University of Camerino

School of Science and Technology

MSc in Computer Science (LM-18)

Performance Analysis and Simulation Project

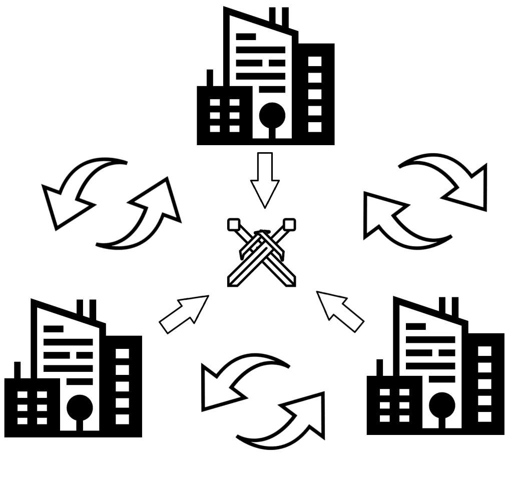
Student
Alessandro Antinori

Student ID
115924

A.Y. 2021/2022

Index

Introduction



This report is about the modeling and the simulation of a particular type of Population Model, called Matrix Population Models, in which matrix algebra is used in order to describe the evolution and the computations these populations "obey" to.

In particular all these population models closely follow an equation, called BIDE, that can be used to model them. More details about this equation and its meaning will be given in the relative chapter of this report.

What is worth noting is that the equation that describes these models is mathematically simple in and of itself, but the implication are quite interesting and varied. Moreover the estimates of the terms that appear in the equation are often difficult to obtain in the real world.

For this reason it is crucially important to use a supportive framework, capable of simulating these systems and collecting essential data to formalize and evaluate hypotheses and predict future outcomes. Different scenarios have also been implemented and their results have been analyzed in the corresponding chapter.

This has been made possible by *Sibilla*[2], a Java framework designed to support analysis of Collective Adaptive Systems. Furthermore, the resulting data (.csv files) have been processed and relevant graphs have been plotted by different Python scripts, collected in a Jupyter Notebook file.

To conclude, a very simple extension for Visual Studio Code has been developed, capable of syntax highlighting .pm and .sib files.

1. Population Model

Let's suppose we have N cities and an initial number of people that lives in these cities that may vary.

The people can perform different actions, among which are the followings:

- going to war against people from other cities;
- reproduce with someone else that belongs to the same city;
- emigrate to a different city in various ways.

There are different levels of abstraction that we can choose from to describe the system, but we'll focus on two main models:

- BD Model: in this model only births and deaths can occur, by respectively increasing and decreasing the number of individuals;
- BIDE Model: is the same as the BD model, but it also accounts for the possibility of individuals to move to another city, thus creating immigration/emigration phenomena.

1.1 BD Model

In the case of BD models, the population is treated as isolated, by taking only births and deaths into consideration when handling changes in size of the population.

However these BD models are incomplete, since even the most isolated populations are subjected to individuals joining and leaving it. That's the reason why it's difficult to find in nature systems that are close as in BD models, since most of the time they are open systems and have at least some form of communication with others.

BD models evolve according to the equation:

$$N_{t+1} = N_t + B_t - D_t$$

Figure 1.1: BD equation

That explains why BIDE models are preferred over BD models, because they give a more accurate abstraction of the system. In fact BIDE models take immigration (individuals joining a group) and emigration (individuals leaving a group) into equation.

1.2 BIDE Model

Matrix population models are mathematical models applied to the study of population dynamics in which matrix algebra is used. These models have a number of practical uses when dealing with population ecology to model the dynamics of wildlife or human populations.
Moreover, all these kind of models follow an equation, called a BIDE equation:

$$N_{t+1} = N_t + B_t + I_t - D_t - E_t$$

Figure 1.2: BIDE equation

The meaning of the equation 1.2 is as follows:
"The population size N at time t+1 is given by the population at time t plus the number of births (B) and immigrants (I), minus the number of deaths (D) and emigrants (E)."
We can make different assumptions regarding this population model:

- B, I, D, E are not rates, but they are the number of events at time t;
- the model is deterministic, not stochastic;
- time is discrete, not continuous.

In reality, things are more complicated, and interest lies in understanding the factors influencing each process[1].
More formally, a BIDE model can be considered as a tuple:

$$M = (X, D, d_0, T)$$

Figure 1.3: BIDE tuple

where:

- X = (P, B, I, D, E) is a vector of variables, where each variable is in turn a vector of size N (the number of cities);
- each variable keeps track of the number of people (e.g.: P represents the current number of people, B is the number of birth events, etc...):

  $$x_i \in R^N, \ \ \forall x_i \in X$$

  and so we have that the domain is:

  $$D = \prod_{x_i \in X} D(x_i)$$

- $d_0 \in D$ is the initial state of the system;
- T = set of transitions, each one identified by a label and each one specifying how the system evolves in time based on some pre-vector, post-vector and rate function.

2. Sibilla Model

The Sibilla framework, in its modelling package, provides support for different specification languages and one of them is a module for the specification of population models. As such, it was pretty straightforward to implement the BIDE model using this module.

