Oblivious tight compaction in O(n) time with smaller constant

Rafail Ostrovsky Sam Dittmer

Tight compaction

Given an array of objects colored blue and red:

412 736 654 198 552 338 797 603

Re-arrange so that all blue elements are at the front of the array:

603 552 654 338 412 736 198 797

Oblivious tight compaction

- Client has O(1) local memory blocks of size O(log n) bits
- Remote server has O(n) memory blocks of size O(log n) bits in array A
- Client stores data on server encrypted with their private key

- Client sends a *data-independent* sequence of memory accesses to the server (either deterministic or randomly generated)
- For each access at index i, server sends encrypted element A[i] to client. Client performs some local calculation and responds with a new A[i].
- Result is encrypted compacted A (either guaranteed, or with high probability)

Applications

- Fast primitive to delete or divide parts of an encrypted server
- Building block in multiple ORAM schemes
- Related to network and routing optimization problems

Existing Results

- Ω(n log n): Lin, Shi, Xie ’19 for *stable* compaction algorithm

- O(n log log n): Mitchell, Zimmerman ‘14, Lin, Shi, Xie ‘19

- O(n): OptORAMa, Asharov, Komargodski, Lin, Nayak, Peserico, Shi, ‘20
  - Deterministic tight compaction in O(n) time.
  - Leads to first asymptotically optimal ORAM construction with O(log n) overhead.

Our Contribution

Theorem: *There is a randomized tight compaction protocol that runs in time 23,913n + o(n), such that the server learns nothing from the pattern of client's memory accesses, and with all but negligible probability, the array is tightly compacted at the end of the protocol.*

Comparison to OptORAMa

	OptORAMa	**Ours**
Deterministic	Yes	No
Density reduction	Simple swaps	Airlock swaps
Framework	Shuffle into super-small bins and store metadata	Same
Expander graphs	Bi-partite Jimbo-Maruoka	Random regular graph and Margulis graph
Self-routing algorithm	Linear superconcentrators	Majority bootstrap percolation
Runtime	$>2^{111}n$	23,913n + o(n)

Warm-up: Random swaps, $O(n^2)$ algorithm

412 736 654 198 552 338 797 603

Swap when we have red on the left, blue on the right

338 736 654 198 552 412 797 603

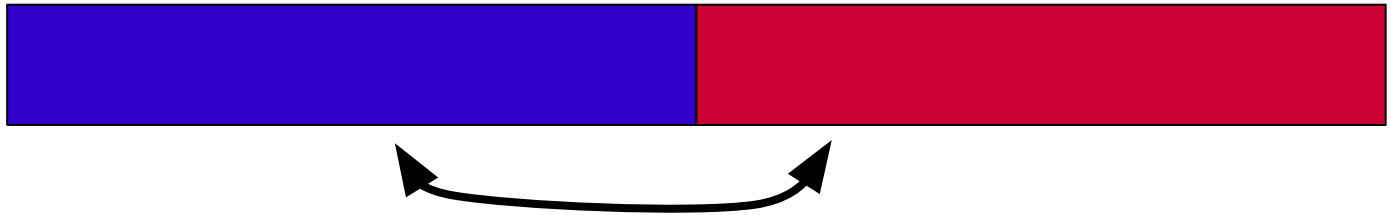
Otherwise, do nothing

338 736 654 198 552 412 797 603

While density of out-of-place balls is high, it takes O(1) time to find a red-blue pair we can swap. But the final swap alone takes $n^2/4$ time on average

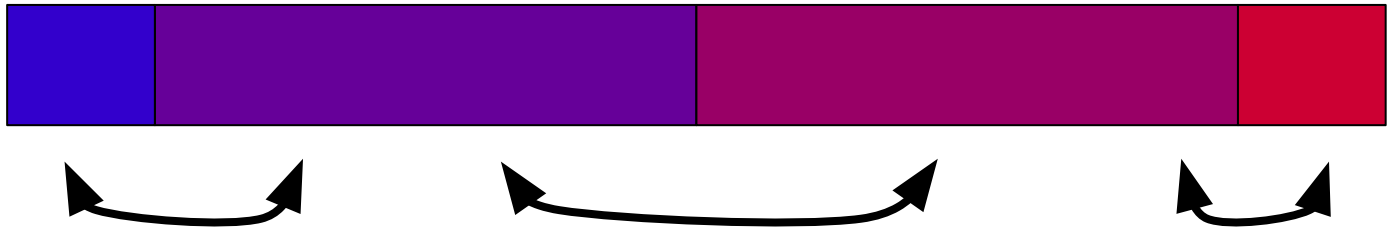
Density Reduction

Simple swaps, via random selection or bipartite expander:



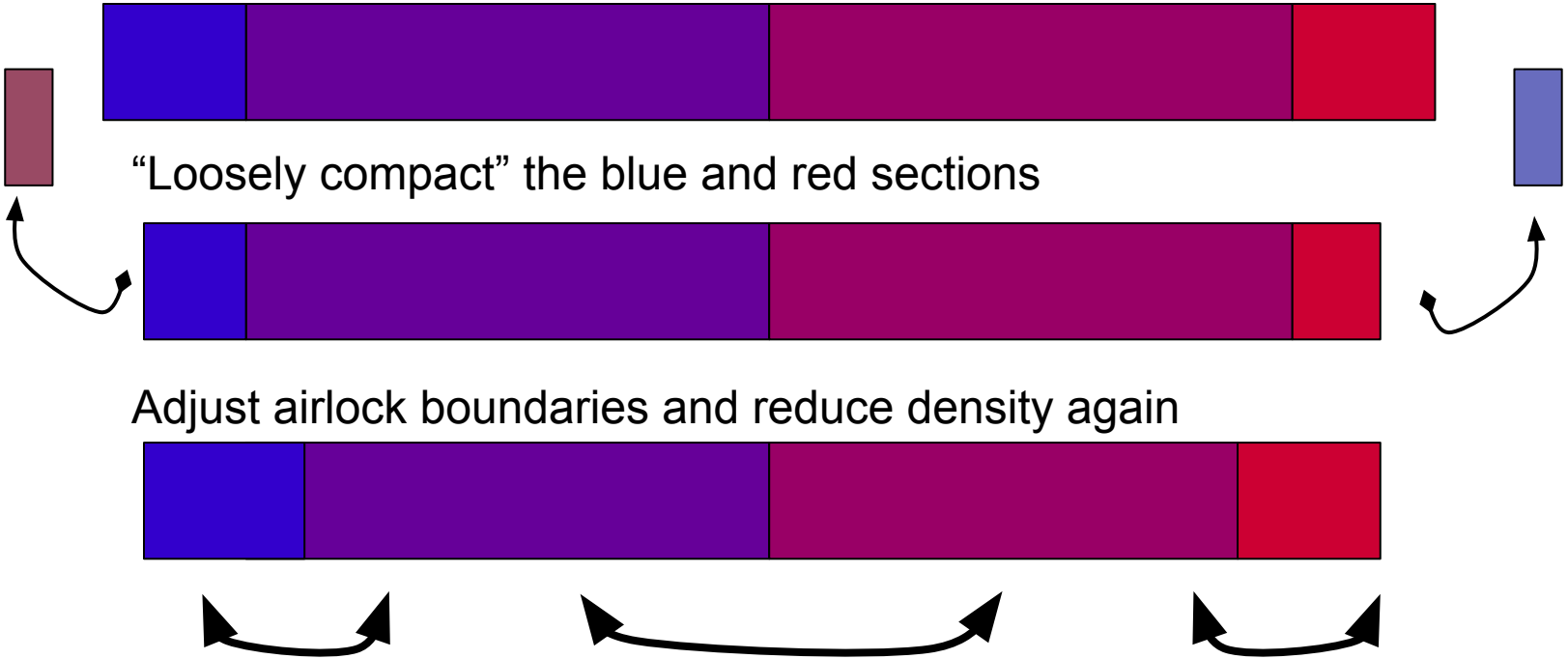
$$\int_{\rho}^{1} \frac{d\rho}{\rho^2}$$

