21393 Final Project

Justin Hsu, Sophia Gu, Dennis Liu

November 2025

1 Introduction

Students at Carnegie Mellon University often walk long distances between back-to-back classes. In many cases, courses that share a large number of students are scheduled in buildings that are not close to one another, which leads to unnecessary travel across campus. Since classroom assignments are mostly determined by room availability and departmental needs, the current system does not take actual student movement into account. As a result, students cannot meaningfully reduce or control the walking patterns created by their schedules.

This observation leads to our main question: if the timetable must remain fixed, is it possible to rearrange classroom locations in a way that reduces the total distance students walk between consecutive periods?

To study this question, we focus on a simplified version of the real scheduling environment. In reality, classes take place across five weekdays, class lengths vary, and students follow many different transitions throughout the week. Modeling all of these interactions at once would create a very large number of course to course relationships, and the problem would quickly become too complex to solve in practice. Therefore, we concentrate on a clean single day setting that still captures the essential behavior of student movement between consecutive periods while remaining computationally manageable.

Within this framework, we investigate whether a sequence of linear assignment problems can produce classroom layouts that reduce campus wide walking distance compared to the original schedule.

2 Mathematical Modeling

2.1 Parameters and assumptions

- C: Set of all courses. General form: $|C| = m$.
- T: Set of time slots (time-slot indices). General form: $|T| = K$ consecutive time slots (e.g., time slot 1: 9:00–9:50; time slot 2: 10:00–10:50; time slot 3: 11:00–11:50).
- $t(c)$: Time slot in which course c is taught (fixed by the timetable).
- C_t: Set of courses taught in time slot t, i.e., $C_t = \{c \in C : t(c) = t\}$.

  Example (for illustration only): Suppose we have 6 courses and 3 time slots, with exactly 2 courses per time slot. Then $C_1 = \{\text{Calc}, \text{Linear Algebra}\}$, $C_2 = \{\text{Proofs}, \text{Probability}\}$, $C_3 = \{\text{CS}, \text{Writing}\}$.
- B: Set of buildings. General form: $|B| = p$ (e.g., 4 representative buildings on campus).
- S: Set of students. General form: $|S| = n$. Each student may take a different subset of courses.
- $A_{s,c}$: Enrollment indicator. $A_{s,c} = 1$ if student $s \in S$ takes course $c \in C$, and $A_{s,c} = 0$ otherwise. This allows the model to represent different course selections across students.
- $W_{i,j}(t)$: Number of students who take course i in time slot t and course j in time slot $t+1$. It measures the overlap of students moving from course i to course j between consecutive periods.
- $D_{b,b'}$: Walking distance between buildings b and b'. This forms a $p \times p$ symmetric matrix for the p buildings, and $D_{b,b} = 0$.
- $\mathbf{Cap}_{b,t}$: Maximum number of courses per building per time slot. In our base model, we assume $\text{Cap}_{b,t} = 1$ for all $b \in B$ and $t \in T$ (one class per building per time).
- $X_{c,b}$ (decision variable): $X_{c,b} = 1$ if course c is assigned to building b, and $X_{c,b} = 0$ otherwise. Since each course c is taught in exactly one time slot $t(c)$, we do not add a separate time index to $X_{c,b}$.
- Z: Total student movement cost (person $\times$ distance) over the whole day. In our sequential approach, Z will be obtained as the sum of per-slot assignment costs.

Note: In the implementation, building assignments are determined sequentially across time slots, but the above parameters describe the full problem over the entire timetable.

2.2 Conceptual idea

We model student movement between courses as a network:

- **Nodes**: Courses.
- **Edges**: Pairs of courses scheduled in consecutive time slots t and $t+1$.
- **Edge weight (overlap)**: $W_{i,j}(t)$, the number of students who take course i in period t and course j in period $t+1$ (computed from $A_{s,c}$).

For a given building assignment, if course i is placed in building b at time t and course j is placed in building b' at time $t+1$, then each student moving along that edge incurs distance $D_{b,b'}$.