2.1 Constant

A constant, named *warChance*, has been declared in the code:

```
const warChance = 1/4;
```

Code 2.1: Constant

Its function is to represent the probability that one of the four war rules expressed in the subsection 2.5.2 may happen.

Without this multiplicative constant factor, the war rules' rates are too high and the populations die rather quickly.

2.2 Parameters

The code also contains two parameters, whose values can be specified in the .sib file. These greatly influence the results of the simulation.

```
param N = 3;            /* cities */
param scale = 10;
```

Code 2.2: Parameters

The first parameter represents how many cities are there in the system. For the sake of simplicity N is considered to be a parameter, since it allows to quickly get different results by changing its value.

This is however true only when populations migrate in the next city. It is in fact not possible in the code to express the index of the city with the biggest/smallest population within a rule body (or at least I didn't manage to do so).

What would be desirable is the possibility to write something like this:

```
rule emigrate_to_biggest/smallest for i in [0, N] {
    [#P[i] is not the biggest/smallest] P[i] -[ #P[i]/#pop ]->
        E[i]|P[biggest/smallest]|I[biggest/smallest]
}
```

Code 2.3: Emigration Problem

Until then, the variable N can't be properly considered a parameter, but more of a constant.

In the project different rules have been nonetheless defined in order to support emigration to the smallest/biggest city, but they are not general and works only for a particular value of N (in this case 3). More details about this problem in subsection 2.5.3.

The second parameter instead is used in the system declaration part of the code as it is very useful to create balanced and unbalanced conditions.

2.3 Species

To model the system, there is a species for each term of the BIDE equation 1.2, plus another one to keep track of the number of people:

- species P: represents the current number of people in each city;
- species B: it keeps track of the number of births in each city;
- species I: it keeps track of the number of immigrants in each city;
- species D: it keeps track of the number of deaths in each city;
- species E: it keeps track of the number of emigrants in each city.

All these species have range [0, N], since they keep track of various things for each population.

In the Sibilla framework, the code is as follows:

```
species P of [0, N];    /* people */
species B of [0, N];    /* birth */
species I of [0, N];    /* immigration */
species D of [0, N];    /* death */
species E of [0, N];    /* emigration */
```

Code 2.4: Species declaration

2.4 Labels

To simplify some computations, some labels have been defined; they have been used to help in calculating rates and measures, while greatly reducing the code needed to do so.

These are their declarations in the code:

```
label pop = { P[i for i in [0, N]] }
label birth = { B[i for i in [0, N]] }
label immigration = { I[i for i in [0, N]] }
label death = { D[i for i in [0, N]] }
label emigration = { E[i for i in [0, N]] }
```

Code 2.5: Labels

When calculating the percentage of a population with respect to others of the same species, it isn't in fact possible to simply use the %P[i] operator, since it gives the fraction of agents of species P, but calculated on the total number of species (including B, I, D, E).

On the other hand, the ratio (#P[i] / #pop) gives us the expected result, since it is calculated with respect to the single species of interest, P.

2.5 Rules

Rules are used to describe the "laws" according to which the system evolves. Generally speaking, the syntax to specify a Sibilla rule is this one:

```
rule_declaration    :
    'rule' name=ID ((local_variables) (guard_expression)?)? '{
       '
        body=rule_body
    '}'
    ;

rule_body :
        ('[' guard=expr ']')?
        pre=species_pattern
        '-[' rate=expr ']->'
        post = species_pattern
    ;
```

Code 2.6: Syntax of a Sibilla rule

As we can see, the rule body is composed of different parts:

- a guard: it defines when a rule can be executed. This feature has been used a lot as it also allows to write complex conditions. It is optional;
- a pre-vector: the species that will be removed from the system;
- a rate: the probability that a certain rule gets executed;
- a post-vector: the species that will be added to the system.

Moreover, the "local_variables" part allows for the definition of loops, thus iterating over a range and this has been heavily used throughout the project, both in rules and in other parts of the code, such as labels and system configurations.

The more complex syntax (with respect to other syntactic structures) is the reason why the most difficult part of the project was certainly the definition of the rules.

To make the system evolve according to the BIDE equation, different rules and different versions for each rule have been written. Each of them belongs to one of these three types:

- Rules for reproduction;
- Rules that control war mechanics;
- Rules for emigration.

2.5.1 Reproduction - The B in BIDE

The easiest of all the rules, it tells us that two people can give birth to another one.

```
rule reproduce for i in [0, N] {
    P[i]<2> -[ #P[i]/#pop ]-> P[i]<3>|B[i]
}
```

Code 2.7: Reproduction and birth of a child

An alternative version of this rule can be one where half of the time two people give birth to a child and the other half to two children. That can be expressed as follows:

```
rule reproduce_1 for i in [0, N] {
    P[i]<2> -[ 0.5*#P[i]/#pop ]-> P[i]<3>|B[i]
}

rule reproduce_2 for i in [0, N] {
    P[i]<2> -[ 0.5*#P[i]/#pop ]-> P[i]<4>|B[i]
}
```

Code 2.8: Reproduction and birth of either one or two children

2.5.2 Making war - The D in BIDE

The general rule that models war between cities is the following one:

```
rule make_war for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#
       P[j] / #pop) ]-> result
}
```

Code 2.9: General rule for making war

In particular, the result of the war between two people P[i] and P[j] from different cities can be modeled as the result of one of four possibilities:

- P[i] survives and P[j] dies;
- P[i] dies and P[j] survives;
- both P[i] and P[j] die;
- both P[i] and P[j] survive.

