From Discrepancy to Majority

David Eppstein and Daniel S. Hirschberg

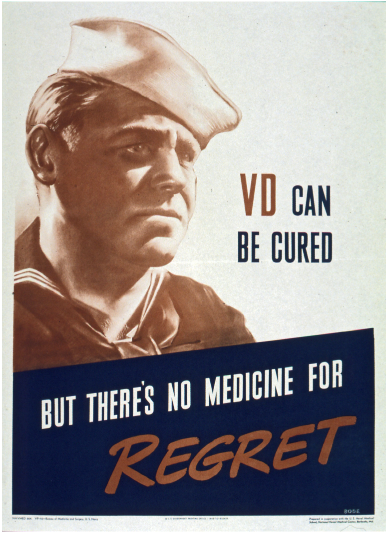
12th Latin American Theoretical Informatics Symposium
(LATIN 2016)
Ensenada, Mexico, April 2016

Fault diagnosis

Test whether a system is behaving correctly (and if not find the problem) using few tests

Classical example: combinatorial group testing

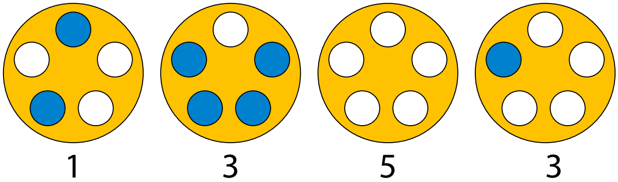
Developed in World War II to identify sick army recruits by applying expensive blood tests to few mixtures of many blood samples



Fault diagnosis of distributed systems

As modeled by De Marco and Kranakis [2015]:

- Majority of processors are assumed to be non-faulty
- Can test k-tuples of processors
- Each test returns *discrepancy* (difference between numbers of processors with each of two answers) but not which processors had which answers
- Goal: identify a non-faulty processor (one that produced a majority answer)



Finding the majority from pairwise comparisons

The (well-studied) $k = 2$ case of De Marco and Kranakis [2015]

Can be solved by the following steps:

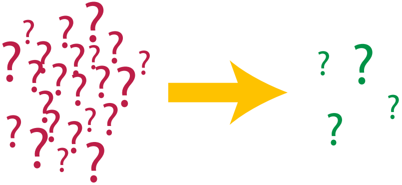
- Pair up items and test each pair
- Eliminate both items in mismatched pairs, keep one item from each matched pair
- Recurse on remaining items and return their majority (if there is one)
- If not, and n is odd, return the left-over item

Total number of queries: $n - O(\log n)$

New results

We can find a majority element in $n/\lfloor \frac{k}{2} \rfloor + O(k)$ queries

Best previous upper bound was $n - k + k^2/2$

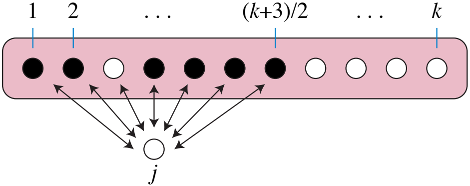


We also improve the lower bound
from $n/k - o(n)$ to $n/(k-1) - o(n)$

showing that the upper bound is optimal to within
a factor of $\frac{k-1}{\lfloor k/2 \rfloor} + o(1) \approx 2$ for all k

Step 1: Find an element in the minority of $[1, k]$

- Test the k-tuple $[1, k]$
- Choose $j > k$, form $(k+3)/2$ k-tuples by swapping j with an element of $[1, k]$, and test each k-tuple
- If all tests give discrepancy ≤ 1, j must be in the minority

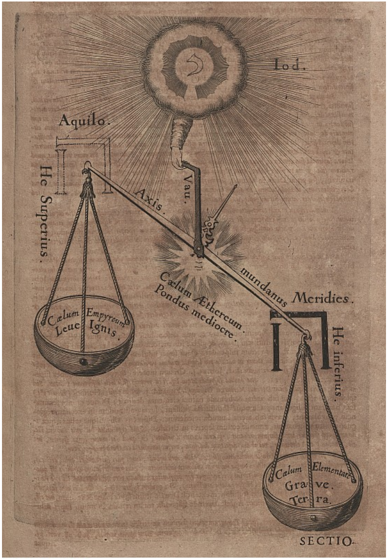


- (If not, we have already found a high-discrepancy k-tuple and can skip to step 3)

