Title of the Paper

Motoki Nakajima
St. Paul's School
motoki.nakajima@sps.edu

February 27, 2026

1 Day 1

1.1 Topic Area

I select 7. A well-defined algorithmic modification.

1.2 Formal Research Question

In a circular harkness table discussion, how effective is the beam search algorithm, compared to a simple greedy algorithm, in determining the table arrangement that maximizes cross-talks between students?

1.3 Formal Problem Definition

The problem models a circular harkness table discussion geometrically, where the cross-talk between student i and student j is modeled as $|v_i v_j \cos(\theta_{ij})|$, where θ_{ij} is the angle between the vector $\overrightarrow{v_i v_j}$ and $\overrightarrow{Ov_i}$, and O is the center of the circular table. Because we take the absolute value of the cosine, having talkative people across the table helps active participation. There are n students in the discussion, where each student is given an index of normalized talkativeness v_i ($0 \leq v_i \leq 1$). The goal is to find the arrangement of students around the table that maximizes the total cross-talk, which can be expressed as $\sum_{i \neq j} |v_i v_j \cos(\theta_{ij})|$.

1.4 Hypothesis

The beam search algorithm will yield a significantly higher total cross-talk compared to the greedy algorithm, because the greedy algorithm tend to get stuck in an inefficient local maximum, while the beam search explores wider and tend to find a better local maximum. Although the beam search cost more time, considering the small number of students in a typical harkness table discussion, the importance of maximizing the cross-talk performance outweighs the cost of time.

1.5 Theoretical Plan

The theoretical analysis will consist of two parts: loop invariant proof that the beam search algorithm will aways find a solution that is at least as good as the greedy algorithm, and a tight bound on the time complexity of the beam search algorithm and the greedy algorithm. The loop invariant proof will be based on the trait of the beam search algorithm that it always explores a wider range of arrangements than the greedy algorithm, while also searching through the same arrangements as

the greedy algorithm. The time complexity analysis will be based on the number of arrangements explored by each algorithm, which is determined by the number of students and the beam width. The greedy algorithm has a time complexity of $O(n^2)$, while the beam search algorithm has a time complexity of $O(B \cdot n^2)$, where B is the beam width.

1.6 Experimental Plan

The input will be a vector of n normalized talkativeness values v_i, which all of them will be generated randomly through `std::random_device` in C++. The single array of input will automatically determine the size of the harkness table and the students' talkativeness, and thus completely determine all the parameters of the problem. The test will be conducted for various sizes of n from 5 to 100, with increments of 5, and for various beam widths B from 1 to 10, with increments of 1. 20 trials will be conducted for each combination of n and B, and the average total cross-talk will be recorded for both the greedy algorithm and the beam search algorithm. The total cross-talk index will then be plotted by `matplotlib` in Python in three different ways: a 3d plot of total cross-talk against n and B, a 2d plot of total cross-talk against n for a fixed B where the difference between the two algorithms are at largest, and a 2d plot of total cross-talk against n for a fixed B where the difference is smallest. The results will be analyzed to determine the effectiveness of the beam search algorithm compared to the greedy algorithm in maximizing the total cross-talk in a circular harkness table discussion.

2 Day 2/3

2.1 Problem definition

The problem models an elliptical harkness table discussion geometrically. The score of each student's contribution to the table is defined by the student's affinity level with the other students, while considering a geometric cross-talk coefficient that depends on the angle between the students.

$$S_i = a_{ij} \cdot k_{ij} \quad (\text{for all } j \neq i)$$

$$k_{ij} = d_{ij} \cdot \left(\sin^{16}(\theta_{ij}) + 0.3\cos^{8}(\theta_{ij}) - 4.0\cos^{64}(\tfrac{\theta_{ij}}{2}) \right)$$

S_i is the score of student i, a_{ij} is the affinity level between student i and student j, k_{ij} is the cross-talk coefficient between student i and student j, θ_{ij} is the central angle between student i and student j, and d_{ij} is the distance between student i and student j. The total score of the table arrangement is defined as the sum of all scores amond the students.

$$S = \sum_{i=1}^{n} S_i$$

The geometrically determined cross-talk coefficient k_{ij} is modeled based on feedbacks from Humanities teachers at St. Paul's School, and it is designed to capture the complex relationship between the angle of seating and the level of cross-talk.