This is the part of code that outlines these rules:

```
rule make_war_1 for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#
       P[j] / #pop) ]-> P[i]|D[j]
}

rule make_war_2 for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#
       P[j] / #pop) ]-> D[i]|P[j]
}

rule make_war_3 for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#
       P[j] / #pop) ]-> D[i]|D[j]
}

rule make_war_4 for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#
       P[j] / #pop) ]-> P[i]|P[j]
}
```

Code 2.10: Rules for making war

The utility of the guard in this case is to avoid conflicts between people from the same cities, since both variables i and j cycle from 0 to N.

The rate of these rules depends instead on the fraction of agents in a certain population with respect to the total population and, as explained in section 2.1, is multiplied by 1/4.

2.5.3 Emigration - The I/E in BIDE

Finally we get to the rules that govern the emigration/immigration phenomena. As done for the rules in section 2.5.2, here the general one is:

```
rule emigrate_to_other_city for i in [0, N] {
    [guard on P[i]] P[i] -[ #P[i]/#pop ]-> E[i]|
       P[index_of_other_city]|I[index_of_other_city]
}
```

Code 2.11: Emigrate to the next city

The main rule that has been written is the one that make people emigrate to the city immediately after (based on the index). It's the most simple one and it is general enough that it can work for every value of N.

```
rule emigrate_to_next for i in [0, N] {
    P[i] -[ #P[i]/#pop ]-> E[i]|P[(i+1)%N]|I[(i+1)%N]
}
```

Code 2.12: Emigrate to the next city

The index of the next city is calculated by doing the sum modulo N, so that whatever number comes after the current one will always be in the mathematical range [0, N).

But this is only one of the possible way to handle emigration to other cities. There are in fact other two rules that have been developed: move to the most populated city or to the least populated one.

As said in section 2.2, these rules are not general and need to be adapted to different values of N. For the sake of simplicity, they have been written with three cities in mind, in order to confront them with the rule in code 2.12.

For the emigration to the biggest city, we have the rule below:

```
rule emigrate_to_biggest_1 for i in [0, N] {
    [(#P[i] < #P[(i+1)%N]) && (#P[i] < #P[(i+2)%N]) && (#P[(i
       +1)%N] < #P[(i+2)%N])] P[i] -[ #P[i]/#pop ]-> E[i]|P[(i
       +2)%N]|I[(i+2)%N]
}

rule emigrate_to_biggest_2 for i in [0, N] {
    [(#P[i] < #P[(i+1)%N]) && (#P[i] < #P[(i+2)%N]) && (#P[(i
       +1)%N] > #P[(i+2)%N])] P[i] -[ #P[i]/#pop ]-> E[i]|P[(i
       +1)%N]|I[(i+1)%N]
}
```

Code 2.13: Emigrate to the biggest city

In a similar fashion we can also derive the rule for emigrating to the smallest city:

```
rule emigrate_to_smallest_1 for i in [0, N] {
    [(#P[i] > #P[(i+1)%N]) && (#P[i] > #P[(i+2)%N]) && (#P[(i
       +1)%N] > #P[(i+2)%N])] P[i] -[ #P[i]/#pop ]-> E[i]|P[(i
       +2)%N]|I[(i+2)%N]
}

rule emigrate_to_smallest_2 for i in [0, N] {
    [(#P[i] > #P[(i+1)%N]) && (#P[i] > #P[(i+2)%N]) && (#P[(i
       +1)%N] < #P[(i+2)%N])] P[i] -[ #P[i]/#pop ]-> E[i]|P[(i
       +1)%N]|I[(i+1)%N]
}
```

Code 2.14: Emigrate to the smallest city

It's worth noting that during the simulation, only one group of rules (among emigrate_to_next, emigrate_to_biggest and emigrate_to_smallest) is enabled, while the other ones are commented out; this is done in order to avoid possible conflicts and getting less meaningful results.

2.6 Measures

Starting from the labels described in section 2.4, we can create some measures that will be saved as separate .csv files during the simulation, in order to facilitate the subsequent analysis of the results:

```
measure P_tot = #pop;
measure B_tot = #birth;
measure I_tot = #immigration;
measure D_tot = #death;
measure E_tot = #emigration;

measure SINGLE_BIDE for i in [0, N] = #B[i] + #I[i] - #D[i] - #E[i];
measure BIDE = #birth + #immigration - #death - #emigration;
```

Code 2.15: Measures

Instead of writing more Python code to calculate the results, we let the Sibilla framework do all the computations for us (e.g.: population count, growth of single and overall population and so on).

2.7 Systems

As last thing, we need a way to specify the initial configurations of our population model and that's where the system keyword comes into play. With that, we can specify different initial states that can be chosen from later on in the Sibilla Script file.

```
system balanced = P[i for i in [0, N]]<scale>;
system unbalanced = P[i for i in [0, N-1]]<scale>|P[N-1]<10*scale>;
```

Code 2.16: System configurations

In the first case we create a system with N cities, each with the same population count; in the "unbalanced" state, we instead start with a city that has 10 times the amount of population of the other ones.