Airlock swaps



$$\int_{\rho}^{1} \frac{d\rho}{\alpha\rho}$$

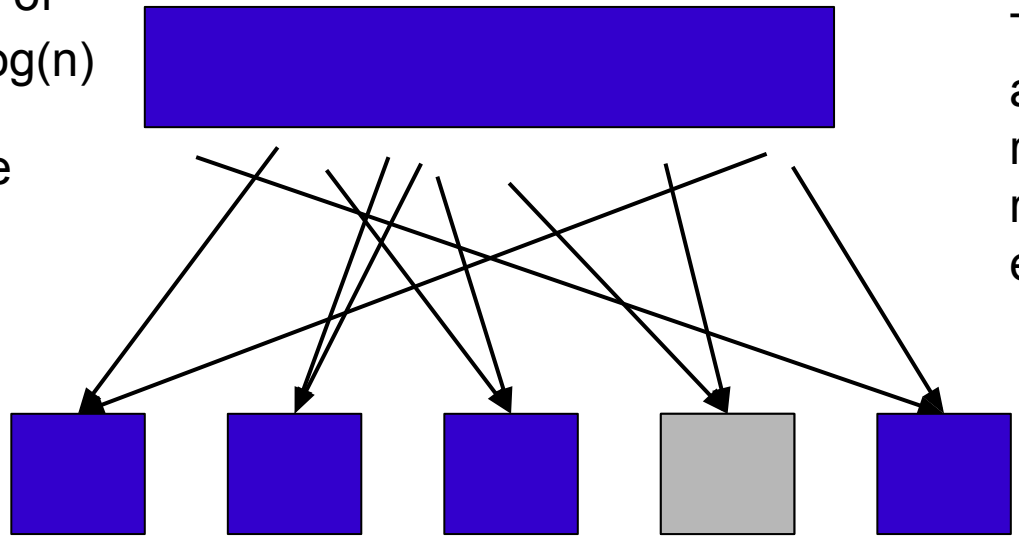
Compressing airlocks



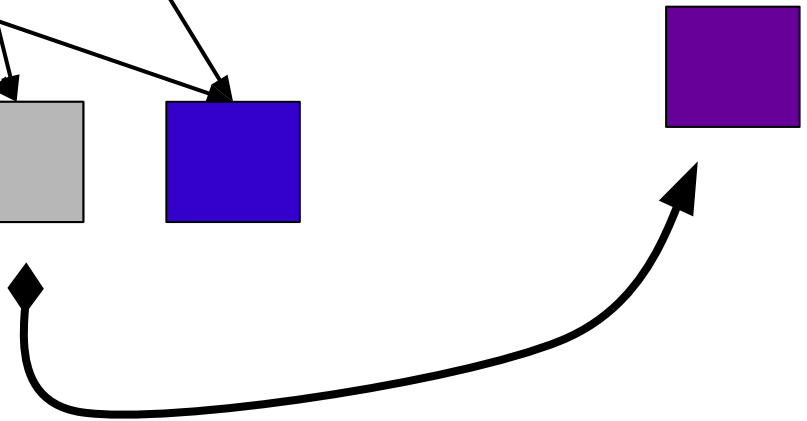
Loose compaction: Small bins

Shuffle into bins of size log(n)/log log(n)

Most bins will be mostly blue



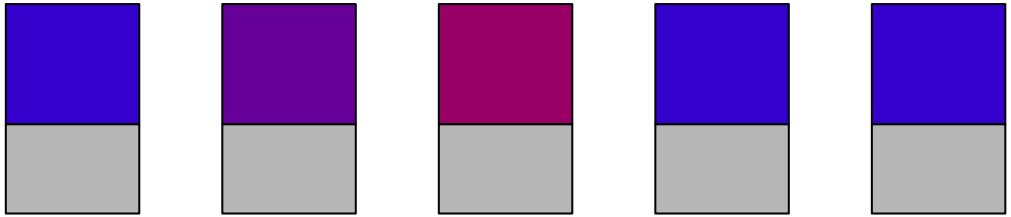
Throw a few bins that are “too red” into a runoff array, and replace with dummy elements.



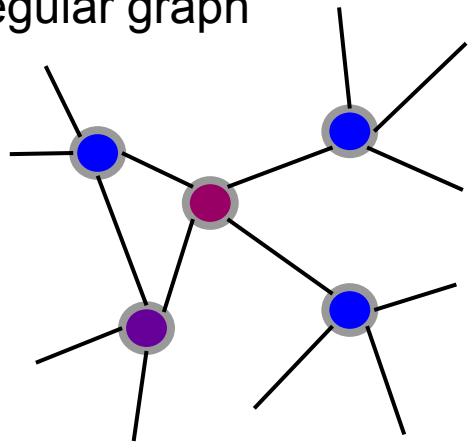
Very small bins

Divide each bin up into small bins (of constant size, d+1), and extend each bin by d dummy elements.

Now we'll have a lot more red.



