The design of the two main classes, representing an individual and a population of individuals in a genetic algorithm, are based on a(n ab)use of the availability of multiple inheritance in the Python class framework. A number of different choices can be made for how each class should behave, for example, how to select mutational positions and recombination cuts in an individual, objective and fitness functions, and how to select individuals for mutations and recombinations in the population. One approach would be to store information specifying these choices in object attributes.

Here we have taken an alternative approach, based on viewing the different choices as various disjoint aspects of the larger class that can be independently specified. So if we have a number of aspects $\{1, \ldots, n\}$ and for the i'th one has settled on choice $\text{class}^{i}_{j_i}$, we can construct the full class in much the same way Frankenstein constructed his monster, by defining the final class as `class(`$\text{class}^{1}_{j_1}$`, ..., `$\text{class}^{n}_{j_n}$`)`. The following text describes the various aspects and implemented choices for individuals and populations, as well as describing the basic contents of these classes.

1 Individuals

An individual basically consists of a current RNA sequence. For efficiency, some extra entities are stored when computed, to avoid recomputing structures, energies, Boltzmann distribution, etc. The methods available for the basic individual class are

getseq Returns the sequence of the individual

getstruc Takes an optional temperature argument and returns the RNAfold predicted structure at this temperature

getboltzmann Takes an optional temperature argument and a structure where unpaired probabilities (single position as index) and base pair probabilities (pair of positions as index) under the Boltzmann distribution at the temperature can be looked up

getminimumenergy Takes an optional temperature and returns free energy of minimum free energy structure at this temperature

gettargetenergy Takes a target structure (an object of class Target) and returns energy of sequences when folded according to the target

getobjective Returns value of objective function for individual

getfitness Returns fitness of individual

getpopulation Returns population individual is member of

setpopulation Sets population individual is member of to the population specified as argument

mutate Takes a position as argument and creates a new individual by forcing a mutation at that position and possibly mutating at all other positions depending on this position through base pairings

crossover Takes a second individual and a cross over point and creates a new individual by creating the recombinant defined by the two individuals and the cross over point

initialise Draws a sequence compatible with an optional target structure (using the population target if no target is specified)

1.1 Objective

The objective aspect of an individual is concerned with the value of the objective function for an individual. The objective should capture how close we are to the specified target, so is a measure of how close we are to having solved the problem. This aspect should provide methods

objective Returns value of objective function for individual

perfectsolution Returns whether individual is a perfect solution

Currently three choices are available under this aspect. `Objective` uses the number of positions where the minimum free energy structure differs from a target structure, averaged over all target structures. If a position is base paired, but to different positions in the MFE structure and a target structure, this is considered a position that differs. A solution is perfect if the MFE structures are identical to all target structures. `BoltzmannObjective` uses the expected number of positions that differs, again averaged over all target structures, where the expectation is under the Boltzmann distribution. A solution is perfect if the Boltzmann probability of all base pairs and unpaired positions is at least p for all targets, with $p = 1$ as default. `RelativeBoltzmannObjective` scores each position based on the difference between the probability of the correct structure and the maximum probability of all alternatives, under the Boltzmann distribution. If p_i denotes the probability of the position having the same annotation (unpaired or base paired to the same position) as in the target structure, and q_i is the maximum probability of any alternative (maximum probability of any base pair including i if i is unpaired in the target, or maximum of probability of i being unpaired and maximum probability of all other base pairs if i is base paired in the target), then the score contribution from position i is $\frac{1}{2} + \frac{3}{4}(q_i - p_i) - \frac{1}{4}(q_i - p_i)^3$. A solution is perfect if $p_i - q_i \geq \delta$ for all positions, with $\delta = 0$ as default corresponding to the target structure annotation being the most probable in all positions.

