Obtaining additional orderings for viable attacks on ATs

June 25, 2020

1 Order Query Algorithm

As the paper discussed, a solution consists of sets of BASs. The BASs in these sets can be permutated to obtain differnt orderings. If we index these additional orderings, we can query the K'th ordering using the discussed algorithm.

First, we transform the problem in multiple smaller problems. This step is similar to decimal to binary conversion. To illustrate this, we consider then decimal number 19. To convert is we divide by bases of two, and continue with the remainder. With (x, y) we mean that x is the division result and y is the remainder. To convert 19 we consider the following computations:

1. 19 / 16 = (1, 3)
2. 3 / 8 = (0, 3)
3. 3 / 4 = (0, 3)
4. 3 / 2 = (1, 1)
5. 1 / 1 = (1, 0)

This division results 10011 is 19 expressed in binary. Similarly, we use the same approach in the algorithm. However, instead of bases we consider possible permutations in the following sets which can be seen in table 1. {a,b} has 12 as the sets of {c,d,e} and {f,g} have 12 permutations together. The permutations of a single set is size!. These numbers for the basis for our conversion. Table 2 employs a similar approach, and results in 231, which means we need the 2nd permutation of {a,b}, the 3rd permutation of {c,d,e} and the 1st permutation of {f,g}. The permutations are 0-indexed with. Concatenating these permutations of the sets gives us the result.

In the second step, we need to obtain the K'th permutation of a set X. If we consider a set of size n, it has a total of n! permutations. For each of the elements we have (n-1)! permutations with that element in the first position. As we have n elements, this leads to n*(n-1)! = n! permutations. If we consider the set X = {c,d,e}, we state that permutations [0, 2!) start with c, [2!, 2*2!)

Table 1: Precomputation for set index conversion

Total	**{a,b}**	**{c,d,e}**	**{f,g}**
2!3!2!	2!3!	2!	1
72	12	2	1

Table 2: Conversion to indices of sets for permutation 31

Remainder Division	**Result**
31 / 12	(2, 7)
7 / 2	(3, 1)
1 / 1	(1, 0)

start with d, and [2*2!, 3*2!) start with e. Thus, the index of first element in the permutation is K/(size-1)!. If K = 3, we have X[3/2!] = X[1] = d. Now we can remove d from the set, and take the remainder of the division as the new K, which is 1 (3 % 2!). By repeating this method, we keep retrieving and deleting from a list until we have obtained permutation. However, as data structures perform either retrieval or deletion in O(n), there is a need for a custom data structure to support these operations.

If have the set X with size n, we create a set Y with [0,1,2,3...n] where X[Y[j] = i] = i'th BAS in X. For retrieval of the i'th BAS in X, we binary search for the left most entry Y[j] = i. After removal of the i'th index in X, indices of each BAS [i, n) decreases by 1. While increasing a range by a certain value might seem O(n), by storing Y in a lazy segment tree this can be optimized to O(log(n)). Retrieving a value from Y becomes O(log(n)) aswell in this case. Thus, deletion can be done in O(log(n)), while the binary search takes $O(log(n)$, it requires a retrieval at each step making it $O(log(n)^2)$. Thus, the algorithm scales linearithmicly with the amount of BASs in the solutions.

As an example, table 3 provides an example execution. Here we query the 3rd permutation of a set {c,d,e}. Y is initialized to [0,1,2], and we use the specified formula to obtain the index of the BAS to take first which leads to (1,1). X[1] = d, and the new K becomes 1, the remainder. The indices [1,2) in the set decrease which corresponds to d and e which result in Y = [0,0,1]. Again we have index 1, which corresponds to e now as X[2] = e. Finally, we search for the value of 0, and since we take the left most entry of 0 in Y, we have index 0. Thus X[0] = c is the last BAS. The result for this set becomes {d,e,c}.

Table 3: Permutation of a single set

K	**Y**	**Size**	**k/(size-1)!**	**Index**	**BAS**
3	[0,1,2]	3	(1,1)	1	d
1	[0,0,1]	2	(1,0)	1	e
0	[0,0,0]	1	(0,0)	0	c