Small Summery

Boris Prochnau

1. Juni 2014

1 Pseudocode

One thing to mention is that the Pseudocode snippets are marked as „Algorithm“, but in fact they are just functions and I dont know yet how to change the name in this particular LaTeX environment. Also I use a „=“ instead of „←“ because I think it provides better readability.
I separated this section in 3 parts, each one is dedicated to one of the functions used in EvolutionStep() below. It is possible to read
We will encounter 2 classes called „Trait“ and „Events“.

Attributes in a Trait Object:
class Trait

- Members
- BirthRate
- DeathRate
- TotalBirthRate
- TotalDeathRate
- TotalTraitRate
- TotalEventRate - [static]
- CompDeathRate[i][j] - [static]
- Mutation - [static]

Attributes in a Events Object:
class Events

- EventTimes[i]
- ChosenTrait[i]
- isBirth[i]

Algorithm 1 EvolutionStep()

Require: -
Ensure: A full evolution Step happened
1: calculateEventRates();
2: sampleEventTime();
3: changeATrait();

This function does a full evolution step.

1.1 Calculating total-event rates

Algorithm 2 calculateEventRates()

Require: -
Ensure: All (total)Rates will be set
1: **for** i=0 **to** n-1 **do**
2: calculateTotalDeathRateOf(i)
3: **end for**
4: calculateTotalBirthRates(0);
5: calculateTotalEventRate();

Algorithm 3 calculateTotalDeathRateOf(TraitIndex: i)

Require: int i
Ensure: Total deathrate of Trait „i“ will be set
1: Trait[i].TotalDeathRate = 0;
2: addTotalIntrinsicDeathRateOf(i);
3: addTotalCompetitionDeathRateOf(i);

Algorithm 4 addTotalIntrinsicDeathRateOf(TraitIndex: i)

1: Trait[i].TotalDeathRate = (Trait[i].DeathRate) · (Trait[i].Members)

Algorithm 5 addTotalCompetitionDeathRateOf(TraitIndex: i)

1: double externalDeath = 0;
2: **for** j=0 **to** n-1 **do**
3: externalDeath += (Trait.CompDeathRate[i,j])·(Trait[j].Members);
4: **end for**
5: externalDeath *= Trait[i].Members;
6: Trait[i].TotalDeathRate += externalDeath;

Algorithm 6 calculateTotalBirthRates(StartIndex: i)

Require: int i
Ensure: Total birthrate of Trait „i“ will be set (recursively)
1: Trait[i].TotalBirthRate = (Trait[i].Members)·(Trait[i].BirthRate)
2: **if** $i < n - 1$ **then**
3: calculateTotalBirthRates(i+1)
4: Trait[i].TotalBirthRate += $\frac{Trait.Mutation}{2} \cdot Trait[i+1].BirthRate \cdot Members$
5: **end if**
6: **if** $i > 0$ **then**
7: Trait[i].TotalBirthRate += $\frac{Trait.Mutation}{2} \cdot Trait[i-1].BirthRate \cdot Members$
8: **end if**

In Algorithm 6 in line 3 is used recursion, because this improves the calculation speed a lot, although it slightly makes code less intuitive.

Algorithm 7 calculateTotalEventRate()

Require: -
Ensure: Current Totaleventrate is set
1: Trait.TotalEventRate = 0;
2: **for** i=0 **to** n-1 **do**
3: Trait[i].TotalTraitRate = Trait[i].TotalBirthRate
+ Trait[i].TotalDeathRate;
4: Trait.TotalEventRate += Trait[i].TotalTraitRate;
5: **end for**

1.2 Sampling the next event-time

Here will appear a, not yet mentioned, object that will not be explained further, called Dice. The Dice Object will provide a uniform or exponential random Variable.

Algorithm 8 sampleEventTime()

Require: -
Ensure: First ringing Eventclock has been sampled
1: double Parameter = Trait.TotalEventRate;
2: double newEvent = this.Dice.RollExpDice(Parameter);
3: Events.EventTimes.push(newEvent);

Here we use Dice.RollExpDice(λ) to get $X \sim exp(\lambda)$. The same is possible for Dice.RollUnifDice(λ) to get $X \sim Unif[0, \lambda]$.

1.3 Changing a trait

Algorithm 9 changeATrait()

Require: -
Ensure: make a change to the Population with current Parameters

```
1: choseTraitToChange();
2: choseEventType();
3: executeEventTypeOnTrait();
```

Algorithm 10 choseTraitToChange()

Require: -
Ensure: Trait is chosen for changing

```
1: double Parameter = Trait.TotalEventRate;
2: double HittenTrait = Dice.rollUnif(Parameter);
3: for i = 0 to n-1 do
4:    if HittenTrait ≤ Trait[i].TotalTraitRate then
5:       Events.ChosenTrait.push(i);
6:       break;
7:    end if
8:    HittenTrait -= Trait[i].TotalTraitRate;
9: end for
```

Algorithm 11 choseEventType()

Require: -
Ensure: Decision for Birth or Death is made

```
1: int i = Events.ChosenTrait.lastentry();
2: double EventType = Dice.rollUnif(Trait[i].TotalTraitRate);
3: if EventType ≤ Trait[i].TotalBirthRate then
4:    Events.isBirth.push(true);
5: else
6:    Events.isBirth.push(false);
7: end if
```

Algorithm 12 executeEventTypeOnTrait()

Require: 2 other alg (later)
Ensure: Chosen event will occur on chosen trait

```
1: if isBirth.lastentry then
2:    Trait[Event.ChosenTrait.lastentry()].Members += 1;
3: else
4:    Trait[Event.ChosenTrait.lastentry()].Members -= 1;
5: end if
```

List of Algorithms