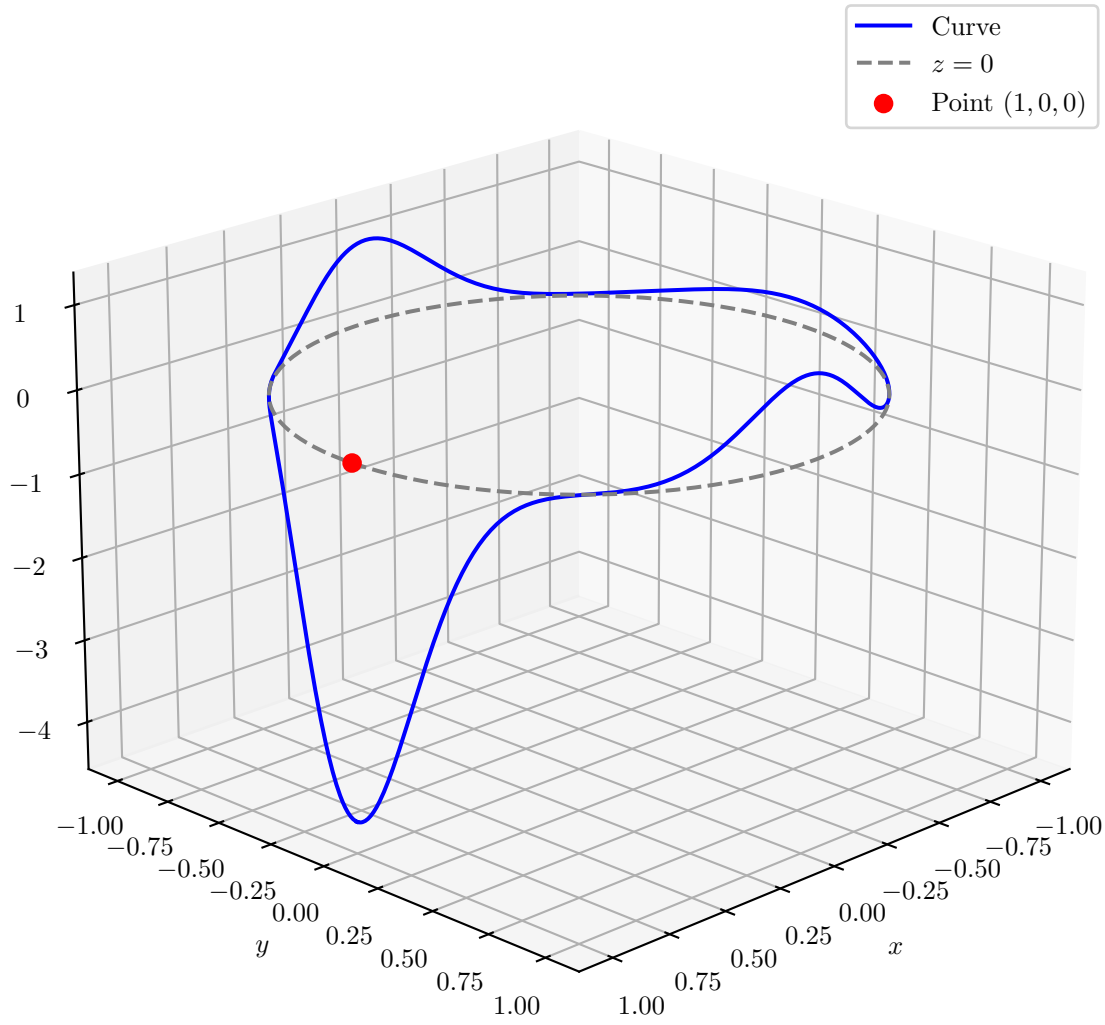

Figure 1: Cross-talk coefficient $\frac{k_{ij}}{d_{ij}}$ around the table.

The goal of the problem is to find the arrangement of students around the elliptical table that maximizes the total score S.