3. Scenarios

Different scenarios have been explored and each of them can be run from the Sibilla Shell by specifing the .sib file of interest. Once we have copied the project content inside the *shell/build/install/sshell/bin* folder, we can open a terminal there and launch one of the scenario as shown in the following command:

```
sshell
> run "child_vs_children/children.sib"
```

The resulting .csv files will be automatically saved in the appropriate subfolders.

When doing these analyses, we always assume unlimited resource and no carrying capacity, which is the maximum population size of a biological species that can be sustained by a specific environment[4].

Furthermore, the parameters used to run all the simulation are summed up in the table 3.1.

Parameter	Value
module	"population"
scale	10.0
deadline	10000
dt	1.0
replica	100

Table 3.1: Simulation parameter

3.1 BD model vs BIDE model

The first scenario that has been explored was the comparison between the project implemented as a BD model (birth-rate) and as a BIDE model (birth-immigration-death-emigration). The difference between the two implementation lies in the use of additional rules for immigration/emigration in the case of the BIDE model, as well as the use of species to keep track of the number of immigrants/emigrants.

As stated in the subsection 2.5.3, there are different strategies to implement emigration to another city and this will also be remarked in the section 3.3 of this chapter. By default, the strategy considered when using the BIDE model is the one defined in the emigrate_to_next rule (code 2.12).

As such, the results we want to highlight regards mostly the population per city and the overall population. First of all we can extract the information on the total population for each model, as shown in figure 3.1.

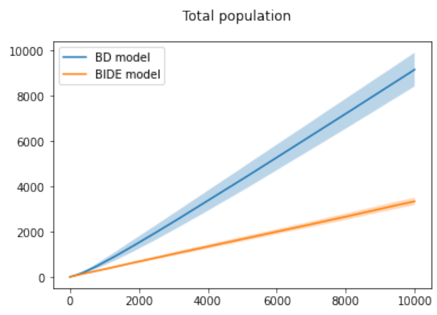


Figure 3.1: BD vs BIDE total population (and standard deviation)

As can be seen, when there are no immigration/emigration phenomena, the population grows faster but we will see that this is not always the best scenario for all the cities. Moreover, we can even plot the total number of people in each city for each model:

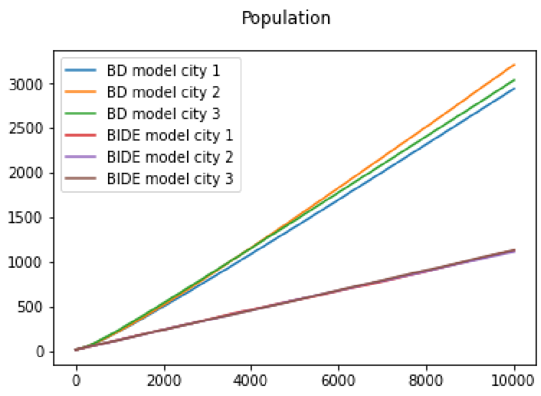


Figure 3.2: BD vs BIDE population per city

As expected, all the cities in both the BD and the BIDE model tend to grow, albeit at different growth rates, but the following graphs will show that having more population isn't necessarily the best scenario:

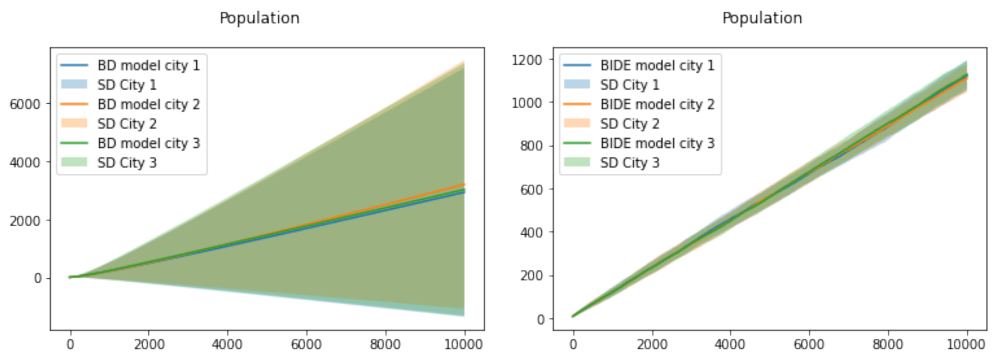


Figure 3.3: BD vs BIDE population per city ($\pm$ SD)

The graph on the right shows indeed that cities in the BIDE model grows more slowly, but different runs produce similar result (small values of standard deviation); on the other hand, the standard deviation in the BD model is so high that it also moves the plot below 0. What is happening here is that most of the time, only one of the three cities survives and thrives, while the other ones get completely destroyed during war.

