A Peer-to-Peer Tutorial Model for CS Theory and Algorithms

An Experiment in Scaling Intensive, Interactive Learning

"The ideal college is [Professor] Mark Hopkins on one end of a log and a student on the other."

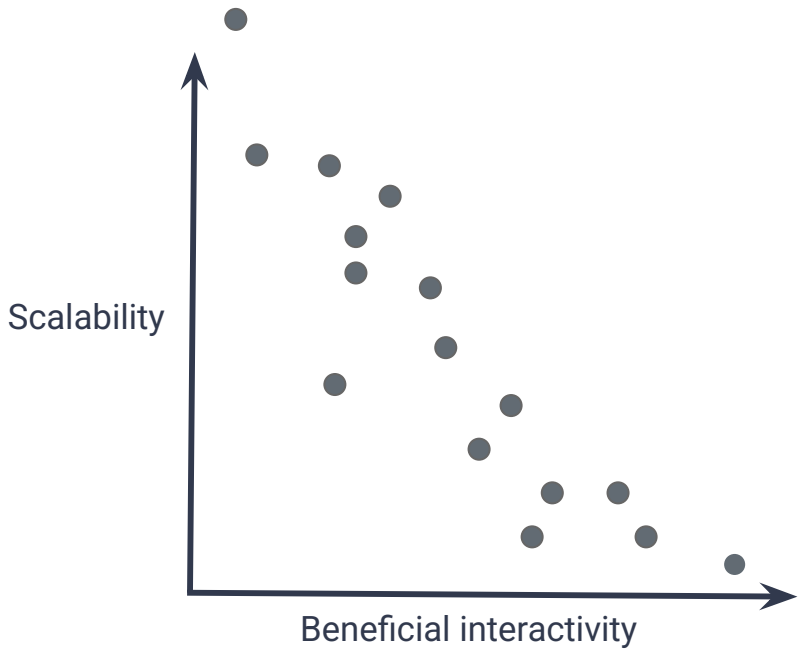
– James A. Garfield

Tim Randolph
Department of Computer Science
Harvey Mudd College

HARVEY MUDD COLLEGE

Overview

Motivation: (Inter)active learning has many benefits for students, but those benefits don't scale easily.

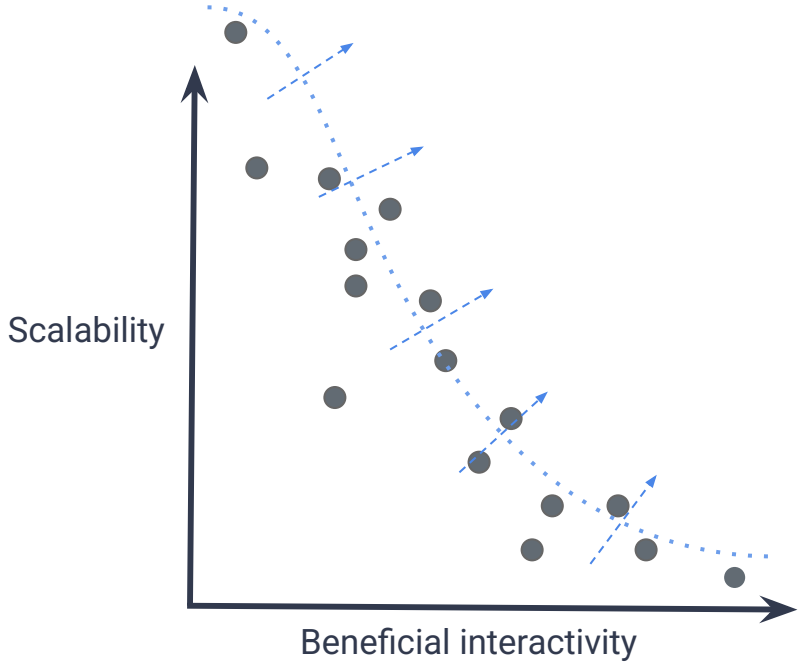


"Scalability" of pedagogical strategies (students per instructor-hour) as a function of interactivity.