1.2 Fitness

The fitness aspect of an individual is concerned with the fitness of an individual used in selecting the next generation of the population. Usually the fitness will be strongly dependent on the objective, however we may want to introduce extra elements to the fitness like base composition or similarity to other individuals. This aspect only provides the method

fitness Returns the fitness for individual

Currently four choices are available. `Fitness`, `BoltzmannFitness`, and `RelativeBoltzmannFitness` coincides with the choices for objectives described in Sec. 1.1, but normalised by target length. `CombinedFitness` combines the other three fitnesses in a weighted manner, taking the sum of `Fitness` multiplied by w_1/w, `BoltzmannFitness` multiplied by w_2/w, and `RelativeBoltzmannFitness` multiplied by w_3/w, where $w = w_1 + w_2 + w_3$ and individual weights defaulting to 1.

1.3 Choose Position

This aspect of an individual is concerned with the choice of position selected for mutation. It provides methods

getposition Returns a position suitable for mutation

pweight Returns the score for a particular position, or for all positions if no position is specified. The score is interpreted as a measure of how wrong the individual is at each position, on a scale from 0 to 1

pweights Iterator over the scores, ordered by position

Currently five choices are available. `RandomPosition` defines a uniform distribution on all positions, oblivious to sequence content. `WrongPosition` defines the score as the fraction of target structures where the minimum free energy structure differs, and chooses a position randomly weighted by score. `BoltzmannPosition` defines the score as the expected difference between target structure and a structure sampled from the Boltzmann distribution, averaged over all target structures, and chooses a position randomly weighted by score. `RelativeBoltzmannPosition` defines the score of a position as described for `RelativeBoltzmannObjective` in section 1.1.

`CombinedPosition` combines the `WrongPosition`, `BoltzmannPosition`, and `RelativeBoltzmannPosition` scores in a weighted manner. For the positional weights returned by **pweight** and **pweights**, this is done in a similar manner to the description in section 1.1 by adding w_i/w times the positional weight of scheme i. For **getposition** the scores are further normalised by the sum of scores for that scheme, so that the weight w_i essentially defines the probability of using scheme i to select the position. I.e. the effect of the sum of scores of scheme i on the importance of scheme i in position selection is eliminated.

1.4 Mutate Dependent Position

This aspect deals with mutations induced by dependencies caused by the base pairs in target structures. With more than one structure, a mutation may cause a larger number of dependent positions to also have to be considered for mutation. This aspect determines how a new nucleotide is drawn, and has a single method

__mutatedependent__ Chooses a new base for position i, given a set of fixed nucleotides in positions this position is base paired to

There are currently two choices for this aspect. `MutateDependent` just ignores how the choices available score on correctness and draws from the distribution given for paired positions. If one of the (at most two) choices available is the current nucleotide at position i, `WeightedMutateDependent` takes the correctness at this position into account, a high correctness will result in a high probability of retaining this choice, while a low correctness makes it more likely to change the nucleotide.

1.5 Score Cut

This aspect of an individual is concerned with assigning a score to each possible cut that can be used for recombination. It provides methods

cweight Returns the score of a specific cut, or sum of scores of all cuts if no cut is specified

cweights Iterator over pairs of cuts and associated weight

Currently five choices are available. All are based on summing scores for each position in the cut, where the score is interpreted as a measure of how correct the individual is at each position. `RandomCut` assigns a score of one half to all positions, resulting in a uniform random choice of cuts. `CorrectCut` defines the score of a position to be the fraction of target structures for which the minimum free energy structure matches. `BoltzmannCut` defines the score of a position as the expected match between target structure and a structure sampled from the Boltzmann distribution, averaged over all target structures. `RelativeBoltzmannCut` defines the score of a position as described for `RelativeBoltzmannObjective` in section 1.1. `CombinedCut` combines the `CorrectCut`, `BoltzmannCut`, and `RelativeBoltzmannCut` scores in a weighted manner, as described for `CombinedFitness` in section 1.2.