Finally, we are able to plot the evolution of each species:

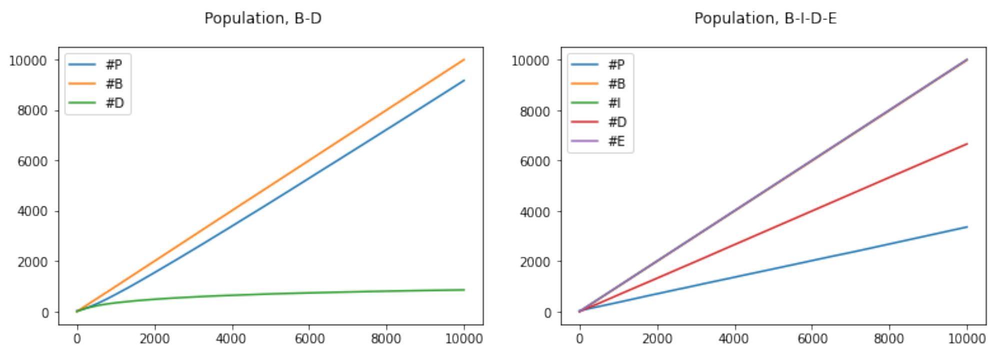


Figure 3.4: BD vs BIDE species

As the equations that govern these two models state, in the BD and BIDE model we respectively have that:

$$N_{t+1} = N_t + B_t - D_t$$
$$N_{t+1} = N_t + B_t + I_t - D_t - E_t$$

But since Sibilla gives us for each species the total number of events up until time t, we can reason about the two models more according to these other equations:

$$N_{t+1} = N_0 + B_t - D_t$$
$$N_{t+1} = N_0 + B_t + I_t - D_t - E_t$$

And from the graphs we can definitely derive these conclusions (in the BIDE model the lines for #I and #E coincide, as expected, and are also pretty similar to #B in value).

From this moment on, we will focus solely on the BIDE model.

3.2 Balanced vs Unbalanced

Now that we've established the BIDE model as our base model, we can study our system by changing some of its behaviours and observing the different outcomes.

First of all, we can analyse how the initial configuration of the system will impact the simulation. As explained in the snippet 2.16, we can have a balanced system (each city has the same number of people) or an unbalanced one (each city but one has the same number of people).

As done before, we can plot some of the same graphs and observe that the initial state of the system doesn't matter too much, since after a while the two systems grow in a similar way.

Here are the result obtained:

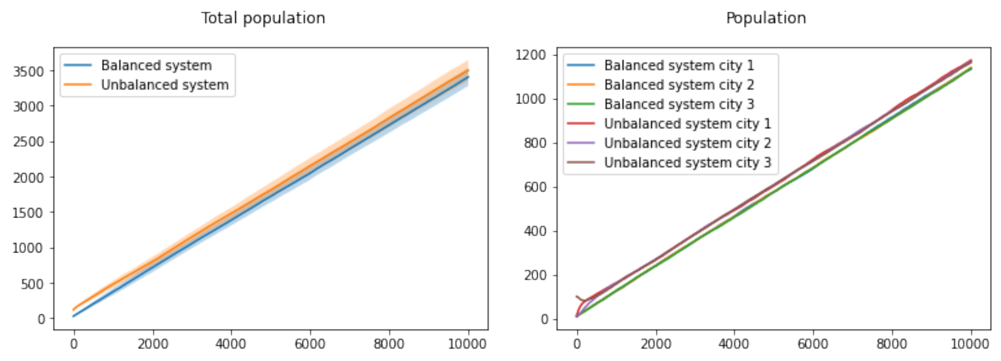


Figure 3.5: Balanced vs Unbalanced total and per city population

It is also worth noting that the difference on the y axis between the two lines related to the total population is due to the number of people present at time t=0:

- Balanced system: each city has 10 people and we have 3 cities, for a total of 30 people;
- Unbalanced system: each city has 10 people but one, which has 10*10=100 people, for a total of 120 people.

If we were to create a balanced system with 40 people for each city, we would probably have overlapping lines in both graphs.

Furthermore, if we zoom on the initial part of the graph on the right, we can also observe that in the unbalanced system, all the cities show a trend toward a situation in which the population is uniformly distributed across all of them. The situation described is represented by figure 3.6.

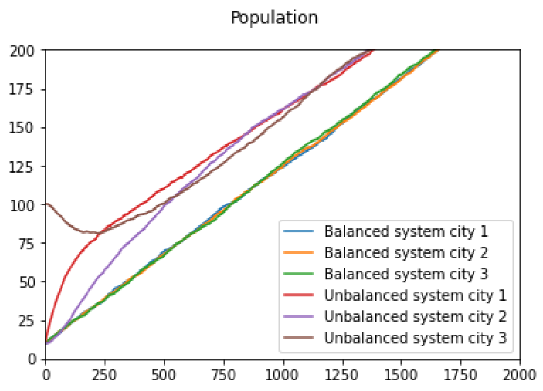


Figure 3.6: Zoom on Balanced vs Unbalanced per city population

3.3 Emigrate to Next vs Biggest vs Smallest city

Next up, we test various emigration strategies and try to assert their impact on the system.

