examples

July 2, 2021

1 Genomes as weighted sums of permutations in `cgt`

First, import the package and everything from the enums module. `cgt` includes enums for every set of options so that your editor can provide you with the options available

[15]:
```
import cgt
from cgt.enums import *
```

We can define a `PositionParadigmFramework` under which we will model genomes. We can choose a number of regions to model, whether or not we want to capture orientation of the DNA in those regions, and the symmetry of the genome (for example, linear or circular). For now we will choose to model oriented, circular genomes with 4 regions. Printing the framework shows us a unique string representation.

[16]:
```
framework = cgt.PositionParadigmFramework(4, oriented=True, symmetry=SYMMETRY.
 ↪circular)
print(framework)
```

```
Framework for circular genomes with 4 oriented regions
```

Right away, we can take a look at a genome. By default, genomes are weighted sums of all the permutations we can use to represent them. The larger the symmetry group, the more terms in the sum. We can think of the coefficients as the proability of obtaining the corresponding instance if we arbitrarily fixed the genome positions (ignoring the symmetry).

[17]:
```
genome = framework.random_genome()
print(genome)
```

```
1/8*(2,4,-3)(3,-2,-4) + 1/8*(1,2)(3,-3)(4,-4)(-2,-1) + 1/8*(1,3,-4)(4,-1,-3) +
1/8*(1,4,-2,-3)(2,3,-1,-4) + 1/8*(1,-4,-2)(2,-1,4) + 1/8*(1,-3,-2,4)(2,-4,-1,3)
+ 1/8*(1,-2,3)(2,-3,-1) + 1/8*(1,-1)(2,-2)(3,4)(-4,-3)
```

We can view the genome in other ways, too. Select a random permutation representing the above genome. These are referred to as genome instances

[18]:
```
instance = framework.random_instance(genome)
print(instance
```

```
(2,4,-3)(3,-2,-4)
```

Each genome has a canonical instance. Here canonical means the instance which maps region 1 to position 1. The instance is returned in one_row notation.

[19]:
```
canonical_instance = framework.canonical_instance(instance)
print(canonical_instance)
```

```
[1, 4, -2, -3]
```

...but can be convered back to cycle notation easily.

[20]:
```
canonical_instance = framework.cycles(canonical_instance)
print(canonical_instance)
```

```
(2,4,-3)(3,-2,-4)
```

We can obtain the genome from a given instace, canonical or otherwise.

[21]:
```
new_genome = framework.genome(instance, format=FORMAT.formal_sum)
print(genome == new_genome)
```

```
True
```

Below, we can see that the canonical instance from above also represents a linear genome. There are two terms because there are two ways to represent a linear genome, since we can flip the whole genome over.

[22]:
```
print(cgt.PositionParadigmFramework(4, symmetry=SYMMETRY.linear).
 ↪genome(canonical_instance, format=FORMAT.formal_sum))
```

```
1/2*(2,4,-3)(3,-2,-4) + 1/2*(1,-4,-2)(2,-1,4)
```

In general though, it is best to not move between frameworks like this.

2 Defining a model

To model genome rearrangements, we need to define a set of possible rearrangements, and the probability each rearrangement has of occuring. This is simple to do in `cgt` for a range of different models. Below we define a model allowing inversions of single regions or pairs of adjacent regions, and specify that it is twice as likely to see a single region inversion. We can print the model's `generating_dictionary` to see the permutations that generate the model under conjugation by the symmety group.

[23]:
```
model = cgt.Model.named_model_with_relative_probs(framework, {
    cgt.MODEL.one_region_inversions: 2/3,
    cgt.MODEL.two_region_inversions: 1/3
})
print(model.generating_dictionary)
```

```
{(1,-1): 2/3, (1,-2)(2,-1): 1/3}
```

We can also obtain the `s` element as defined in [how do I do references in a notebook??]

[24]:
```
s = model.s_element(in_algebra=ALGEBRA.genome)
print(s, '\n ...or, in the group algebra:\n', model.s_element(in_algebra=ALGEBRA.
 ↪group))
```

```
1/3*(1,-2)(2,-1) + 2/3*(1,-1)
 ...or, in the group algebra:
 1/6*(4,-4) + 1/12*(3,-4)(4,-3) + 1/6*(3,-3) + 1/12*(2,-3)(3,-2) + 1/6*(2,-2) +
1/12*(1,-4)(4,-1) + 1/12*(1,-2)(2,-1) + 1/6*(1,-1)
```

To obtain the 'model element' in the genome algebra, we can multiply `s` by the symmetry element. Note that the order of the multiplication is reveresed, since `SageMath` multiplies permutations from left to right.

[25]:
```
z = framework.symmetry_element()
zs = s*z
print(zs)
```

```
1/24*(3,-4)(4,-3) + 1/12*(2,-4)(3,-3)(4,-2) + 1/24*(1,2,3,-1,-2,-3)(4,-4) +
1/12*(1,2,-1,-2)(3,-4)(4,-3) + 1/24*(1,3,-2,-4)(2,4,-1,-3) +
1/12*(1,3,-1,-3)(2,-2)(4,-4) + 1/24*(1,4,-2,-1,-4,2)(3,-3) +
1/12*(1,4,-1,-4)(2,-3)(3,-2) + 1/12*(1,-4,-3,-2,-1,4,3,2) +
1/24*(1,-4,-2,3)(2,-3,-1,4) + 1/12*(1,-3,-1,3)(2,4)(-4,-2) +
1/24*(1,-3,-4,-1,3,4)(2,-2) + 1/12*(1,-2,-3,-4,-1,2,3,4) + 1/24*(1,-2)(2,-1) +
1/12*(1,-1) + 1/24*(1,-1)(2,-4,-3,-2,4,3)
```

3 Applying rearrangements