Thus, for any complete building layout, the total walking distance can be obtained by summing

$$W_{i,j}(t) \cdot D_{b(i),\, b(j)}$$

over all edges (i, j, t) connecting consecutive time slots, where $b(c)$ denotes the building assigned to course c.

If we tried to decide all $X_{c,b}$ for all time slots simultaneously and minimize this total distance, we would obtain a quadratic assignment problem: the walking cost couples building decisions across different periods through products of the form $X_{i,b}X_{j,b'}$. Such problems are computationally hard even at moderate sizes.

We replace this global quadratic assignment by a sequence of standard linear assignment problems, one for each pair of consecutive time slots. Intuitively:

- At a given time (e.g., 9:00), a set of classes has already been assigned to buildings.
- At the next time (e.g., 10:00), we have another set of classes to place into the same set of buildings.
- For each possible "class at 10:00 $\rightarrow$ building" assignment, we can compute the total walking distance from all 9:00 classes, because the 9:00 building locations are fixed.
- Therefore, assigning 10:00 classes to buildings becomes an ordinary assignment problem: each class is matched to one building, each building hosts at most one class, and the cost matrix is built from student flows and building distances.

By repeating this argument from time slot 1 to 2, 2 to 3, and so on, we construct a sequence of small linear assignment problems instead of one large quadratic model. The resulting solution is not guaranteed to be globally optimal, but it is easy to compute and directly targets student walking distance between consecutive periods.

2.3 Sequential assignment formulation

Using the enrollment indicators $A_{s,c}$, we first define the student overlap between two courses in adjacent periods as

$$W_{i,j}(t) = \sum_{s \in S} A_{s,i}\, A_{s,j}\, \mathbf{1}\{t(i) = t,\ t(j) = t+1\}, \tag{1}$$

where $\mathbf{1}\{\cdot\}$ is an indicator function.

In our approach, we do not optimize over all time slots simultaneously. Instead, we construct and solve a sequence of *linear assignment problems*, one for each pair of consecutive time slots $(t, t+1)$.

Suppose that, for a given time slot t, the building assignments of all courses in C_t are already fixed. That is, each course $i \in C_t$ has an assigned building $b(i) \in B$. For each course $j \in C_{t+1}$ and each candidate building $b' \in B$, we define the cost of assigning j to b' in period $t+1$ as

$$\mathrm{Cost}_{j,b'}^{(t+1)} = \sum_{i \in C_t} W_{i,j}(t)\, D_{b(i),\, b'}. \tag{2}$$

This captures the total walking distance generated by all students moving from courses in C_t to course j in C_{t+1}, if j is placed in building b'.

Given these costs, assigning all courses in C_{t+1} to buildings becomes a standard linear assignment problem. Using the same assignment variables $X_{c,b}$, the subproblem for time slot $t+1$ can be written as

$$\min_{X} \quad Z^{(t+1)} = \sum_{j \in C_{t+1}} \sum_{b' \in B} \text{Cost}_{j,b'}^{(t+1)} X_{j,b'} \tag{3}$$

$$\text{s.t.} \quad \sum_{b' \in B} X_{j,b'} = 1, \qquad \forall j \in C_{t+1}, \tag{4}$$

$$\sum_{j \in C_{t+1}} X_{j,b'} \leq 1, \qquad \forall b' \in B, \tag{5}$$

$$X_{j,b'} \in \{0,1\}, \qquad \forall j \in C_{t+1},\ \forall b' \in B. \tag{6}$$

Constraint (4) ensures that each course in period $t+1$ is assigned to exactly one building, while (5) enforces that each building hosts at most one course in that time slot (consistent with $\text{Cap}_{b,t} = 1$).