First of all, we plot the total population for each strategy:

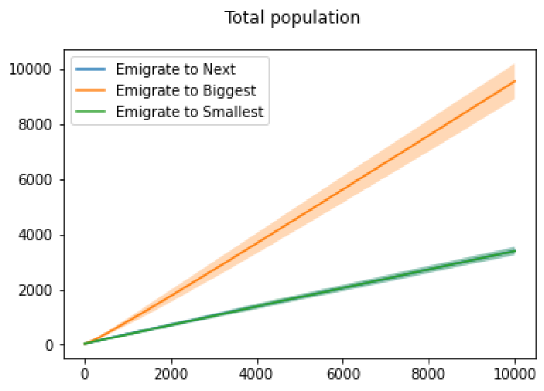


Figure 3.7: Total population for each emigration strategy

At first glance, we see that emigrating to the biggest city produce the greatest overall population but, as was the case before, we need to plot it in a more meaningful way to extract more information on why this happens. And, just like before, we can show the standard deviation to have a grasp of it:

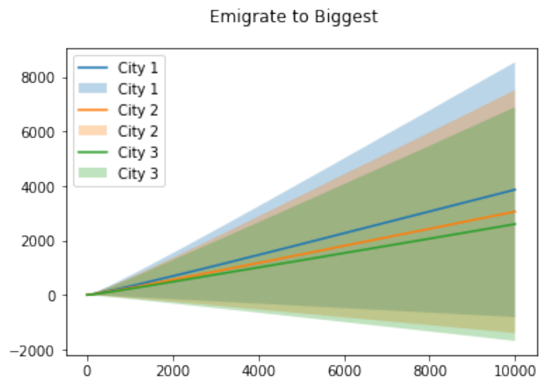


Figure 3.8: Emigrate to biggest city strategy

The high values for the standard deviation indicate once again that only one city remains and grows, in fact by making people move to the biggest city, we let the other cities become progressively depopulated.

Analysing the difference between the other two strategies is harder as it is more subtle. Nevertheless, we can plot the same graph of figure 3.8 for them:

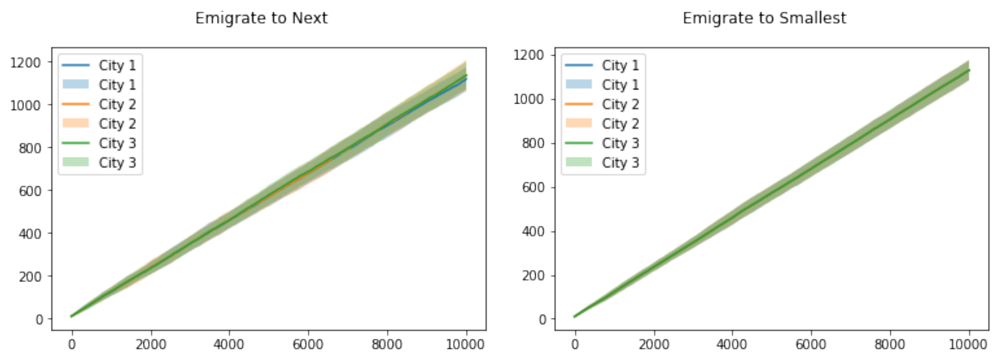


Figure 3.9: Emigration to next city vs Emigration to the smallest city

By resorting to the standard deviation graph, we are able to tell two main things:

- the SD values for the first strategy are slightly greater than for the second strategy and the lines often differ by small amounts; in the second case we can see that the lines for each city nearly always coincide. This also means that, from an initial situation, the emigrate to smallest strategy will take less time to reach a point of equilibrium;
- on the long run, the two strategies lead to the same amount of people, evenly distributed over the three cities. This is probably the most interesting and unexpected result.

3.4 Different number of children

And lastly, we modify the rules for reproduction and see what happens when we allow for more than one child to be born at a time.

Normally, the rule in code 2.7 (we'll call that Child) states that two people of the same city can generate one and only one child at a time. In the alternative version used here (called Children), described in code 2.8, 50% of the time one child is born and the other 50% two children are birthed.

As we expect, the alternative version leads to a higher number of overall population, as can be clearly seen here:

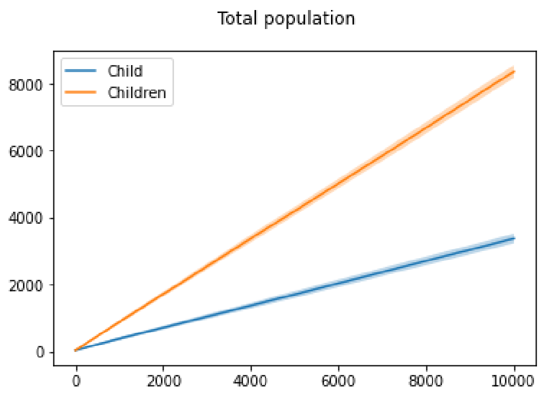


Figure 3.10: Child vs Children total population

The percentages used in the rules have also deeper implications:

