UIC UNIVERSITY OF ILLINOIS CHICAGO

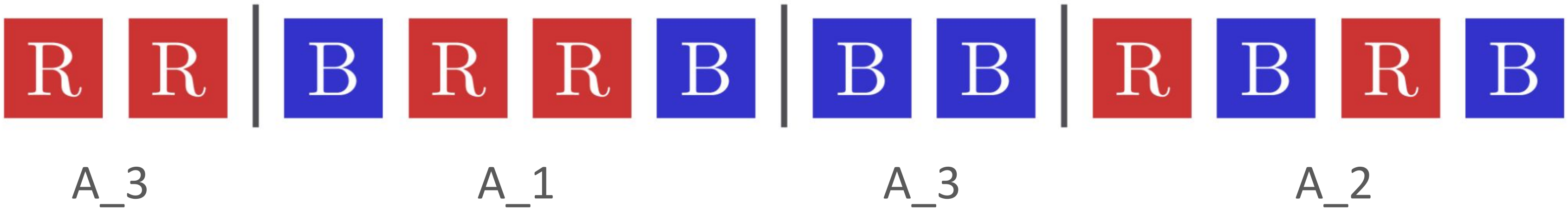
InDeX Lab
Innovative
Data
eXploration
Laboratory

Dynamic Necklace Splitting

Rishi Advani, Abolfazl Asudeh, Mohsen Dehghankar, Stavros Sintos

University of Illinois Chicago

What is necklace splitting?



We want to cut the necklace such that the resulting intervals can be fairly distributed between the agents: each agent should receive the same number of red beads and the same number of blue beads.

Applications

- Fair hash maps: partition data (beads) across buckets (agents) such that each bucket receives an equal share of each attribute type (Shahbazi et al., SIGMOD '24)
- Load-balancing: spread tasks (beads) across servers (agents) where tasks need to be completed in a specified order
- Bucketization for fairness in ML: training and test data should have similar minority representation

Static necklace splitting

- m beads (m_1 red, m_2 blue)
- k agents
- Goal: efficiently cut the necklace in as few places as possible such that the resulting intervals can be distributed such that each agent receives m_1/k red beads and m_2/k blue beads
- Optimal solution: $2(k-1)$ cuts (Alon and Graur, ICALP '21)

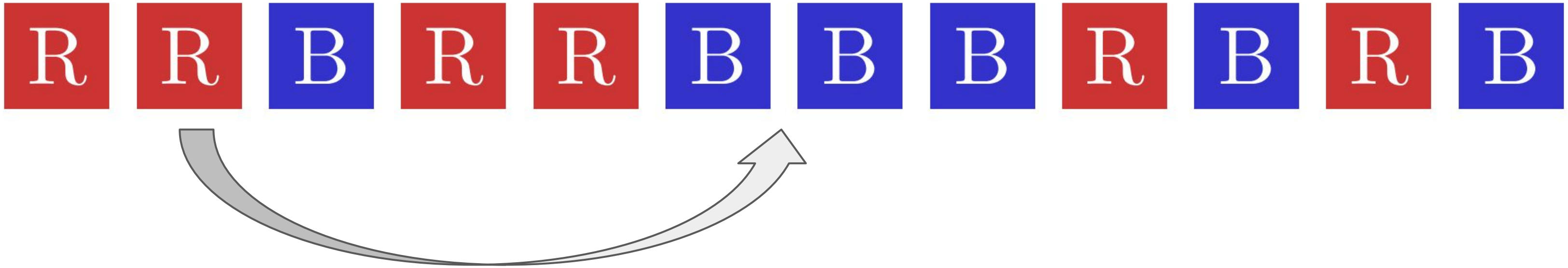
Static necklace splitting (cont.)

Offline algorithm (Shahbazi et al., SIGMOD '24):

- Consider range of length m/k
- Move this range bead by bead until it contains exactly m_1/k red beads
- Allocate the range to the current agent
- Consider the remaining beads and the next agent — repeat

Dynamic setting

- Relocation: move a bead to an arbitrary position in the necklace
- Insertion: insert *k* beads of the same color at arbitrary positions in the necklace
- Deletion: remove *k* beads of the same color from the necklace



Key results

Exact algorithm:

- 2(k-1) cuts
- Running time: linear in m and k

Approximate algorithm:

- Approximately fair w.h.p.
- 2(k-1) cuts
- Running time: polylogarithmic in m for fixed k

Exact algorithm: relocation

We will start by designing an algorithm for relocation, and later we will extend it to handle insertion/deletion.

Exact algorithm: adjacent positions

If we need to relocate a bead to an adjacent position:

- Remove all cuts between the agents corresponding to the starting and ending positions of the bead
- Rerun the offline algorithm on those two agents

Exact algorithm: nonadjacent positions

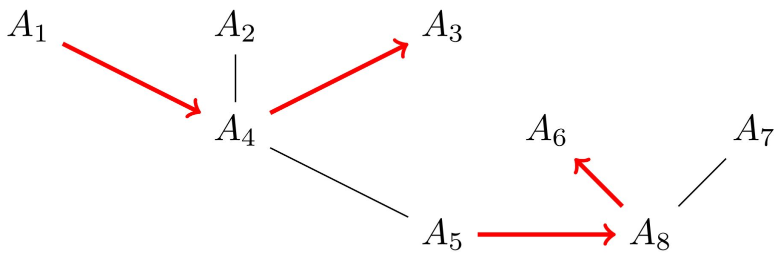
If we need to relocate a bead to a nonadjacent position:

- Model the set of agents as a graph with edges between agents with shared boundaries
- Find the shortest path between the agents corresponding to the starting and ending positions of the bead
- Remove all cuts between the agents on the path
- Rerun the offline algorithm on those agents

Exact algorithm: batch updates

Finally, we can perform batch updates to increase efficiency:

- Construct a flow network:
  - Agents to give beads are sources
  - Agents to receive beads are sinks
- Find a max flow with the number of active nodes minimized
  - This problem is NP-hard
  - 2-approximate solution on a spanning tree
- Remove all cuts between active agents
- Rerun the offline algorithm



Running time: linear in *m* and *k*

Exact algorithm: insertion/deletion

Now that we have an algorithm for relocation, we can use it to solve insertion/deletion.

- Insertion: give each agent a new bead, and then relocate the beads to the desired positions
- Deletion: remove a bead from each agent, and then relocate the beads to be deleted to the empty spots

Approximate algorithm

- Construct a binary search tree over the necklace
- To perform a dynamic update, we relocate/insert/delete the relevant beads in the tree in O(log *m*) time
- To generate the set of cuts on demand:
  - Sample subsets of beads and run the offline algorithm on those subsets
  - This gives us an approximately fair solution in polylogarithmic time (for fixed *k*)

Future work

- Generalizing results to greater than two colors
- Designing an approximate algorithm that is subexponential in k
- Experiments

Takeaways

- Applications: fair hash maps, load-balancing, and bucketization
- Exact solution with optimal 2(*k*-1) cuts generated in linear time
- Approximately fair solution generated in polylogarithmic time for fixed *k*
- Open directions: generalizing to more colors, improving *k*-dependence, experiments

link to paper

website: rishi-advani.com
email: radvani2@uic.edu

This work has been supported in parts by the UIC University Fellowship and the National Science Foundation (Grant No. 2348919).