Step 2: Find an unbalanced k-tuple

Unbalanced: not evenly split, i.e. discrepancy is > 1

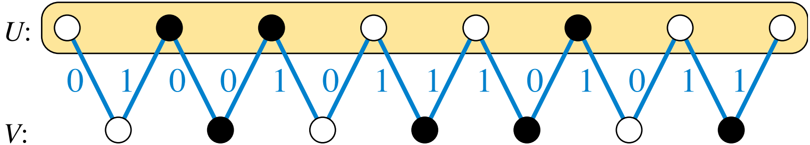
- Use Step 1 to find $j, j' > k$, both in the minority of $[1, k]$; set $Y = \{j, j'\}$
- Repeatedly double $|Y|$ (with $O(1)$ queries/step) preserving the property that $\mathrm{maj}(Y) \neq \mathrm{maj}([1, k])$, until $|Y| = k - 1$
- One of $\{1\} \cup Y$, $\{2\} \cup Y$, or $[3, k] \cup \{j, j'\}$ is unbalanced



Quelle: Deutsche Fotothek

Step 3: Find a homogeneous k-tuple

Homogeneous: all elements equal to each other



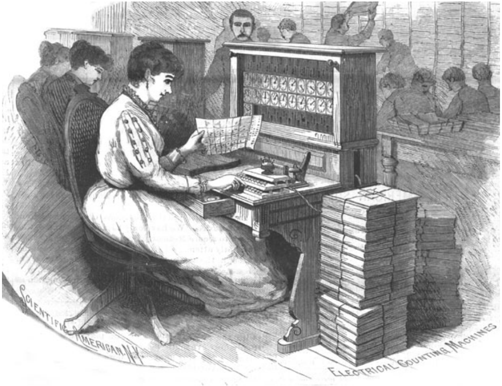
Form a path alternating between the unbalanced k-tuple and $k-1$ other elements

For each edge, replace the unbalanced k-tuple element by its neighbor, and test the resulting k-tuple to determine whether the endpoints of each edge are equal (0) or unequal (1)

Use the results to 2-color of the path and return a k-tuple of elements from the majority color class

Step 4: Calculate the discrepancy of $[1, n]$

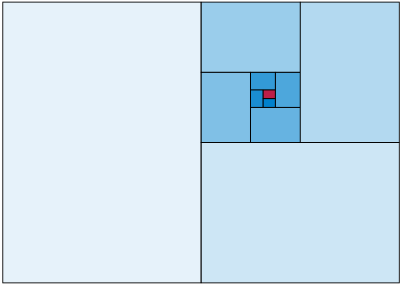
- Partition the input (outside the homogeneous k-tuple) into $\lfloor k/2 \rfloor$-tuples
- Use a single test per $\lfloor k/2 \rfloor$-tuple to count how many of its items are equal to the items in the homogeneous tuple
- Sum the results



Step 5: Find a majority element

If the homogeneous k-tuple is in the majority, choose any of its elements. Otherwise:

- Find a $\lfloor k/2 \rfloor$-tuple that contains an item unequal to the homogeneous k-tuple
- Repeatedly divide its size in two (preserving the property that it contains a majority element), with a single test per subdivision, until only one item remains



Analysis

- Step 1: $O(k)$ tests
- Step 2: $O(\log k)$ tests
- Step 3: $O(k)$ tests
- Step 4: $\left\lceil \frac{n-k}{\lfloor k/2 \rfloor} \right\rceil$ tests
- Step 5: $O(\log k)$ tests

Conclusions

New and nearly-tight solution to the problem of finding a majority element by counting (discrepancy) queries

To reduce the gap between upper and lower bounds, we probably need stronger lower bounds

Steps 1 and 2 (finding an unbalanced query) take
$O(1)$ queries when $k = 2 \bmod 4$, and
$O(\log k)$ queries when k is even,
but $O(k)$ when k is odd – can this be improved?

References

Gianluca De Marco and Evangelos Kranakis. Searching for majority with k-tuple queries. *Discrete Math. Algorithms Appl.*, 7(2):1550009, 2015. doi: 10.1142/S1793830915500093.