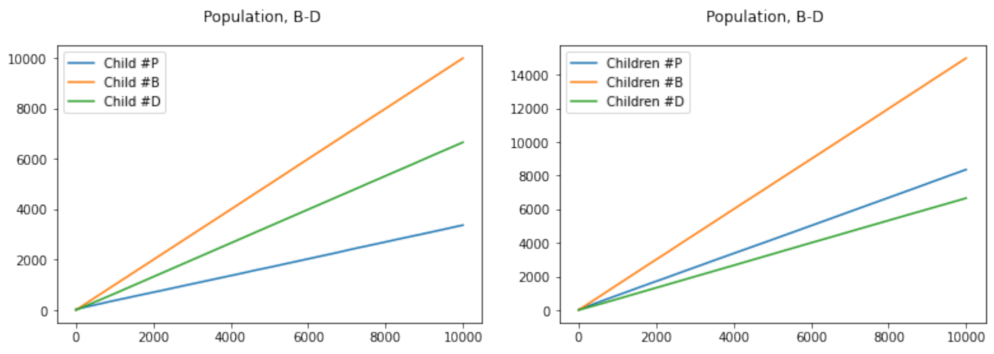


Figure 3.11: Child vs Children Births and Deaths

On average, the Children rule generates in fact 1.5 children for each reproduction, as opposed to an average of 1 child for each reproduction in the Child rule. For the number of births in the graph of the left (around 10000), we get indeed an higher number on the graph on the right (around 15000), which reflects these averages. For the same amount of deaths, this also affects the total population count.

4. Conclusion

As written in the introduction part of this report, the goal of the project was to study a particular kind of population models, called Matrix Population Models and the underlying equation (the BIDE model) that governs them.

Almost all the features available in the Sibilla Population Model has been used to provide a clean and compact code.

Moreover, I wanted to implement and analyse different scenarios in which I kept constant some behaviours and altered others. As such, different variants have been explored: BD model vs BIDE model, Balanced vs Unbalanced systems, Emigration strategies (next city, biggest city, smallest city) and different reproduction rule. Each scenario has been subsequently analysed and the corresponding result explained in the relative chapter.

The Sibilla framework was powerful enough to carry out all the simulations and also implement all the various scenarios by changing just few lines of code.

Finally I made use of a Jupyter Notebook containing all the Python code needed to extract the different graphs shown throughout the report.

A language support for Visual Studio Code for Sibilla Population Model and Sibilla Script has also been developed. It is pretty trivial and takes inspiration from the Java extension to also support various things, such as boolean, char and so on; as such, it can definitely be improved.

To end this report, I would like to share a quotation from a British statistician, which I think best sums up the importance of simulation, in all the fields in which it can and should be applied.

> All models are wrong, but some are useful[1].
>
> —George E. P. Box

[1]The aphorism recognizes that statistical or scientific models always fall short of the complexities of reality but can still be of use[3]

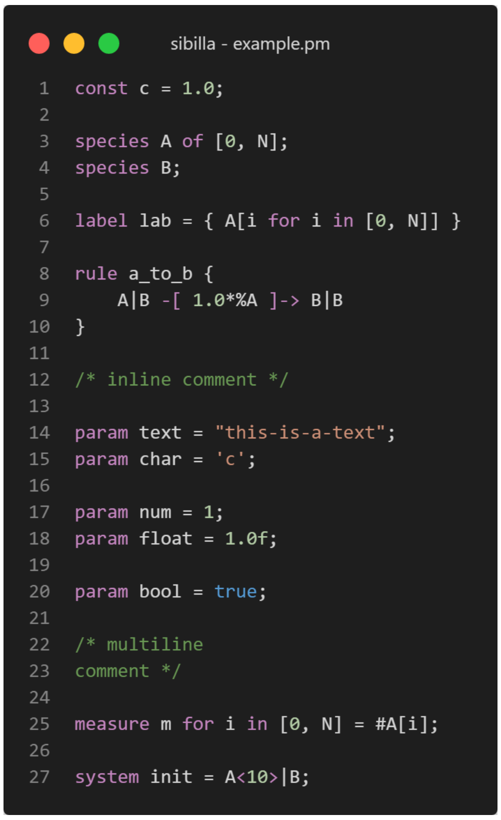
5. Visual Studio Code Extension

To ease the development of the project and also to easily create more appealing presentation slides, a Visual Studio Code extension has been developed for the syntax highlighting of Sibilla's .pm and .sib files.

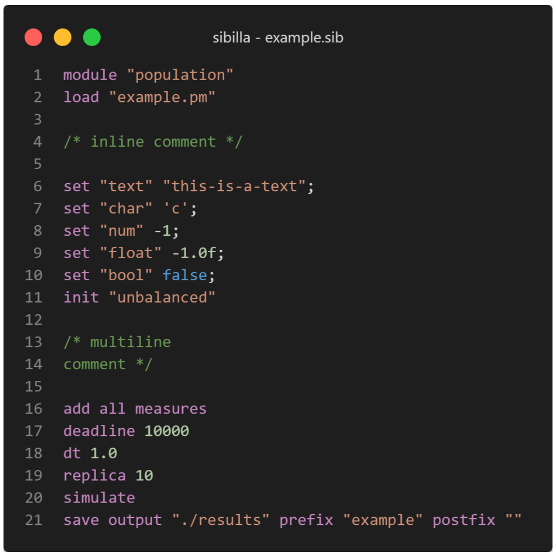
To add this extension to an instance of Visual Studio Code, all you need to do is:

- Move the sibilla folder containing the developed extension in the Visual Studio Code installation folder, inside the directory "resources/app/extensions"
- Restart Visual Studio Code

The implementation is pretty trivial and for this reason it may lack some features. However all the keywords defined in the SibillaScript.g4 and PopulationModel.g4 files are highlighted as such and there is also support for comments, strings, booleans and so on.



```
const c = 1.0;

species A of [0, N];
species B;

label lab = { A[i for i in [0, N]] }

rule a_to_b {
    A|B -[ 1.0*%A ]-> B|B
}

/* inline comment */

param text = "this-is-a-text";
param char = 'c';

param num = 1;
param float = 1.0f;

param bool = true;

/* multiline
comment */

measure m for i in [0, N] = #A[i];

system init = A<10>|B;
```



```
module "population"
load "example.pm"

/* inline comment */

set "text" "this-is-a-text";
set "char" 'c';
set "num" -1;
set "float" -1.0f;
set "bool" false;
init "unbalanced"

/* multiline
comment */

add all measures
deadline 10000
dt 1.0
replica 10
simulate
save output "./results" prefix "example" postfix ""
```

Figure 5.1: Example of Sibilla Population Model and Script.