(Not a real chart.)

Overview

Motivation: (Inter)active learning has many benefits for students, but those benefits don't scale easily.

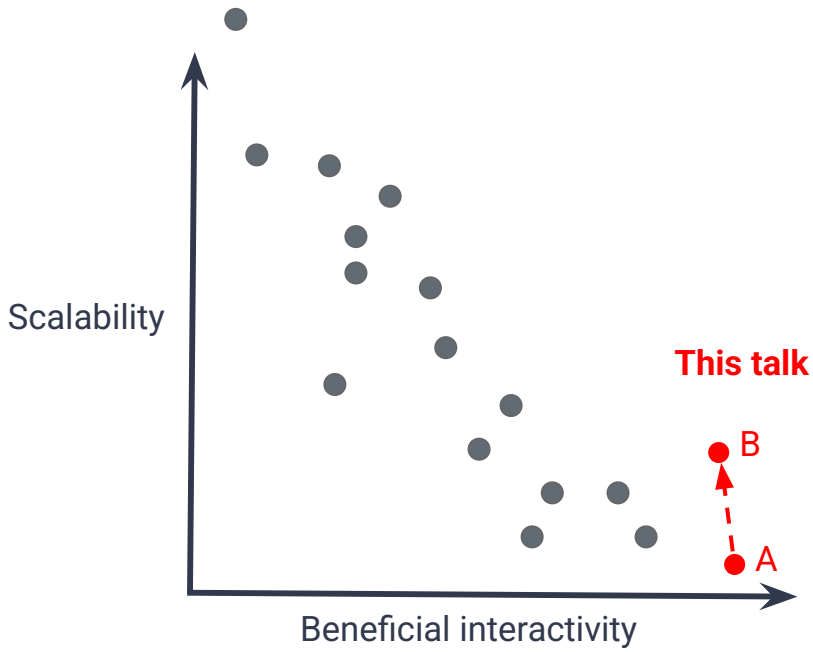


"Scalability" of pedagogical strategies (students per instructor-hour) as a function of interactivity.

(Not a real chart.)

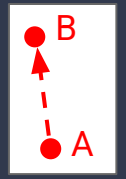
Overview

Motivation: (Inter)active learning has many benefits for students, but those benefits don't scale easily.

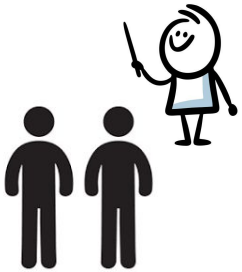


"Scalability" of pedagogical strategies (students per instructor-hour) as a function of interactivity.

(Not a real chart.)



Point A: Tutorials

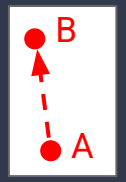


- 2 students per “section”
- Meet once a week for 60-120 minutes.

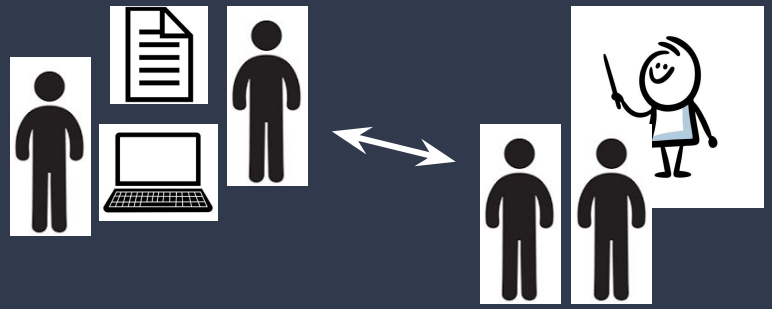
Weekly structure:

- Guided by a **course of study**, students work independently and/or with a partner outside of class.
  - Readings
  - Problem sets
  - (Pair) programming deliverables
  - Presentation plan
- Instructor meeting
  - Students present technical material and deliverables
  - Instructor probes, prompts, challenges

UNIVERSITY OF OXFORD



Point A: Tutorials



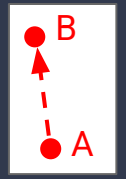
Features:

- Students actively practice a variety of technical and **personal** skills
  - Collaboration
  - Independent, self-directed learning
  - Communication
  - Reflection and metacognition
- Student understanding stress-tested by peers and instructor
  - Built-in oral evaluation allows relative freedom outside the classroom
- Modular, extensible course structure
- Flexible depth of engagement week-to-week

Well-suited for:

- Mature, intrinsically motivated learners
- Topics courses, seminars

Major drawback: only feasible for 8-10 students



Point B: “Peer-to-Peer” Tutorials

Thursday

Friday

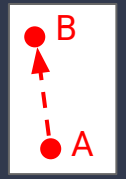
Saturday

Sunday

Monday

Tuesday

- All student pairs and instructor meet same classroom, same time
- Two weekly class meetings of ~75 minutes



Point B: “Peer-to-Peer” Tutorials

Thursday

Friday

Saturday

Sunday

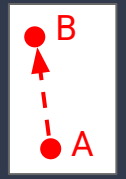
Monday

Tuesday

- All student pairs and instructor meet same classroom, same time
- Two weekly class meetings of ~75 minutes

Meeting 1: Overview Lecture

- What’s coming this week?
  - Key concepts and relationships
  - Mental models and metaphors
  - Likely sticking points
  - Examples
  - ~~Fine details and long proofs~~
- Prep time
  - Review course of study
  - Pairs plan meetings and work blocks



Point B: “Peer-to-Peer” Tutorials

Thursday

Friday

Proving the Cook-Levin Theorem

Saturday

Sunday

PSPACE; Polynomial Hierarchy

Monday

Tuesday

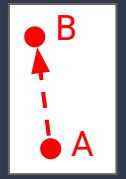
- All student pairs and instructor meet same classroom, same time
- Two weekly class meetings of ~75 minutes

Meeting 1: Overview Lecture

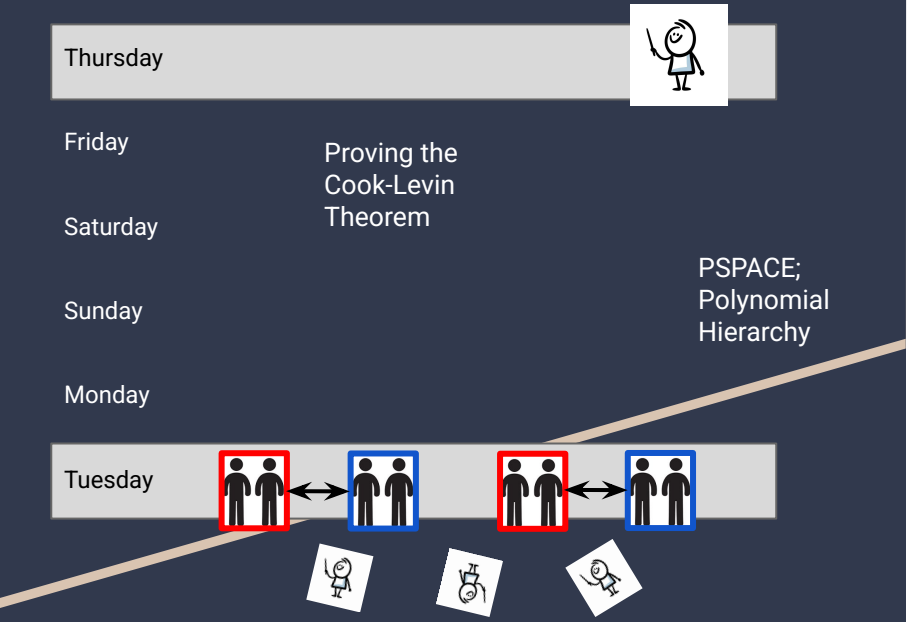
- What’s coming this week?
  - Key concepts and relationships
  - Mental models and metaphors
  - Likely sticking points
  - Examples
  - ~~Fine details and long proofs~~
- Prep time
  - Review course of study
  - Pairs plan meetings and work blocks

Partner Work:

- “**Red pairs**” and “**Blue pairs**” follow *complementary* courses of study
- Prompts focus on process and metacognition, not outcomes
- Flexible depth of study



Point B: “Peer-to-Peer” Tutorials



- All student pairs and instructor meet same classroom, same time
- Two weekly class meetings of ~75 minutes

Meeting 1: Overview Lecture

- What’s coming this week?
  - Key concepts and relationships
  - Mental models and metaphors
  - Likely sticking points
  - Examples
  - ~~Fine details and long proofs~~
- Prep time
  - Review course of study
  - Pairs plan meetings and work blocks

Partner Work:

- “**Red pairs**” and “**Blue pairs**” follow *complementary* courses of study
- Prompts focus on process and metacognition, not outcomes
- Flexible depth of study

Meeting 2: Co-Presentations

- Red pairs present to blue pairs and vice versa
- Instructor rotates
- Short share-out (High points? Muddy points?)
- Written reflection:
  - Self and partner: organization, equal contribution, presentation preparation
  - Co-team: preparation, clarity, communication of must-do’s, active listening, highs and lows, compliments
- Submit materials and artifacts

Example Course of Study

CSCI 145: Advanced Topics in Algorithms Week 5
Approximation Algorithms: When "Good Enough" is Good Enough

1 Readings

Everything listed below should be available in PDF form in Canvas/Files/Readings or Canvas/Modules under Week 5. Let me know if that's not true!

1. Kleinberg-Tardos pp. 612-617. Both teams: read for understanding.
   - 11.3: Approximating Weighted Set Cover. A neat, standalone $O(\log n)$-approximation algorithm for Set Cover. (Is that good? It depends on your perspective. As it turns out, you provably *can't* achieve an $o(\log n)$-approximation for this problem unless $\mathsf{P} = \mathsf{NP}$.) This algorithm pursues the natural strategy of greedily choosing the set that maximizes the amount of "bang for your buck"; that is, it attempts to minimize total weight by choosing sets that cover the most elements relative to their weight. The analysis uses a "pricing" strategy that might remind you of amortized analysis arguments.
   - (Optional) 11.4: Vertex Cover can be seen as a special case of Set Cover (given an instance of Vertex Cover, create a set for each vertex consisting of all the adjacent edges). As a result, our approximation algorithm for Set Cover actually gives us an approximation algorithm for Vertex Cover! But we can do even better if we're careful about counting the "price" of our algorithm's choices.
2. Kleinberg-Tardos 644-649. [red] teams: read for understanding (and implementation!). [blue] teams: skim for active listening.
   - We already know a *pseudopolynomial* algorithm for the knapsack problem: given n input items with weights and values and a maximum capacity W, we can solve the problem in time $O(nW)$ by dynamic programming.
3. Arora pp. 1-13. Odd teams: read for understanding. Even teams: skim for active listening.
   - This survey walks through a famous algorithm due independently to Sanjeev Arora and Joseph Mitchell: there exists a polynomial-time approximation scheme for Euclidean TSP! This discovery came about as a result of trying to prove that *no* such approximation algorithm existed, and won Arora and Mitchell the Gödel prize (the highest award in theoretical computer science).
   - Pages 1-3 (introduction) provides interesting background. This section breezes through a lot of research results very quickly, and you don't need to follow closely. Your main reading assignment is Sections 1 and 2 (pages 4-9). Section 3 (pages 9-13) is encouraged but not required.
   - Note: this is a research-caliber exposition of a prize-winning result. It may be challenging! Digesting and explaining this reading will be Odd Teams' major task for the week. Take it one step at a time and feel free to skip over technical pieces and come back to them on a second read.
   - Here's a high-level summary of the algorithm to get you started:
     - Our problem is (2-dimensional) Euclidean TSP, specifically the optimization version. The input is a set of n points in the plane and the goal is to find the shortest possible cycle that visits every point exactly once.
     - **Step 1: Rounding, a.k.a. "Perturbation".** We do a "snap to grid" operation on our input points using a very small grid. If the grid squares are small enough, then the length of any cycle on the rounded instance is extremely close to the length of the corresponding cycle on the real instance.
     - **Step 2: Randomized Dissection.** Divide the space containing our points into quadrants, divide the quadrants into quadrants, and so on. The result is a bunch of little boxes containing very few points each, organized in a way that can be stored in a quadtree (tree of branching factor 4).
       Our strategy will be to solve the problem within each little box and recursively combine the solutions. However, this strategy will depend on the number of times our TSP tour crosses the boundary of the box, which could be very large if many points are positioned badly relative to the box boundary. We can rule out this bad case by translating the grid of little boxes by a random amount (making the position of each point uniformly random with respect to the box containing it).
     - **Step 3: Thinking with Portals.** We specify that our TSP tour can only enter or leave each box through a small number of special points called "portals". We'll characterize the way that a TSP tour travels through the box in terms of the way it enters and exits the various portals and prove that there are a limited number of ways that it can do this.
     - **Step 4: Dynamic Program & Conquer.** Arora's write-up describes the dynamic agree
       al TSP
       ng out
       nstant
       n time
       nations
       ve the
       ore all
       these solutions in a DP table. Then we solve the problem for each level of larger boxes by trying all 4-tuples of compatible solutions for its four sub-boxes.

