CSE 584A Class 23

Jeremy Buhler

April 18, 2018

1 Translating LSH on Sequences to Practice

Let's see how the LSH construction described last time works out in practice.

- First, let's set up a problem to solve.
- *Idea*: we have a set of queries of common length ℓ, and we want to find all occurrences of each query in a database S with at most R mismatches.
- We'll draw inspiration from word-counting approaches to phylogenetic assignment (e.g. Kraken, Wood and Salzberg 2014) and genetic distance estimation (e.g. AAF, Fan et al. 2015).
- Those tools use ℓ of 25-35 to balance sensitivity against false positives due to chance matches.
- $\ell = 30$, $R = 0$ would be one typical set of parameters for this problem.
- Suppose we want to extend these algorithms to work with approximate matches, so as to improve their sensitivity.
- $\ell = 39$, $R = 3$ has about the same overall rate of rate of chance matches in iid random DNA with equal base frequencies as $\ell = 30$, $R = 0$, so let's use these parameters.
- (The implied $P_1 \approx 0.923$.)
- We'll set the database size $n = |S| = 3 \times 10^9$, about the size of an assembled human genome (or a moderately deep-coverage set of reads for a smaller higher-eukaryote genome).

To set up an LSH problem, we also need a multiplier c to specify the set of distant objects to robustly reject.

- We assume that the most important source of extra work in a homology search problem is chance matches.
- Let $P_b(d, \ell)$ be the probability that a fixed ℓ-mer matches an iid random sequence of length ℓ with equal base freqs, to within at most d mismatches.

- P_b is the cumulative binomial tail probability

$$P_b(d, \ell) = \sum_{j=0}^{d} \binom{\ell}{j} \left(\frac{3}{4}\right)^j \left(\frac{1}{4}\right)^{\ell-j}.$$

- The expected number of chance matches in S to an ℓ-mer with at most cR mismatches is about $nP_b(cR, \ell)$.
- The larger we set c, the more such spurious matches LSH will "let through", and hence the more we'll have to process to find the desired (ℓ, R)-mismatches.
- We'll limit c so that $nP_b(cR, \ell) \leq 1$; that is, we'll ask LSH to hold the overall rate of chance matches we have to process below 1 per query.
- For $\ell = 39$ and $n = 3 \times 10^9$, this limit is achieved for $cR = 11$.
- (The implied $P_2 \approx 0.718$, and the implied $\rho \approx 0.242$.)

So what happens when we plug these parameters into the formulas for k and L?

- Assume as before that our error threshold $\delta = 0.01$.
- We get $k \geq 66$, and $L \geq 1368$!
- This seems pretty awful – more than 1000 hash functions, and each samples the ℓ-mer so densely as to repeat many positions with certainty.
- This seems remarkably impractical. Can we do better?

2 Relaxing Worst-Case Behavior of Distant Objects

- Why did LSH do so badly for what seemed like a reasonable set of parameters?
- It makes worse-case assumption regarding how the distant objects in the database, i.e. those at distance $\geq cR$, are distributed.
- Because we picked k so that $nP_2^k \leq 1$, LSH would reject nearly all chance matches *even if* they were all at distance precisely cR from the query.
- But this is ridiculously pessimistic for real DNA sequence.
- Instead, we'll assume that chance matches to the query arise due to "random" unrelated DNA (as we did in choosing c), and then pick k so that the expected number of such matches is small under this assumption.
- More precisely, we assume (as is typically the case in stringology) that DNA unrelated to the query is an iid sequence with equal base frequencies.
- The expected number of chance matches in S that agree with the query at the k fixed positions of one hashing pattern is simply $n/4^k$.

- To keep this number below 1 per pattern, we require

$$k \geq \log_4(n).$$

- For $n = 3 \times 10^9$, this implies $k \geq 16$ (much better!).

Now how about L?

