Simulating travel behaviours

Erik Cenci

October 23, 2014

Contents

1 Reasons for the project

One of the most interesting problems, economically speaking, is the analysis of the mobility network.
In recent years we faced the emerging of some economic novelties that disturbed the equilibria of the entire system like the low-cost air companies, and the high speed train connections, that especially in Europe are often preferred by private traveller.
Starting from this background the project tries to create a framework able to simulate people moving around the world with the ability to choose their system of transportation depending on their characteristics.

2 The model

Two different transportation systems, i.e. trains and air planes are considered in the model, these can be modelled as two network-layers, where the first layer represents the railway network in which the nodes are the stations and the edges are the possible connections among stations, whereas the second one represents the airports network where the nodes model the airport and the edges represent the flights.
People are asked to move around the world but constrained to remain on the nodes of a network, and they can "change" the layer (or the transportation systems) only when are on those nodes that belong to both the networks.
Each person, given a destination, has to choose the means of transportation depending on her features (her propensity to choose speedy travel against cheaper ones).
Thus we have two layers of network, upon the nodes people are placed, so that some people care more about costs of travel, others about time spent travelling, and the remaining about the comfort of trip related to the number of stopover, those are given destinations, so that depending on the preferences they have to choose, if possible, the way to reach the destinations, moreover if the destination appears not reachable for their location they don't move.
Moreover each train and flight update its price depending on how many people travel on them

3 The interface

As we can see in Figure 6 the interface of the project is composed by 2 buttons, 9 sliders, the world representation and 3 plots. About the two button those are called setup and go: the first is to be clicked to build the system and the second, that is a forever button makes the system to evolve.
The sliders are used to define quantitatively the quantities that characterize the system: like number of links that each rail station creates toward an other one, number of flights that each airport creates toward an other one; the total amount of walkers, the fractions of walkers that prefer to minimize the expenditure, and the time (the fraction of people that prefers to maximize comfort is computed by difference); the initial price of trains and flights and their capacity.
The plots represent emerging behaviours of the system.
The first one shows three lines: the fraction of walkers that choose train on the total of travelling walkers, the fraction that choose to fly on the travelling walkers, and the fraction of the total of walkers that is not able to get to the destination; in the second are displayed the averages of passengers of trains and planes; in the third are showed the mean and maximum prices of flights and trains.

4 The code

The first lines of the code are dedicated to the declarations of global variables, the breeds of links and turtles and their data member:

```
breed [railStations railStation]
breed [airports airport]
directed-link-breed [trains train]
trains-own [price time comfort passengers capacity]
directed-link-breed [flights flight]
flights-own [price time comfort passengers capacity]
breed [walkers walker]
walkers-own [location destination TotExpenditure...
 ...TotTimeTravel TotComfort Preference]
globals [DayTrainCount DayFlightCount DayNoTravelCount]
```

In particular we can see that are defined 3 breeds of turtles: railStations airports and walkers; 2 breed of link: trains and flights and for each of them their properties.
The first procedure defined is the SetUp:

```
to setup
  clear-all
```

```
set DayTrainCount 1
set DayFlightCount 1
set DayNoTravelCount 0
set-default-shape railStations "square"
set-default-shape airports "circle"
create-railStations 49
file-open "./DatiAeroporti_Stazioni/ProveCoord.txt"
while [not file-at-end?][
  ask railStation file-read [
    set color blue
    set xcor file-read
    set ycor file-read]]
file-close
let MoneySavers MoneySaversFraction * NumberWalkers
let TimeSavers TimeSaversFraction * NumberWalkers
create-walkers MoneySavers [
  set color red
  set location one-of railStations
  set TotExpenditure 0
  set TotTimeTravel 0
  set TotComfort 0
  set preference 1]
create-walkers TimeSavers [
  set color red
  set location one-of railStations
  set TotExpenditure 0
  set TotTimeTravel 0
  set TotComfort 0
  set preference 2]
create-walkers NumberWalkers - TimeSavers - MoneySavers [
  set color red
  set location one-of railStations
  set TotExpenditure 0
  set TotTimeTravel 0
  set TotComfort 0
  set preference 3]
ask railStations [repeat NumberTrainForStation [
    let node one-of other railStations
    create-train-to node
    create-train-from node]]
```

```
  ask trains [
    set price PriceTrain
    set time 250
    set comfort 100
    set passengers 0]
  file-open "./DatiAeroporti_Stazioni/Airports.txt"
  while [not file-at-end?][
    ask railStation file-read [
      set breed airports
      set color green]]
  file-close
  ask airports [repeat NumerFlightForAirports [
      let node one-of other airports
      create-flight-to node
      create-flight-from node]]
  ask flights [
    set price PriceFlight
    set time 25
    set comfort 70
    set passengers 0]
  ask trains [set capacity round TrainCapacity]
  ask flights [set capacity round FlightCapacity]
  ask walkers [move-to location]
  reset-ticks
end
```