Introductory reading (Both teams)

Heavily scaffolded technical reading
(Red team: read to present)
(Blue team: skim for active listening)

Example Course of Study

CSCI 145: Advanced Topics in Algorithms Week 5
Approximation Algorithms: When "Good Enough" is Good Enough

2 Problems

1. Odd team: strictly speaking there are no *problems* this week, but the Arora reading sometimes makes claims without supporting explanation. See if you can work out the following as you read:
   - Warm-up: In Section 1.1, Arora makes some claims that are easy to prove but not necessarily obvious. For starters, why is it always true that $2d \leq OPT \leq nd$? Why does making "excursions" imply that rounding the input points affects the cost of an optimum tour by at most $\varepsilon d/n^{0.5}$? Why is it OK to "rescale distances"?
   - In Section 1.2, why is it safe to assume that L is a power of 2 w.l.o.g.?
   - In Section 1.3, why can we assume that the optimal tour never visits any portal more than twice? (Hint: the Galanis and Nagarajan notes both have good figures for this.)
   - In Section 1.3 (DP section), why is the number of portal respecting tours that enter/exit each dissection square at most $4k$ times bounded by $(m^{O(k)}k!)$? Looking back at how we set the variables m and k, what is this quantity in terms of n and ε?

Exercises as additional scaffolding
(Red team)

3. Arora pp. 1-13. Odd teams: read for understanding. Even teams: skim for active listening.
   - This survey walks through a famous algorithm due independently to Sanjeev Arora and Joseph Mitchell: there exists a polynomial-time approximation scheme for Euclidean TSP! This discovery came about as a result of trying to prove that *no* such approximation algorithm existed, and won Arora and Mitchell the Gödel prize (the highest award in theoretical computer science).
   - Pages 1-3 (introduction) provides interesting background. This section breezes through a lot of research results very quickly, and you don't need to follow closely. Your main reading assignment is Sections 1 and 2 (pages 4-9). Section 3 (pages 9-13) is encouraged but not required.
   - Note: this is a research-caliber exposition of a prize-winning result. It may be challenging! Digesting and explaining this reading will be Odd Teams' major task for the week. Take it one step at a time and feel free to skip over technical pieces and come back to them on a second read.
   - Here's a high-level summary of the algorithm to get you started:
     - Our problem is (2-dimensional) Euclidean TSP, specifically the optimization version. The input is a set of n points in the plane and the goal is to find the shortest possible cycle that visits every point exactly once.
     - **Step 1: Rounding, a.k.a. "Perturbation".** We do a "snap to grid" operation on our input points using a very small grid. If the grid squares are small enough, then the length of any cycle on the rounded instance is extremely close to the length of the corresponding cycle on the real instance.
     - **Step 2: Randomized Dissection.** Divide the space containing our points into quadrants, divide the quadrants into quadrants, and so on. The result is a bunch of little boxes containing very few points each, organized in a way that can be stored in a quadtree (tree of branching factor 4).
       Our strategy will be to solve the problem within each little box and recursively combine the solutions. However, this strategy will depend on the number of times our TSP tour crosses the boundary of the box, which could be very large if many points are positioned badly relative to the box boundary. We can rule out this bad case by translating the grid of little boxes by a random amount (making the position of each point uniformly random with respect to the box containing it).
     - **Step 3: Thinking with Portals.** We specify that our TSP tour can only enter or leave each box through a small number of special points called "portals". We'll characterize the way that a TSP tour travels through the box in terms of the way it enters and exits the various portals and prove that there are a limited number of ways that it can do this.
     - **Step 4: Dynamic Program & Conquer.** Arora's write-up describes the dynamic
       agree
       l TSP
       ng out
       nstant
       n time
       ations
       ve the
       ore all
       these solutions in a DP table. Then we solve the problem for each level of larger boxes by trying all 4-tuples of compatible solutions for its four sub-boxes.