2.2 Pseudocode

Algorithm 1 Arrangement constructor, gain, and place

function InitArrangement(n)
    $order \leftarrow [-1, \ldots, -1]$ (size n)
    $seat\_taken \leftarrow [\textbf{false}, \ldots, \textbf{false}]$
    $student\_placed \leftarrow [\textbf{false}, \ldots, \textbf{false}]$
    $partial\_gain \leftarrow n \times n$ matrix of 0.0
    $current\_score \leftarrow 0.0$
function Gain($student\_idx, seat$)
    return $2.0 \cdot partial\_gain[student\_idx][seat]$
function Place($student\_idx, seat, problem$)
    $current\_score \leftarrow current\_score +$ Gain($student\_idx, seat$)
    $order[seat] \leftarrow student\_idx$
    $seat\_taken[seat] \leftarrow \textbf{true}$
    $student\_placed[student\_idx] \leftarrow \textbf{true}$
    for $m \leftarrow 0$ to $n-1$ **do**
        if $student\_placed[m]$ **then**
            continue
        for $s \leftarrow 0$ to $n-1$ **do**
            if $seat\_taken[s]$ **then**
                continue
            $partial\_gain[m][s] \leftarrow partial\_gain[m][s] + affinity[m][student\_idx] \cdot$
    GetCrossTalk($problem, s, seat$)

Algorithm 2 Problem initialization and cross-talk lookup

function InitProblem($n, seed, aspect\_ratio$)
    $a \leftarrow aspect\_ratio,\ b \leftarrow 1.0$
    for $i \leftarrow 0$ to $n-1$ **do**
        $px[i] \leftarrow a\cos(2\pi i/n)$
        $py[i] \leftarrow b\sin(2\pi i/n)$
    Initialize $cross\_talk\_coeff$ as $n \times n$ zeros
    for $i \leftarrow 0$ to $n-1$ **do**
        for $j \leftarrow 0$ to $n-1$ **do**
            if $i = j$ **then**
                continue
            $dot \leftarrow px[i]px[j] + py[i]py[j]$
            $len_i \leftarrow \sqrt{px[i]^2 + py[i]^2},\ len_j \leftarrow \sqrt{px[j]^2 + py[j]^2}$
            $c_\theta \leftarrow$ Clamp$(dot/(len_i len_j), -1, 1)$
            $\theta \leftarrow \arccos(c_\theta)$
            $s \leftarrow |\sin(\theta)|,\ c \leftarrow |\cos(\theta)|,\ h \leftarrow \cos(\theta/2)$
            $cross\_talk\_coeff[i][j] \leftarrow s^{16} + 0.3c^8 - 4.0h^{64}$
    Initialize symmetric $affinity$ matrix using RNG seeded by $seed$
function GetCrossTalk($problem, i, j$)
    return $problem.cross\_talk\_coeff[i][j]$

Algorithm 3 GreedySolver::solve()

```
1: function GreedySolve(problem)
2:     arrangement ← InitArrangement(problem.n)
3:     for step ← 0 to problem.n − 1 do
4:         best_student, best_seat ← −1, −1
5:         best_gain ← −10^18
6:         for k ← 0 to problem.n − 1 do
7:             if arrangement.student_placed[k] then
8:                 continue
9:             for s ← 0 to problem.n − 1 do
10:                if arrangement.seat_taken[s] then
11:                    continue
12:                g ← Gain(k, s)
13:                cur ← arrangement.current_score + g
14:                best ← arrangement.current_score + best_gain
15:                if cur > best or (cur = best and tie-break on smaller k, then smaller s) then
16:                    best_gain ← g
17:                    best_student ← k
18:                    best_seat ← s
19:        Place(best_student, best_seat, problem)
20:    return arrangement
```