We treat the building assignments for the first time slot (courses in C_1) as fixed constants, initialized using the current timetable (or any reasonable initial layout). Then we apply the above assignment procedure sequentially for $t = 1, \ldots, K-1$, each time treating the previous time slot's building assignments $b(i)$, $i \in C_t$, as fixed. The total student movement cost is then

$$Z = \sum_{t=1}^{K-1} Z^{(t+1)},$$

where $Z^{(t+1)}$ is the optimal objective value of the assignment problem (3)–(6).

This sequential assignment formulation replaces the original quadratic assignment problem by a sequence of tractable linear assignment problems. It yields a heuristic solution that is not guaranteed to be globally optimal for the full quadratic model, but is easy to compute and directly focuses on minimizing student walking distance between consecutive periods.

3 Code Implementation

```
"""
Sequential assignment solver (Gurobi) for course->building assignment
Assumptions:
 - Cap_{b,t} = 1 (at most one course per building per timeslot)
 - Each course happens in exactly one timeslot (t(c))
 - Distances D[b,b'] are provided for all building pairs
"""

import csv
import gurobipy as gp
from gurobipy import GRB
from collections import defaultdict

# ---------------------------
# compute W_{i,j}(t)
# ---------------------------
def compute_W(courses, course_time, students):
    """
    Returns W dict keyed by (i,j,t) where t = time of course i and course j occurs at t
        +1.
    W[(i,j,t)] = number of students taking i at t and j at t+1.
    """
    W = defaultdict(int)
```

```
    for s, clist in students.items():
        # For each student consider all ordered pairs (i,j) they take
        # Only count if t(j) == t(i) + 1
        for i in clist:
            ti = course_time.get(i)
            if ti is None:
                continue
            for j in clist:
                tj = course_time.get(j)
                if tj is None:
                    continue
                if tj == ti + 1:
                    W[(i, j, ti)] += 1
    return dict(W)

# ---------------------------
# sequential assignment procedure
# ---------------------------
def sequential_assignment_gurobi(courses, course_time, buildings, students, D,
                                 initial_assignments=None,
                                 gurobi_params=None,
                                 verbose=True):
    """
    Performs sequential assignment from t=1..K-1.
    - initial_assignments: optional dict {course: building} for courses in the first time
        slot(s).
    - gurobi_params: optional dict of Gurobi parameters (time limit, threads, etc.)
    Returns:
       assignments: dict course -> building (final)
       total_cost: float
    """
    # Build set of times and C_t
    times = sorted(set(course_time[c] for c in courses))
    K = max(times)
    C_t = {t: [c for c in courses if course_time[c] == t] for t in times}

    # Basic feasibility check: capacity 1 -> number of courses per timeslot <= number of
        buildings
    for t, clist in C_t.items():
        if len(clist) > len(buildings):
            raise ValueError(f"Infeasible: time slot {t} has {len(clist)} courses but only
                {len(buildings)} buildings (Cap=1).")

    # Compute W
    W = compute_W(courses, course_time, students)
    if verbose:
        print(f"Computed W entries: {len(W)} (nonzero consecutive overlaps).")

    # Initialize assignments mapping
    assignments = {}
    # If user gave initial assignments for courses in earliest time(s) use that; else
        assign greedily
    earliest_t = min(times)
    C1 = C_t[earliest_t]
```

```
if initial_assignments:
    # verify provided assignments cover C1
    for c in C1:
        if c in initial_assignments:
            assignments[c] = initial_assignments[c]
    # If some C1 not provided, assign remaining greedily
    remaining_courses = [c for c in C1 if c not in assignments]
    free_buildings = [b for b in buildings if b not in set(assignments.values())]