Heavily scaffolded technical reading
(Red team: read to present)
(Blue team: skim for active listening)

Example Course of Study

CSCI 145: Advanced Topics in Algorithms Week 5
Approximation Algorithms: When "Good Enough" is Good Enough

2. Even teams: implement the PTAS for the Knapsack problem described in Kleinberg-Tardos 11.8!

   - You're welcome to use any programming language or environment you'd like.
   - Your program should allow you to specify the following parameters: ε (the small constant that determines the approximation factor), W (maximum capacity), n (the number of inputs), and n items of the form (w_i, v_i), where w is the weight and v is the value.
   - Your program should round the inputs, solve the problem using a 2D dynamic programming table, and output a valid solution that is guaranteed to be within a $(1+\varepsilon)$-factor of optimal.
   - Note: there's a bug in the Knapsack PTAS pseudocode given on pp. 648-649 in Kleinberg-Tardos. The line that says "$M[i, V] = w_i + M[i-1, V]$" should say "$M[i, V] = w_i + M[i-1, V-v_i]$". Thanks to the spring 2026 section of CSCI 145 for pointing this out!
   - Try generating some large random instances and evaluating them with different approximation factors. Compare the results to each other and to the optimal solution (which ard dynamic programming approach). How good of an ram produce in practice?

Implementation with lightweight specs (Blue team)

Scaffolded presentation instructions

3 Presentations

1. Odd teams: your primary goal will be to present Arora's PTAS for the Euclidean TSP problem.
   - Make sure you cover the background first, including the problem setting and how the parameter ε creates a trade-off between runtime and quality
   - This is a tough topic, and you may not come to class on Monday fully understanding every bit. I suggest designing your presentation in a "top-down" manner: give a high-level explanation and then recursively explain each part of the algorithm in more detail to the extent possible.
   - Figures are strongly encouraged! You're welcome to copy from the readings and/or make your own.
   - Expected presentation time: up to 40 minutes.
2. Even teams: your primary goal will be to explain the PTAS for Subset Sum and show off your code! Expected presentation time: up to 15 minutes.

Red team: must-do's

Red team: fallbacks

Blue team: Presentation-based evaluation shifts emphasis to explanation and understanding rather than output artifact

How did it go? A few reflections

- Student surveys:
  - Strong increases in perceived ability to learn algorithmic content independently, explain
  - Identified collaboration and presentation prep as most impactful on learning
- Possible to cover lots of content
  - Feasible to include difficult/technical material, even if few teaching resources exist
  - Allows students to discover and dive into topic interests
- Students may learn different topics in varying depth. An “open-ended” learning model has pros and cons.
  - Well-suited for topics, seminar-style courses
  - Well-suited when students can experience intrinsic motivation
  - Well suited when prioritizing process over product
- Energetic, collaborative, and fun classroom experience

Open questions and works in progress:

- Centering student agency and responsibility over their own learning requires students who are present and invested.
- Evaluation: transparency, fairness, scalability.

I’ll stop there. Please reach out for details, data, clarifications, modifications. Thank you!