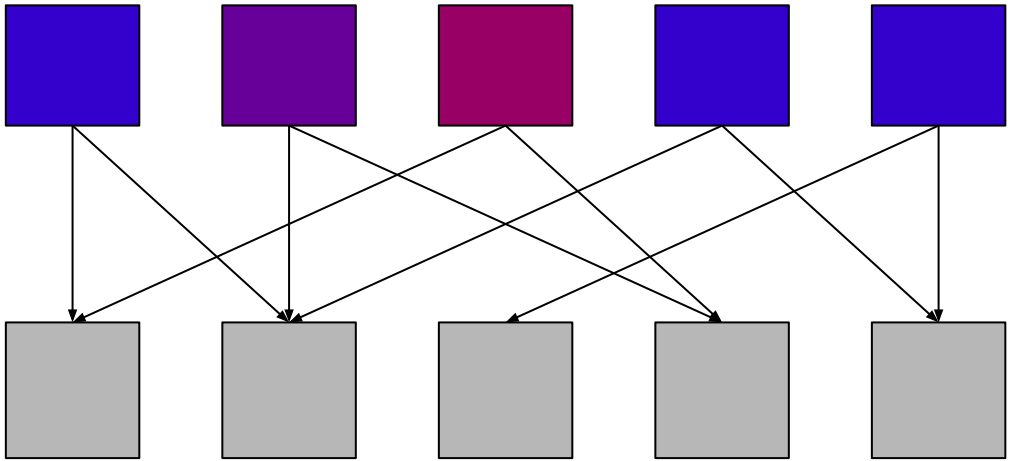
Map bins onto a random d-regular graph



The edge set can be written in O(1) memory blocks

The problem of non-bipartite graphs

Self-routing Superconcentrators

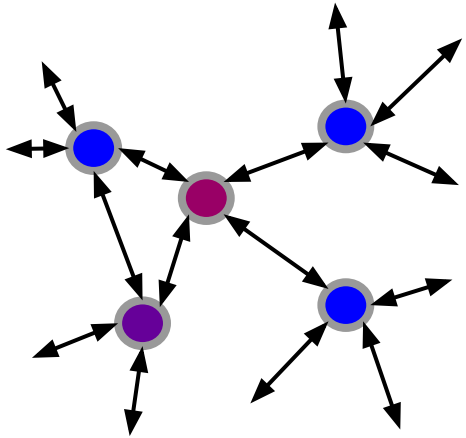


We need G to be an expander of large degree.

Then, for each gray box connected to few purple boxes, move red balls along arrows. Repeat. (Pippenger ‘96)

Assign edges first, “non-obliviously”. Then move.

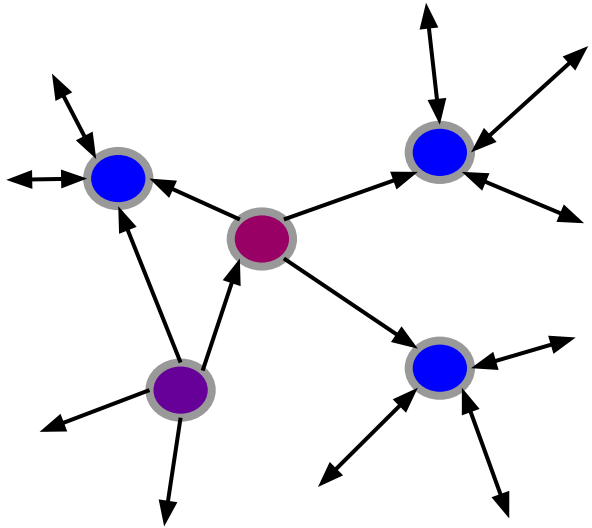
???



- Which way do arrows go?
- What about a red cluster?
- How small can we make d?

Majority bootstrap percolation

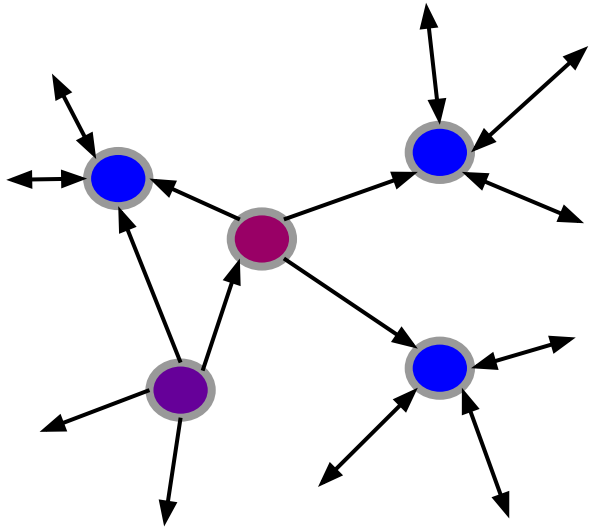
- Blue nodes are those with zero red balls
- Direct arrows so that more arrows point out than in for each red or purple node
- For each arrow, swap a red ball with a blue or gray ball
- Gives d/(d+1) loose compaction regardless of order arrows are visited