    for c, b in zip(remaining_courses, free_buildings):
        assignments[c] = b
else:
    # default: map courses in C1 to first |C1| buildings (deterministic)
    for c, b in zip(C1, buildings):
        assignments[c] = b

if verbose:
    print(f"Initial assignments (t={earliest_t}):")
    for c in C1:
        print(f" {c} -> {assignments[c]}")

total_cost = 0.0

# Set Gurobi parameters if provided
if gurobi_params is None:
    gurobi_params = {}
# iterate through timeslots sequentially
for t in range(earliest_t, K):
    t_next = t + 1
    if t_next not in C_t:
        continue # no next slot
    Ccur = C_t[t]
    Cnext = C_t[t_next]
    if verbose:
        print(f"\nSolving assignment for timeslot {t_next} (placing {len(Cnext)}
            courses considering flows from slot {t})")

    # Build Cost_{j,b'} as Cost[(j,b')]
    Cost = {}
    for j in Cnext:
        for bprime in buildings:
            csum = 0.0
            # sum over all i in Ccur: W_{i,j}(t) * D[b(i), b']
            for i in Ccur:
                w = W.get((i, j, t), 0)
                if w == 0:
                    continue
                bi = assignments.get(i)
                if bi is None:
                    raise RuntimeError(f"Building for course {i} at time {t} is not
                        assigned.")
                dist = D.get((bi, bprime))
                if dist is None:
                    # if symmetric and only (bprime,bi) present, try reversed (but
                        distances should be filled both ways)
```

```
                dist = D.get((bprime, bi))
                if dist is None:
                    raise KeyError(f"Distance not found for building pair ({bi},{
                        bprime})")
            csum += w * dist
        Cost[(j, bprime)] = csum

# Build and solve Gurobi assignment model for Cnext
model = gp.Model(f"assign_t{t}_t{t_next}")
# optional parameter setting
for key, val in gurobi_params.items():
    model.setParam(key, val)
# create variables X[j,b'] for j in Cnext, b' in buildings
X = model.addVars(Cnext, buildings, vtype=GRB.BINARY, name="X")

# constraints: each course assigned to exactly one building
for j in Cnext:
    model.addConstr(gp.quicksum(X[j, b] for b in buildings) == 1, name=f"
        one_building_{j}")

# capacity: at most 1 course per building
for b in buildings:
    model.addConstr(gp.quicksum(X[j, b] for j in Cnext) <= 1, name=f"cap_{b}_t{
        t_next}")

# objective: minimize sum Cost[(j,b)] * X[j,b]
model.setObjective(gp.quicksum(Cost[(j, b)] * X[j, b] for j in Cnext for b in
    buildings), GRB.MINIMIZE)

model.setParam('OutputFlag', 0 if not verbose else 1)
model.optimize()

if model.status != GRB.OPTIMAL and model.status != GRB.TIME_LIMIT and model.status
     != GRB.SUBOPTIMAL:
    raise RuntimeError("Gurobi did not find a feasible/optimal solution for the
        assignment subproblem.")

# extract assignments for Cnext and accumulate cost
for j in Cnext:
    assigned_b = None
    for b in buildings:
        val = X[j, b].X
        if val > 0.5:
            assigned_b = b
            break
    if assigned_b is None:
        raise RuntimeError(f"No assignment found for course {j} at timeslot {t_next
            }.")
    assignments[j] = assigned_b

sub_obj = model.objVal
total_cost += sub_obj
if verbose:
    print(f"Subproblem objective Z^{t_next} = {sub_obj:.2f}")
```

```
            print("Assignments (this step):")
            for j in Cnext:
                print(f" {j} -> {assignments[j]}")

    if verbose:
        print("\nFinal assignments for all courses:")
        for c in sorted(assignments.keys(), key=lambda x: (course_time[x], x)):
            print(f" t={course_time[c]}: {c} -> {assignments[c]}")

        print(f"\nTotal walking cost Z (sum of subproblem objectives) = {total_cost:.2f}")

    return assignments, total_cost
```

4 Data Analysis

We ran our algorithm on a set of sample data from 40 students across 15 different classes in 4 buildings on campus. The following tables outline our data and resulting building assignments.

