Measuring Relatedness Between Strands of DNA

My genetic algorithm takes two DNA strands as inputs, and returns two dictionaries, one with the list of each strand selected in each intermediate generation, and one of the peptide (intermediate or not) that is coded by the strands. The heart of program is the function *evolution*, where the origin DNA is bred into a list (progeny) by the function *replicate*, and then evaluated by the functions *hamming_distance* and *pep_dif*. As *evolution* iterates through the progeny an evaluation criteria is applied to each element in the progeny list. Each element is assigned a score based on this evaluation. One point is added for each mismatch between the strand being analyzed and the descendent DNA (hamming distance) and one point is added for every amino acid (found by *get_peptide*) that does not match with the previous progeny's parent strand (peptide difference).

Although the function of my algorithm is to simulate evolution, it differs from nature in a number of ways. One way is the uniformity of the selection against intermediate peptide structures. In reality, some peptide intermediates are far more detrimental than others and should be hard selected against, whereas some peptides may not affect the overall fitness of the organism. My algorithm works with selection at a highly reduced level when in reality, selection takes place on a holistic, whole organism scale. I account for this in a primitive manner by placing heavier selection against the formation of premature stop codons (the most disruptive type of mutation). With more research into the biological interactions of different amino acids additional depth could be incorporated in the selection parameters.

Another highly practical way that my genetic algorithm deviates from nature is the mutation rate. Because the strands of DNA are finite I decided to use a brute force, exhaustive mutation process rather than having random mutations. Initially, I performed only every possible substitution mutation in each progeny. This program ran smoothly, but did not fully capture the natural potential as many phenotypically beneficial changes likely arise by frameshift mutations. To incorporate frameshift mutations, I extended the progeny list to account for every possible single point insertion or deletion in addition to substitution. I was able to do this successfully, but due to the exponential size increase of the progenies I am no longer able to efficiently run the program with lengths of DNA over fifty bps without freezing my terminal. This is a significant disappointment, as ideally I would use this program to compare orthologous genes between distant species. Given that the average gene is ten thousand base pairs long, it is unlikely that my program can be used to do this.

In the first edition of this algorithm I encountered a tricky error. Often the program would get stuck in an infinite loop, selecting the same best fit strand, or series of strands (the local maximum(s)), instead of progressing along a route towards the descendent DNA. This was because the selection against forming a new intermediate was stronger than the selection against having a large hamming distance with the descendent. My initial approach, changing the intermediate peptide cost to 0.25, worked, but was not ideal as it left the aversion to intermediates very low. Instead, I ended up creating a simplistic adaptive genetic algorithm. An adaptive genetic algorithm is another artificial enhancement for a genetic algorithm. It works by changing the parameters of of the program (generally the mutation rate, but also the evaluation criteria), as the computation takes place. My algorithm begins the process with a high cost of

forming intermediates, but if the current strand becomes equal to a strand in the dictionary that has already been selected (i.e, it is looping), then the cost of forming an intermediate is temporarily removed. The algorithm can then progress solely by hamming distance towards the descendent DNA. This is an example of an adaptive genetic algorithm, as the parameters of selection change based on the solutions being returned. It represents a way of getting around the local maximum problem by forcing the solutions to explore normally unfavorable regions. This is easier in my genetic algorithm than in others as I start out knowing the final form (the global maximum is just the descendent DNA). In most genetic algorithms a major challenge is determining whether a solution is a local maximum or the true final form.

My genetic algorithm only provides a sample path of how evolution between two strands may have occurred, and should only be used as a metric for their *likeliness* of relatedness. It does not make any empirical claims about the path. Primarily, this is because evolution is not a rational force and mutations occur randomly rather than exhaustively. I chose to mutate strands exhaustively so that different trial runs would provide consistent results. This means the result represents nature at its 'most lucky', when in reality, any two strands are likely separated by far more insignificant intermediates. It is also important to understand that my code does not provide the best possible solution when moving between strands. If an intelligent person were doing this by hand they could move between most pairs of strings without any intermediates! They could do this by artificially stacking the codons so that a single frameshift mutations would abruptly bring about the final phenotype. This does not represent nature however, as nature acts on each generation without any capabilities of foresight. Although my algorithm does have foresight in that it has a descendent strand it is moving towards, encoding a 'strategy' in it would

be counterproductive to its purpose. When I first incorporated selection against intermediates I became frustrated that sometimes, with the selection turned on, my paths ended up increasing the intermediate count. I also noticed that my paths tended to be longer with selection turned on. This was because in a snapshot of a single generation, when an intermediate had to be formed, the algorithm was choosing the intermediate with the least amino acid differences from the parent. With no selection, less intermediates were being formed, but those intermediates were wildly different from their parents. My algorithm does not actually minimize the number of intermediates, it minimizes the phenotypic changes between a parent strand and its descendent.

My code provides a single most probable path in an entirely deterministic fashion. Because of this, my code cannot act randomly when settling ties between two strands of equal fitness. The solution I employed was to choose the first element in the progeny list that shared the lowest score. Because of how I ordered the process, this weights mutations to C's over those to A's, which are more often selected than those to G, which in turn are chosen over mutations to T. Also, substitutions are chosen first over insertions, which are chosen first over deletions. (See the function *replicate*)

My code is useful because it provides a realistic metric for how related two strands of DNA are. Because it reflects the actual evolutionary force that diverged the strands rather than an arbitrary, human measure of distance, this algorithm can provide better insights into whether two strands are indeed closely related.