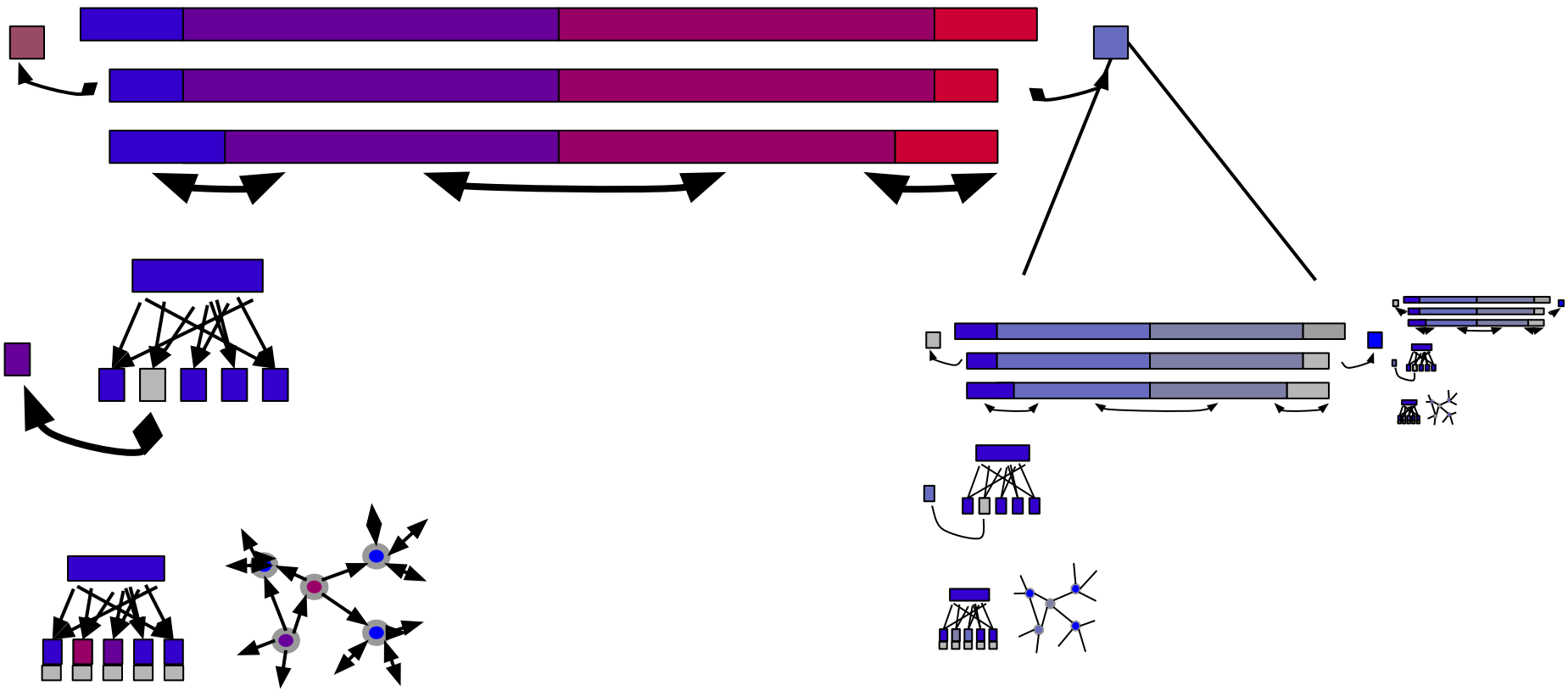
Majority bootstrap percolation

Make a single linear scan. Assign arrows when majority of neighbors are blue, and re-check neighbors. When does this work?

- Margulis graph. Sparse density, d=4
- Random regular graph that is a good expander: d=13, density = 0.005307



The whole algorithm on one slide



Closing comments

- New result in percolation theory: Majority bootstrap percolation reaches all nodes *always*, for d > 12 (except d=14) and sufficiently small density of inactive nodes

- Our algorithm is still not practical - the straightforward recursive algorithm takes ½ n log n time, which beats 23,913 n except for very large n

- Open problem: Can this algorithm be de-randomized to give an algorithm that succeeds with probability 1 without a much larger constant?