A. Sibilla Population Model

```
const warChance = 1/4;

param N = 3;              /* cities */
param scale = 10;

species P of [0, N];     /* people */
species B of [0, N];     /* birth */
species I of [0, N];     /* immigration */
species D of [0, N];     /* death */
species E of [0, N];     /* emigration */

label pop = { P[i for i in [0, N]] }
label birth = { B[i for i in [0, N]] }
label immigration = { I[i for i in [0, N]] }
label death = { D[i for i in [0, N]] }
label emigration = { E[i for i in [0, N]] }

rule make_war_1 for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#P[j] / #pop) ]->
        P[i]|D[j]
}

rule make_war_2 for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#P[j] / #pop) ]->
        D[i]|P[j]
}

rule make_war_3 for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#P[j] / #pop) ]->
        D[i]|D[j]
}

rule make_war_4 for i in [0, N] and j in [0, N] {
    [!(i == j)] P[i]|P[j] -[ warChance * (#P[i] / #pop) * (#P[j] / #pop) ]->
        P[i]|P[j]
}

rule emigrate_to_next for i in [0, N] {
    P[i] -[ #P[i]/#pop ]-> E[i]|P[(i+1)%N]|I[(i+1)%N]
}

/*rule emigrate_to_biggest_1 for i in [0, N] {
    [(#P[i] < #P[(i+1)%N]) && (#P[i] < #P[(i+2)%N]) && (#P[(i+1)%N] < #P[(i
       +2)%N])] P[i] -[ #P[i]/#pop ]-> E[i]|P[(i+2)%N]|I[(i+2)%N]
}

rule emigrate_to_biggest_2 for i in [0, N] {
    [(#P[i] < #P[(i+1)%N]) && (#P[i] < #P[(i+2)%N]) && (#P[(i+1)%N] > #P[(i
        +2)%N])] P[i] -[ #P[i]/#pop ]-> E[i]|P[(i+1)%N]|I[(i+1)%N]
}*/

/*rule emigrate_to_smallest_1 for i in [0, N] {
    [(#P[i] > #P[(i+1)%N]) && (#P[i] > #P[(i+2)%N]) && (#P[(i+1)%N] > #P[(i
        +2)%N])] P[i] -[ #P[i]/#pop ]-> E[i]|P[(i+2)%N]|I[(i+2)%N]
}

rule emigrate_to_smallest_2 for i in [0, N] {
    [(#P[i] > #P[(i+1)%N]) && (#P[i] > #P[(i+2)%N]) && (#P[(i+1)%N] < #P[(i
        +2)%N])] P[i] -[ #P[i]/#pop ]-> E[i]|P[(i+1)%N]|I[(i+1)%N]
}*/

rule reproduce for i in [0, N] {
    P[i]<2> -[ #P[i]/#pop ]-> P[i]<3>|B[i]
}

/*rule reproduce_1 for i in [0, N] {
    P[i]<2> -[ 0.5*#P[i]/#pop ]-> P[i]<3>|B[i]
}

rule reproduce_2 for i in [0, N] {
    P[i]<2> -[ 0.5*#P[i]/#pop ]-> P[i]<4>|B[i]
}*/

measure P_tot = #pop;
measure B_tot = #birth;
measure I_tot = #immigration;
measure D_tot = #death;
measure E_tot = #emigration;

measure SINGLE_BIDE for i in [0, N] = #B[i] + #I[i] - #D[i] - #E[i];
measure BIDE = #birth + #immigration - #death - #emigration;

system balanced = P[i for i in [0, N]]<scale>;
system unbalanced = P[i for i in [0, N-1]]<scale>|P[N-1]<10*scale>;
```

Code A.1: Sibilla Population Model

B. Sibilla Script

```
module "population"
load "model.pm"
set "scale" 10.0
init "system"
add all measures
deadline 10000
dt 1.0
replica 100
simulate
save output "results_folder" prefix "prefix_" postfix ""
```

Code B.1: Sibilla Script

Bibliography

[1] Richard Chandler. *APD Lecture 2 - The BIDE Model*. https://rbchan.github.io/applied-popdy/lectures/BIDE/lecture-BIDE-handout.pdf. 2020.

[2] Michele Loreti. *Sibilla*. https://github.com/quasylab/sibilla.

[3] Wikipedia contributors. *All models are wrong — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=All_models_are_wrong&oldid=1058929559. 2021.

[4] Wikipedia contributors. *Carrying capacity — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Carrying_capacity&oldid=1070436439. 2022.