A Appendix

A.1 Almost universal set of hash function

An invertible matrix A is a sequence of vectors $v_1, \ldots, v_{2k}$ such that the i^{th} vector is not in the space spanned by the vectors $v_1, \ldots, v_{i-1}$. This vector space has size 2^{i-1} and the number of choices for v_i is $2^{2k} - 2^{i-1}$. Hence

$$|\mathcal{H}| = \prod_{i=0}^{2k-1} (2^{2k} - 2^i) = (2^{2k} - 1) \prod_{i=1}^{2k-1} (2^{2k} - 2^i) . \quad (1)$$

For a given pair (x, y), let N be the number of invertible matrices for which $Ax \equiv Ay \pmod{2^\ell}$, or equivalently $Az \equiv 0 \pmod{2^\ell}$ by setting $z = x - y$.

Let B be an invertible matrix such that $z = Be_1$, where $e_1 = (1, 0, \ldots, 0)$, and let $C = AB$. B can be constructed by setting the first column to z and choosing the remaining columns as above to make B invertible. Then the matrix C is invertible if and only if A is invertible and $Az = ABe_1 = Ce_1$. Hence, N is the number of invertible matrices C for which $Ce_1 \equiv 0 \pmod{2^\ell}$.

Therefore there is only $2^{2k-\ell} - 1$ choice for v_1, the first column of C, and the number of choices for $v_i, i > 1$, is unchanged. Hence

$$N = (2^{2k-\ell} - 1) \prod_{i=1}^{2k-1} (2^{2k} - 2^i) . \quad (2)$$

Provided that $2^{2k} \gg 1$ and $2^{2k-\ell} \gg 1$, the relation $N \approx |\mathcal{H}|/2^\ell$ holds.

The proportion of invertible matrices is

$$P_k = \frac{|\mathcal{H}|}{2^{2k}} = \prod_{i=1}^{2k} \left(1 - \frac{1}{2^i}\right) . \quad (3)$$

P_k is a decreasing sequence with a limit > 0.28. Hence, by random drawing, an invertible matrix will be found in an expected 4 steps.

A.2 Distributed locks to reduce contention when writing table to disk

When a thread fails to add a key into the hash table because the table is full, the hash table is written to disk and reinitialized. For data consistency, all threads must be prevented from making any updates to the hash table while it is written to disk. A reader-writer lock (e.g. POSIX's `pthread_rwlock`) would suffice. However, when the number of contentions is high, this performs very poorly.

Instead, we implement a distributed reader-writer lock where the frequent case (acquiring a read lock) is optimized as much as possible at the expense of the infrequent case (acquiring a write lock). Functionally, the distributed lock behaves to each thread i as a distinct reader-writer lock *rwlock*$_i$. For a thread to make an update to the hash table, it only needs to acquire a read lock of its own lock, *rwlock*$_i$. On the other hand, to write the hash table to disk, a thread i is required to acquire a write lock on all of the locks, *rwlock*$_j$ $\forall j$. In this scheme, the frequent case involves only acquiring a lock with no contention, which is fairly fast.

The implementation again uses the CAS operation instead of POSIX `pthread_rwlock` reader-writer locks. Each thread maintains a status variable which can have three states: FREE, INUSE, BLOCKED. The frequent non-contentious case is as follows: before an update, a CAS operation is made to change the status from FREE to INUSE. In case of success, the read lock is considered acquired and the thread can proceed with the update. After the update, a compare-and-swap operation is made to change the status from INUSE to FREE. In case of success, the read lock is considered released and the thread is done. In this frequent non-contentious case our implementation incurs only the cost of two compare-and-swap operations.

A thread that discovers a full hash table when it tries to add a key will set the status variable of every other thread to BLOCKED. Using a condition variable, it will then wait for every thread that was in the INUSE state to finish their update, and then proceed to write the hash table to disk.

While the writing is occurring, every thread's status variable will be BLOCKED and any thread will fail in an attempt to to change its status from FREE to INUSE using the CAS operation. If this occurs, the thread waits for the writing of the hash to disk to be finished (its status changed from BLOCKED to FREE). If a thread fails to change its status from INUSE to FREE it notifies, using the condition variable, the thread that wants to write the hash to disk, that it is done with its update.

A.3 Impact of mer length on runtime.

Detailed running time of Jellyfish counting k-mers for different values of k on coverage 5x of the Turkey reads.