6.438 Project Part II

Will Whitney

Model

The structure present in shotgun DNA reads provides an opportunity to use the redundant information present in overlapping reads to improve the compression ratio of the data. However, to take the fullest advantage of this redundancy, it is necessary to first determine where it occurs.

My solution takes advantage of the small size of the input data and its fixed structure by first reconstructing the true overlap of the shotgun sequences and encoding this information in `temp`. During decoding, I add an additional edge to the source graph between every overlapping pair of elements (which correspond to the same position in the true DNA sequence). These edges have a high "same" potential, encouraging multiple reads from the same DNA location to have the same value.

Preprocessing

The first step in my system is determining the overlapping indices of the shotgun sequences. Since the input data are so small, it is highly practical to do this by brute force, comparing every shotgun strand to every other at each possible overlap location. The information we wish to extract is, for each strand, which other strand most overlaps its tail, at what offset, and with what fidelity.

Since we know that each strand has at least one strand which comes after it in the sequence while still overlapping it, I check all forward offsets of a candidate match (`s2`) with the strand I am currently pairing (`s1`). Matches are scored by the percentage of `s1`'s and `s2`'s elements which are identical given the current

offset. In order to avoid chance trivial matches, I do not check offsets which would result in only a few overlapping elements.

Once I have a best 'tail overlap' calculated for each strand, I encode the `[match_percentage matching_strand offset]` triple for each strand into `temp`. In practice `match_percentage` is not really necessary, but it could be used to deal with high noise levels or other types of (inexact) matches by changing the potentials between the overlapped strands.

Code graph

No changes were made to the code graph either in implementation or in model.

Source graph

Armed with the information in `temp`, I construct a massive $m \times m \times 4 \times 4$ matrix corresponding to the potential between every pair of values for every pair of source nodes. In this matrix Ψ, all the diagonal-adjacent 4×4 blocks are set equal to `psi_source` or its transpose, reflecting the ordinary Markov structure. Then, for every pair of nodes (i, j) which my preprocessing has determined should be the same, I set $\Psi_{i,j}$ equal to a 4×4 matrix whose diagonal is `match_percentage` and whose rows and columns sum to 1.

This new graph corresponds to the original source Markov chain with additional edges added between the overlapping elements of each overlapping shotgun sequence. The use of `match_percentage` for the "same" potentials of these nodes allows for even a relatively poor (e.g. 85%) match to contribute information to the decoding.

I use ordinary loopy belief propagation on this new graph.

Tricks

Stability of added edges

Because `temp` must be composed of integers, the `match_percentage` describing the fidelity of the overlap between two strands may be rounded. In some cases, this rounding may result in a `match_percentage` value of 100, corresponding indicator functions as the potentials between each node in the overlap of those strands. When this happens incorrectly (e.g. by rounding), this can cause LBP to diverge, as it may attempt to enforce mutually exclusive constraints. To resolve this, I multiply `match_percentage` by a confidence factor (which is a function of the allowed number of digits of `match_percentage` which were passed, and is e.g. 0.99 when `match_percentage` lies in [0, 99]) when constructing Ψ; this relaxation results in convergence.

Determination of the doping rate

I experimented extensively with how much doping was necessary at various compression rates, noise levels, etc. Based on my results, I devised a function which gave `r_dope` as a function of the compression rate, using more doping at higher compression ratios, so as to minimize the doping rate while still having high confidence in convergence.

Additionally, I noted that the amount of overlap between the strands and the noise level of the reads greatly affected the amount of doping needed to ensure convergence. I summarized these effects by calculating the "average redundancy" which had been found in the alignment procedure and which was recoverable from `temp`. This value is zero if no strand has any overlap with any others, and 1 if every strand perfectly overlaps some other strand. I use this value to

increase the doping in cases where `temp` is less informative, allowing my system to continue to perform in adverse conditions without wasting bits when there is a great deal of redundancy in the data.

Message scheduling

I found that running two iterations of LBP on the source graph per loop improved convergence times.

Conclusion

The strategy I employed here is deeply flawed in that the encoder and decoder are both highly dependent on the artificial structure of this problem. Sequence alignment is a *hard problem*. It scales as $O(l^2 \cdot n^2)$ (where l is the length of each strand and n is the number of strands) and as such is not a reasonable component of a compression system where the amount of data may be large. Additionally, we are ignoring common real-world issues for this type of data, such as heterozygous samples, deletions, etc.

However, with those reservations, this method works well. It produces (normalized) compression ratios which can be quite small (as low as 0.46 in my experiments) under ideal conditions with lots of overlap between strands, yet scales smoothly to high-noise and low-overlap scenarios. Taking advantage of the structure of the problem yields predictably high-quality results.