
if __name__ == "__main__":
    instance = read_instance("trending_4000_10K.in")
    print("Instance loaded successfully!")
    print(instance)
```

5. Code Python pour Gurobi

```
from gurobipy import *
from test import read_instance


def solve_exact(filename):


    # ----------------------------
    # Load data
    # ----------------------------
    V, E, R, C, X, video_size, LD, connections, requests =
read_instance(filename)

    # ----------------------------
    # Build model
    # ----------------------------
    model = Model("HashCodeExact")

    # Variables
    x = model.addVars(V, C, vtype=GRB.BINARY, name="x")
    y = model.addVars(R, C, vtype=GRB.BINARY, name="y")

    # ----------------------------
    # Objective
    # ----------------------------
    obj = 0
    for r, (v, e, n) in enumerate(requests):
        for c in range(C):
            if c in connections[e]:    # cache is connected
```

```
                dc_lat = LD[e]
                cache_lat = connections[e][c]
                saving = (dc_lat - cache_lat) * n
                if saving > 0:
                    obj += saving * y[r, c]
            # else: y[r,c] must be constrained to 0 later

    model.setObjective(obj, GRB.MAXIMIZE)

    # ----------------------------
    # Constraints
    # ----------------------------

    # 1. One cache at most serves a request
    for r in range(R):
        model.addConstr(sum(y[r, c] for c in range(C)) <= 1)

    # 2. Coupling constraints y[r,c] ≤ x[v(r),c]
    for r, (v, e, n) in enumerate(requests):
        for c in range(C):
            model.addConstr(y[r, c] <= x[v, c])

    # 3. Caches capacity
    for c in range(C):
        model.addConstr(sum(video_size[v] * x[v, c] for v in range(V))
<= X)

    # 4. y[r,c] = 0 if cache not connected
    for r, (v, e, n) in enumerate(requests):
        for c in range(C):
            if c not in connections[e]:
                model.addConstr(y[r, c] == 0)

    # ----------------------------
    # Solve
    # ----------------------------
    model.optimize()

    # ----------------------------
    # Print solution summary
    # ----------------------------
    print("\nOptimal objective =", model.objVal)
```

```
    return model, x, y



if __name__ == "__main__":
    solve_exact("trending_4000_10k.in")
```

6. Analyse des résultats Gurobi

Instance :

- Variables : 2 385 700 binaires
- Contraintes : 2 005 657
- Coefficients : 6 357 100

Résultats notables :

- Première solution faisable (heuristique interne) : 3.5837e+08
- LP relaxation optimum : 3.9163e+10 (borne supérieure)
- Meilleure solution trouvée par Gurobi : 3.9022e+10
- Gap final : 0.36% → solution quasi optimale

Temps :

- Presolve : ~24 s
- Root LP + simplex/barrier : ~210 s
- Branch & bound : ~460 s
- Total : ~8 minutes pour un MIP énorme

7. Conclusion

L'utilisation de Gurobi pour résoudre le problème exact de placement de vidéos dans les caches s'est révélée très efficace, même pour une instance de taille relativement grande. Le modèle MIP que nous avons construit respecte toutes les contraintes du problème : capacité

des caches, unicité de service des requêtes, couplage entre requêtes et vidéos stockées, et connexion des caches aux endpoints. Grâce à Gurobi, nous avons pu obtenir une solution quasi optimale avec un écart de seulement 0,36 % par rapport à la borne supérieure fournie par la relaxation linéaire. Cela montre que le modèle est bien formulé et que le solveur est capable de gérer des milliers de variables binaires et de contraintes complexes. La solution obtenue permet donc d'optimiser efficacement le temps de streaming moyen pour les utilisateurs, tout en respectant les contraintes de capacité et de connexion des caches.