Cost and Quality Trade-Offs in Crowdsourcing

Anja Gruenheid, Donald Kossmann, Besmira Nushi
Introduction

What am I going to talk about?
- Crowdsourcing (who’d have thought?)
- Quality and cost
- Sorting algorithms

What do we want to achieve with this project?
- Revisit traditional algorithms and complexity in the presence of error
Problem

Order the five adjectives ‘neutral’, ‘painful’, ‘good’, ‘excellent’, and ‘bad’ according to their positivity

What we expect:

excellent > good > neutral > bad > painful
Sorting with Noise

Problem

Order the five adjectives ‘neutral’, ‘painful’, ‘good’, ‘excellent’, and ‘bad’ according to their positivity

What we expect:

excellent > good > neutral > bad > painful

What impact has wrong partial information?
QuickSort

- Cost is in $O(n \log(n))$
- QuickSort leverages transitivity
QuickSort

- Cost is in $O(n \log(n))$
- QuickSort leverages transitivity
QuickSort

- Cost is in $O(n \log(n))$
- QuickSort leverages transitivity
QuickSort

- Cost is in $O(n \log(n))$
- QuickSort leverages transitivity
QuickSort

- Cost is in $O(n \log(n))$
- QuickSort leverages transitivity
  - faulty comparisons get propagated
QuickSort

- Cost is in $O(n \log(n))$
- QuickSort leverages transitivity

- faulty comparisons get propagated
- transitivity makes QuickSort less robust
BubbleSort

run 1: neutral painful good excellent bad

- BubbleSort is input sensitive and makes local comparisons
- Cost is in $O(n^2)$ in the worst case, in the best case it is $O(n)$
BubbleSort

- BubbleSort is input sensitive and makes local comparisons
- Cost is in $O(n^2)$ in the worst case, in the best case it is $O(n)$
BubbleSort

- BubbleSort is input sensitive and makes local comparisons
- Cost is in $O(n^2)$ in the worst case, in the best case it is $O(n)$
BubbleSort

- BubbleSort is input sensitive and makes local comparisons
- Cost is in $O(n^2)$ in the worst case, in the best case it is $O(n)$
BubbleSort

- BubbleSort is input sensitive and makes local comparisons
- Cost is in $O(n^2)$ in the worst case, in the best case it is $O(n)$
BubbleSort

- BubbleSort is input sensitive and makes local comparisons.
- Cost is in $O(n^2)$ in the worst case, in the best case it is $O(n)$.
BubbleSort

run 1: excellent neutral painful → good bad

run 2: excellent neutral → good bad painful

- BubbleSort is input sensitive and makes local comparisons
- Cost is in $O(n^2)$ in the worst case, in the best case it is $O(n)$

BubbleSort is less dependent on transitivity
BubbleSort

- BubbleSort is input sensitive and makes local comparisons
- Cost is in $O(n^2)$ in the worst case, in the best case it is $O(n)$

- BubbleSort is less dependent on transitivity
- Faulty comparisons can be compensated through multiple runs
Input Sensitivity

When is an algorithm input sensitive?

If it can improve its output with an improved input.

Example:

- QuickSort makes **global** comparisons (pivot elements), the input order is neglected
- BubbleSort makes **local** comparisons (bubbling), the input order becomes relevant
QuickSort vs BubbleSort

Setup:
- 1000 items
- error rate = 20%
- 500 runs
- majority = 3

- BubbleSort has a worse initial result quality but improves its output over time
- QuickSort has constant result quality
QuickSort vs BubbleSort I

Setup:
- 1000 items
- error rate = 20%
- 500 runs
- majority = 3

- BubbleSort has a worse initial result quality but improves its output over time
- QuickSort has constant result quality

if your goal is good quality, choose BubbleSort
QuickSort vs BubbleSort II

Setup:

- 1000 items
- error rate = 20%
- 500 runs
- majority = 3

- BubbleSort benefits from more accurate input cost-wise
- QuickSort has constant average cost
QuickSort vs BubbleSort III

What do we take from this comparison?

- *Cost is not the only factor* that is important to judge a sorting algorithm by in a noisy setting
  - If you want low cost with reasonable quality, choose QuickSort
- *Quality can be improved* over time with additional cost
  - If you want good quality, choose BubbleSort
QuickSort vs BubbleSort III

What do we take from this comparison?

- **Cost is not the only factor** that is important to judge a sorting algorithm by in a noisy setting
  - If you want low cost with reasonable quality, choose QuickSort
- **Quality can be improved** over time with additional cost
  - If you want good quality, choose BubbleSort

Do we need to re-evaluate our traditional complexity theory in the context of noise?
Cost vs Quality I

What do we aim for?

(1) A function that estimates the budget given a certain input algorithm, quality requirements, and error rate.

(2) A function that estimates the quality given a certain input algorithm, quality requirements, and error rate.
Cost vs Quality II

What do we aim for?

A way to compare different algorithms for one problem independent of the problem.
Cost vs Quality III

Setup:
- 1000 items
- error rate = 20%
- 500 runs
- majority = 3

![Diagram showing the comparison between BubbleSort and QuickSort based on cost and quality.](image-url)
Cost vs Quality III

Setup:
- 1000 items
- error rate = 20%
- 500 runs
- majority = 3

how can we measure the actual trade-off?

BubbleSort?
QuickSort?
Cost vs Quality III

Setup:
- 1000 items
- error rate = 20%
- 500 runs
- majority = 3

Vision: Quality and cost-based complexity theory

how can we measure the actual trade-off?

BubbleSort?
QuickSort?

cost

complete algorithms
Future Work

(1) Define a model that takes the error rate and budget/quality specifications into consideration when evaluating an algorithm

(2) Revisit established algorithms in a noisy environment
Related Work

Quality Guarantees
- Roy et al.: Using the Crowd for Top-k and Group-by Queries
- Venetis et al.: Max Algorithms in Crowdsourcing Environments

Social Sciences
- Schulze: A new Monotonic, Clone-Independent, Reversal Symmetric and Condorcet-Consistent Single-Winner Election Method

Machine Learning
- Busse et al.: The Information Content in Sorting Algorithms

Crowdsourcing Algorithms
- Entity Resolution
  - Question Selection for Crowd Entity Resolution (Stanford)
  - Leveraging Transitive Relations for Crowdsourced Joins (Berkeley, Brown)
- Data Retrieval
  - Crowdsourced Enumeration Queries (Berkeley, Brown)
- Data Mining
  - Crowd Mining (Tel Aviv)

Crowdsourcing Complexity
- Ahn: Human Computation
Thanks for listening!
Any questions?
Why is BubbleSort so cost-effective?

- We store answers within one run
- Wrong comparisons act as barriers which reduce the bubbling of items

But: Barriers shift over runs which increases quality