- Our estimate for k assumes that each hashing pattern samples *exactly* k sequence positions; original LSH allowed patterns to sample fewer positions because they could sample with replacement.
- So, let's adjust our match probability to accommodate the assumption of sampling without replacement.
- Probability $P_m(\ell, R, k)$ that a match of length ℓ with at most R mismatches will match under a random hash pattern of length k is given by

$$P_m(\ell, R_k) = \frac{\binom{\ell-R}{k}}{\binom{\ell}{k}}.$$

- Hence, to achieve false negative rate $\leq \delta$, we need

$$(1 - P_m(\ell, R, k))^L \leq \delta.$$

- Equivalently,

$$L \geq \frac{\log(\delta)}{\log(1 - P_m(\ell, R, k))}.$$

- For our chosen parameters, this yields $L \geq 22$.

In conclusion, assuming a non-worst-case distribution for the background substantially decreases the anticipated cost of achieving high sensitivity. Note that we are still guaranteed to find each (ℓ, R)-mismatch to the query with probability at least $1 - \delta$, no matter how the R mismatches are distributed!

3 More Thoughts About Search Costs

- Our parameters (as well as original LSH) are based on an arbitrary decision to strictly limit the amount of time spent processing objects that are far from the query.
- In particular, we set k big enough to ensure at most about one chance match to a query in the whole database for each hash pattern.
- Is this rational? What if the cost of hashing (in particular, the cost of using L distinct hash patterns) far exceeds the cost of processing the (extremely rare) chance matches?
- To better balance these costs, assume we need $t_1 \cdot k$ seconds to hash a query with a single pattern and retrieve the corresponding bucket from one hash table.

- Moreover, assume we need t_2 seconds to verify by explicit comparison that a given ℓ-mer matches the query to within R mismatches.
- Then the overall cost of a search (not counting any non-chance matches, i.e. our real output) is given by
$$T(k) = L(t_1 k + t_2 n/4^k),$$
where L is computed from k as described previously.
- We should choose k so as to minimize this total cost, which might entail allowing a larger number of false positives to slip through in exchange for reducing L.

What other considerations might reduce the work needed?

- We could relax the guarantee of high sensitivity for *arbitrary* arrangements of mismatches within an ℓ-mer.
- In particular, specify some probabilistic model of how these mismatches are distributed, and use that when estimating L given k.
- In fact, we could pick a *deterministic* set of hash patterns that is optimized to maximize sensitivity against this model for fixed L and k.
- When $L = 1$, this is the problem of *optimal spaced seed design.* When $L > 1$, it's the problem of *optimal spaced seed set design.*
- See, e.g., PatternHunter (Li, Ma, and Tromp), Mandala (Sun and Buhler), work from Dan Brown, etc.
- This approach loses strong guarantees of sensitivity in the worst case but may yield excellent average-case and practical sensitivity at lower cost than the randomized approach.

Finally, consider how this problem relates to the generate-and-filter paradigm (e.g. BLAST) we studied earlier.

- The probability estimates above assume that we get exactly one chance per ℓ-mer per hash function to discover a match.
- But what if we are looking for a sequence feature longer than ℓ bases, with the same overall level of similarity to our query throughout?
- Such a matching feature might contain multiple (ℓ, R)-mismatches, and we need find only one of them to find the feature.
- (This is similar to BLAST getting multiple chances to find word matches in a longer alignment; we saw the impact for alignments with randomly distributed differences a few classes back.)
- It's not clear how to put worst-case sensitivity bounds on the chance of finding at least one (ℓ, R)-mismatch between a query and a feature, without imposing some kind of probabilistic model on how mismatches are distributed.

- Moreover, if we are willing to do a little ungapped extension (linear-time DP) around each hash pattern match, we can use it to check *multiple* nearby database locations for a match to the query.
- (Efficient implementation when using multiple patterns is left as an exercise, but consider what BLAST does with diagonal tracking.)
- These are potentially important improvements that dramatically reduce the necessary number of hash patterns L in practice, at the cost of easily obtained worst-case performance bounds.