2 Population

A population consists of a set of individuals. Additionally it has an associated target. Furthermore, all individuals are initially added as new members of the population, and need to be actively made standard members. This is to facilitate sequential creation of new individuals in the evolution step of the genetic algorithm, without using proposals as input to the evolution. The methods available for the basic population class are

gettarget Returns target for population

getbasedistribution Returns the distribution that is used for sampling nucleotides when initialising a sequence and mutating a position. The distribution has subdistributions for independent positions, i.e. positions not forming base pairs in any of the target structures, for base pairs, and for sampling nucleotides dependent on one or more positions with a fixed nucleotide (in this case we would usually have a choice either between C and U, or between A and G).

addrandom Adds n new individuals, randomly initialised, to the population

addnew Promote new members to standard members of the population

2.1 Mutate

The mutate aspect of a population defines how current members of the population are selected for creating new individuals by mutations. It provides a single method,

mutate Add n new members, constructed by mutations of standard members, to the population. If no n is specified, the current population size is used

Currently three choices are available. `Mutate` uses each standard member an equal number of times, choosing a random subset of these for one extra use if population size does not divide n. `RandomMutate` n times chooses a standard member uniformly at random with replacement. `FitnessWeightedMutate` n times chooses a standard member at random with replacement, with each member chosen with probability proportional to one minus its fitness.

2.2 Recombine

The recombine aspect of a population is concerned with choosing pairs of sequences for recombination, and specific crossover points for each pair. It provides a class, `Recombine`, with a single method,

recombine Add n new members, constructed as recombinants pairs of standard members. If no n is specified, the current population size is used

The aspect covers two subsidiary aspects, namely the choice of pairs of standard members and the choice of crossover point for a specific pair.

2.2.1 Select Pair

This aspect defines how pairs of individuals are chosen as parents in recombination events. It provides a single method,

__selectpair__ Selects n pairs of individuals, with n defaulting to current population size if not specified

Currently three choices are available. `SelectPair` chooses pairs uniformly at random with replacement. `WeightedSelectPair` also chooses pairs at random with replacement, but with the probability of choosing a particular individual proportional to one minus its fitness. `CombinedSelectPair` is the only choice where the two individuals are not chosen independently. Instead, for each crossover point the score of the `cweights`, cf. Sec. 1.5, for the positions the cut would specify for each individual are summed and the probability of choosing a pair is proportional to the sum over all cuts. As the `cweights` are intended to measure how correct an individual is at a position, this choice tends to favour pairs of sequences that complement each other, i.e. where we can choose crossover points such that each individual contributes to the recombinant from regions where it is mostly correct.

2.2.2 Select Cut

This aspect defines how a crossover point is selected for a particular pair of individuals. It provides a single method,

__selectcut__ Given two individuals, it selects a cut for creating a recombinant from the two individuals

Currently there are two choices available for this aspect. `SelectCut` chooses a cut such that each position that can be in a cut has a uniform probability of being selected. To elaborate, compatible cut points can be identified as disjoint sets from which any pair of points form a compatible pair. So rather than choosing a pair uniformly at random, which would tend to favour positions in large sets, a set is first chosen weighted by size and then a pair of positions from that set is chosen uniformly at random. `WeightedSelectCut` further weights each pair of positions with its `cweight`, cf. Sec. 1.5, while still normalising by the multiplicity of compatible pairs a position is part of.

2.3 Reduce

The reduce aspect of a population defines how a population is reduced to a smaller size, i.e. the selection step of the genetic algorithm. It provides a single method,

reduce Reduces the current population to n individuals

There are currently two choices available. `Reduce` chooses the n individuals with best fitness, with a uniformly at random choice between equally fit individuals. `DiversityReduce` iteratively chooses the n members, at each stage selecting an individual having a minimum weighted sum of fitness and average sequence similarity to the individuals already selected.

3 Targets

The implementation comes fully functional for multi-target inverse folding, i.e. finding a sequence that folds into multiple different structures, possibly at different temperatures. This is all captured in the `Target` class that represents a target by a list of target structures, where each structure in addition to the actual secondary structure also has an associated temperature. As mentioned in Sec. 1, if the associated temperature is `None` the default temperature of RNAfold is used.