Algorithm 4 BeamSearchSolver::solve()

function BeamSearchSolve($problem, beam\_width$)
  $beam \leftarrow$ [InitArrangement($problem.n$)]
  for $step \leftarrow 0$ to $problem.n - 1$ **do**
    $scored \leftarrow$ [ ]
    for each parent index bi in $beam$ **do**
      for each unplaced student k **do**
        for each empty seat s **do**
          $score \leftarrow beam[bi].current\_score + beam[bi].gain(k, s)$
          append $(score, bi, k, s)$ to $scored$
    Stable-sort $scored$ by: descending score, then ascending (bi, k, s)
    $max\_per\_parent \leftarrow 2$
    $parent\_count \leftarrow$ empty map, $selected \leftarrow$ [ ]
    $greedy\_cand\_idx \leftarrow$ first index in $scored$ with $bi = 0$ (or none)
    if $greedy\_cand\_idx$ exists **then**
      add $scored[greedy\_cand\_idx]$ to $selected$
      $parent\_count[0] \leftarrow 1$
    for $i \leftarrow 0$ to $|scored| - 1$ while $|selected| < beam\_width$ **do**
      if $i = greedy\_cand\_idx$ **then**
        continue
      if $parent\_count[scored[i].bi] < max\_per\_parent$ **then**
        add $scored[i]$ to $selected$
        increment $parent\_count[scored[i].bi]$
    $new\_beam \leftarrow$ [ ]
    for each candidate c in $selected$ **do**
      $next \leftarrow$ copy of $beam[c.bi]$
      Place($c.student\_idx, c.seat, problem$) on $next$
      append $next$ to new_beam
    $beam \leftarrow new\_beam$
  return arrangement in $beam$ with maximum $current\_score$

Algorithm 5 main() experiment sweep in `src/main.cpp`

function Main
  Set constants: N_MIN=5, N_MAX=60, N_STEP=5, B_MAX=20, $TRIALS$=50
  Open CSV output file and write header
  for $n \leftarrow N\_MIN$ to N_MAX step N_STEP **do**
    for $trial \leftarrow 0$ to $TRIALS - 1$ **do**
      $seed \leftarrow n \cdot 10000 + trial$
      $problem \leftarrow$ InitProblem($n, seed, 11/7$)
      $greedy\_score \leftarrow$ GreedySolve($problem$).$current\_score$
      for $B \leftarrow 1$ to B_MAX **do**
        $beam\_score \leftarrow$ BeamSearchSolve($problem, B$).$current\_score$
        Write $(n, B, trial, greedy\_score, beam\_score)$ to CSV
  Close file and return success

2.3 Theoretical reasoning

The beam search algorithm is the core contribution in this project. The key implementation detail in `BeamSearchSolver::solve()` is that at every step, the algorithm forcibly includes the best candidate whose parent index is $bi = 0$ (stored as `greedy_cand_idx`) before filling the rest of the beam. Since index 0 is recursively the greedy-descendant state from the previous step, this preserves one full greedy trajectory throughout all steps.

After step t, the beam contains the same partial arrangement that greedy would produce after t placements.

Base case ($t = 0$). The beam is initialized with a single empty arrangement, which is identical to greedy's initial state.

Inductive step. Assume the greedy partial arrangement is present at index 0 before step $t+1$. The algorithm explicitly selects the highest-scoring child of parent 0. Therefore, the greedy child is inserted into the next beam and remains at index 0 in `new_beam`. Therefore, the invariant holds for step $t + 1$.

At termination ($t = n$), one complete greedy arrangement exists in the final beam. The returned solution is the maximum-score arrangement in that beam, so

$$S_{\text{beam}} \geq S_{\text{greedy}} \quad \text{for all } B \geq 1.$$

Thus, beam search strictly generalizes greedy in this implementation, while potentially improving quality by exploring additional high-scoring branches.

2.4 Complexity derivation

At each step, candidate generation scans all B current arrangements and all feasible $(student, seat)$ pairs, so it creates $O(Bn^2)$ candidates. Sorting takes $O(Bn^2 \log(Bn))$ time. Expanding up to B arrangements into the next beam costs $O(Bn^2)$ per step.

Each expansion performs a copy of an `Arrangement` (includes an $n \times n$ `partial_gain` table) and one `place()` call, both bounded by $O(n^2)$. Hence expansion cost is $O(Bn^2)$ per step. Summing over all n steps gives

$$T_{\text{beam}}(n, B) = O\left(\sum_{r=1}^{n} Br^2 \log(Br^2)\right) + O(Bn^3) = O\big(Bn^3 \log(Bn)\big)\,.$$

For greedy, each step scans all valid $(student, seat)$ pairs and then calls `place()`, both bounded by $O(n^2)$ in this implementation, so

$$T_{\text{greedy}}(n) = O\left(\sum_{r=1}^{n} r^2\right) = O(n^3).$$

Therefore, beam search adds an approximately linear factor in B (plus sorting overhead) in exchange for higher solution quality and the guarantee $S_{\text{beam}} \geq S_{\text{greedy}}$.

