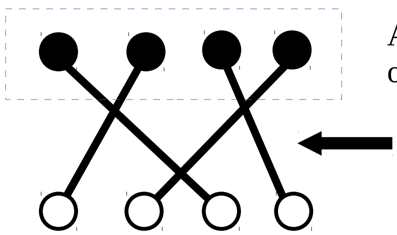
Permutation Intersections

by Sven Nilsen, 2020

In this paper I show how to count the intersections of function permutations.

The number of intersections, or inversions, in a function permutation uses the following intuition:



In the figure above, the number of intersecting points is `3`.

The minimum amount of swaps required to retrace the permutation back to `id`, is the same number as the intersecting points. Therefore, one can efficiently compute the number of intersections in `O(N log N)` time complexity, but at some memory cost to keep track of swaps during retracing:

```
/// Computes the intersections of a permutation.
pub fn intersections(a: Vec<usize>) -> usize {
   let mut b = a.clone();
   let mut count = 0;
   for i in 0..b.len() {
       while b[i] != i {
           count += 1;
           let j = b[i];
           b.swap(i, j)
       }
   }
   count
}
```

The number of intersections plus number of cycles is equal to the sum of the length of cycle orbits.

intersections(f) + |cycles(f)| = ∑ c : cycles(f) { orbit_len(c) }

The parity of a permutation is the `even` property of intersections:

f : [intersections] [even] true   "even" permutation
f : [intersections] [even] false   "odd" permutation