Table 1: Building-to-Building Walking Distances

Building	**Porter**	**Baker**	**Wean**	**Doherty**
Porter	0	2	5	7
Baker	2	0	5	4
Wean	5	5	0	5
Doherty	7	4	5	0

Table 2: Student Schedules Across Time Slots

Student	Slot A	Slot B	Slot C	Slot D
Student 1	Math	Biology	Art	Psychology
Student 2	Math	History	Music	Psychology
Student 3	Math	Economics	Philosophy	Psychology
Student 4	Math	Computer Science	Statistics	Psychology
Student 5	Math	Biology	Art	Psychology
Student 6	English	History	Philosophy	Psychology
Student 7	English	Economics	Statistics	Psychology
Student 8	English	Computer Science	Art	Psychology
Student 9	English	Biology	Music	Psychology
Student 10	English	History	Philosophy	Psychology
Student 11	Physics	Economics	Music	Psychology
Student 12	Physics	Computer Science	Philosophy	Psychology
Student 13	Physics	Biology	Statistics	Psychology
Student 14	Physics	History	Art	Psychology
Student 15	Physics	Economics	Music	Psychology
Student 16	Chemistry	Computer Science	Statistics	Psychology
Student 17	Chemistry	Biology	Art	Psychology
Student 18	Chemistry	History	Music	Psychology
Student 19	Chemistry	Economics	Philosophy	Psychology
Student 20	Chemistry	Computer Science	Statistics	Psychology
Student 21	Math	History	Philosophy	Sociology
Student 22	Math	Economics	Statistics	Sociology
Student 23	Math	Computer Science	Art	Sociology
Student 24	Math	Biology	Music	Sociology
Student 25	Math	History	Philosophy	Sociology
Student 26	English	Economics	Music	Sociology
Student 27	English	Computer Science	Statistics	Sociology
Student 28	English	Biology	Art	Sociology
Student 29	English	History	Philosophy	Sociology
Student 30	English	Economics	Music	Sociology
Student 31	Physics	Computer Science	Art	Political Science
Student 32	Physics	Biology	Statistics	Political Science
Student 33	Physics	History	Music	Political Science
Student 34	Physics	Economics	Philosophy	Political Science
Student 35	Physics	Computer Science	Art	Political Science
Student 36	Chemistry	Biology	Music	Political Science
Student 37	Chemistry	History	Statistics	Political Science
Student 38	Chemistry	Economics	Art	Political Science
Student 39	Chemistry	Computer Science	Philosophy	Political Science
Student 40	Chemistry	Biology	Music	Political Science

Table 3: Final Building Assignments by Time Slot (Total Cost: 354)

Time Slot	Course	Assigned Building
1	Chemistry	Baker
	English	Doherty
	Math	Porter
	Physics	Wean
2	Biology	Porter
	Computer Science	Wean
	Economics	Doherty
	History	Baker
3	Art	Porter
	Music	Baker
	Philosophy	Doherty
	Statistics	Wean
4	Political Science	Porter
	Psychology	Baker
	Sociology	Doherty

5 Conclusion

Our sequential algorithm illustrates the trade off between saving time and sacrificing accuracy. A typical LP can be solved using simplex in $O(n^2m)$ time, where n is the number of decision variables and m is the number of constraints. In our case, $n = c * b$ and $m = c + b$. Let c equal the number of courses for a single time slot and b equal the number of buildings for a single time slot. Thus, the time complexity for our algorithm is $O(c^2b^2(c+b))$. On the other hand, a quadratic assignment problem is of $O((c*b)!)$. Comparing the two run times, we see that our model reduces the time complexity drastically.

However, in exchange, we settle for a sub-optimal solution. Since our data set was small, we were able to find the true optimal assignment, with cost 338. Our algorithm produced an assignment with a cost of 354. At least for our data, our algorithm gave us a reasonable solution.

Moving forward, we could improve our algorithm by accounting for more complexity in class transitions. For example, assume we assign a lot of classes in the same area. In theory, this is good because no two classes are very far apart. In reality, this will cause traffic jams in the hallway as students leave class at the same time. There are other nuances to schedules, such as classes of different lengths, student accommodations, etc. Future models based of ours may account for these other factors.