2.5 Experimental plan

To isolate the effect of beam search, we evaluate $B \in \{1, 2, \ldots, 20\}$ over $n \in \{5, 10, \ldots, 60\}$ with 50 trials per n. Each trial uses a deterministic seed $seed = n \cdot 10000 + trial$, and both algorithms run on the same generated `Problem`. The CSV records $(n, B, trial, greedy\_score, beam\_score)$, enabling direct paired comparisons.

Primary metrics are:

- absolute gain: $\Delta = S_{\text{beam}} - S_{\text{greedy}}$,
- relative gain: $\Delta_{\%} = \frac{S_{\text{beam}} - S_{\text{greedy}}}{|S_{\text{greedy}}|} \times 100$,
- marginal benefit curve as B increases.

Because $B = 1$ reproduces the greedy trajectory in this implementation, it serves as an internal correctness check for the beam framework.

2.6 Preliminary results

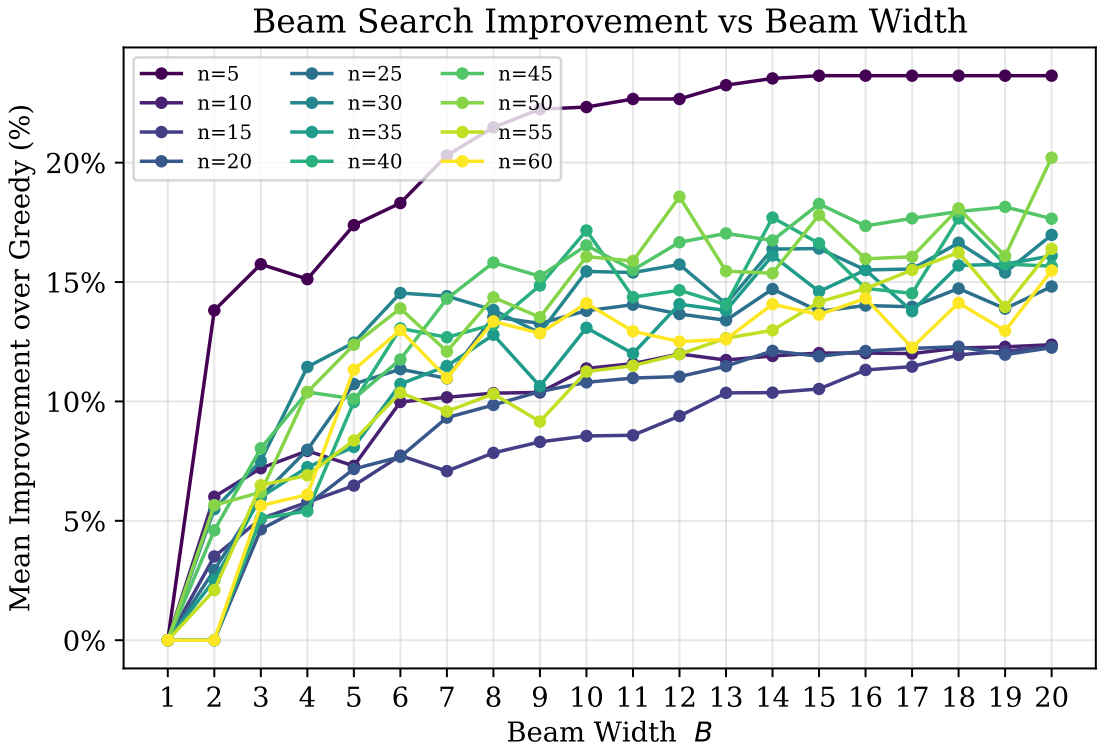

Figure 2: Preliminary results showing the average total score for greedy and beam search across different n and B.