The first lines of this procedure contain the clear all command, and the setting of the shapes of airports and railStations.
Then 49 railStation turtles are created, and their coordinates are setted, reading them from a file.
Later depending on the sliders fixed by user through the interface a number of walkers is created, they model the people that travel around the world, these are setted on a railStation.
The following section creates the connections among the railStations, to do that each station is asked to establish some links (always depending on the user definition) with one of other stations, this links are trains with their costs, time comfort and number of passengers.
Now that a network of stations is created, 21 of them, the ones enumerated in the file, are transformed in airports, and again to each airport is asked to establish links with other airports, these represent flights that share same properties of trains. Thus we have airports connected both through trains and flights.

The last steps are fix capacity of links, move walkers to their location and reset the ticks counter.
The second procedure implemented is the ChooseDestination one:

```
to ChooseDestination
  let loc [who] of location
  let index [who] of location
  while [loc = index] [
    set index random 49
    set destination turtle index]
end
```

This procedure simply selects randomly a destination for the walker that invokes it ensuring that it is different from the location where the walker stands.
Below that procedure there are three procedures, very similar to each other. Those procedures are invoked by walkers and test whether exist in the network a continue path to get from the location to the assigned destination. The first is called FindWayByTrain:

```
to-report FindWayByTrain
  let station -1
  let intermedio -1
  let index1 [who] of location
  let index2 [who] of destination
  let trains1 trains with [[who] of end1 = index1]
  let trains2 trains with [[who] of end2 = index2]
  ask trains1 [set intermedio [who] of end2
    ifelse intermedio = index2 [
      set station intermedio
      stop]
    [ask trains2 [if intermedio = [who] of end1[
        set station intermedio
        stop]
      stop]]]
  report  station
end
```

the second FindWayByFligh:

```
to-report FindWayByFlight
  let station -1
  let intermedio -1
  let index1 [who] of location
```

```
  let index2 [who] of destination
  let flights1 flights with [[who] of end1 = index1]
  let flights2 flights with [[who] of end2 = index2]
  ask flights1 [set intermedio [who] of end2
    ifelse intermedio = index2 [
      set station intermedio
      stop]
    [ask flights2 [if intermedio = [who] of end1[
          set station intermedio
          stop]
        stop]]]
  report station
end
```

the last is FindMixedWay:

```
to-report FindMixedWay
  let station -1
  let intermedio -1
  let index1 [who] of location
  let index2 [who] of destination
  let links1 links with [[who] of end1 = index1]
  let links2 links with [[who] of end2 = index2]
  ask links1 [set intermedio [who] of end2
    ifelse intermedio = index2 and...
     ...passengers < capacity - 1 [
      set station intermedio
      stop]
    [ask links2 [if intermedio = [who] of end1...
     ...and passengers < capacity - 1 [
          set station intermedio
          stop]
        stop]]]
  report station
end
```

We can notice that these are to-report procedures that implies that the result of each is reported in the enclosing procedure. All are built so that in the first step the location and the destination are defined as numbers and are declared some auxiliary variables, then two agentsets of links are created the first contains all out-links of location and the other all the in-link of destination.
Later the procedure firstly test if there is a direct link from location to destination,

if not looks for a couple of links (one contained in the first link subset and the other in the second) such that the node where the first link arrives is the same node from which the second starts, thus if there exists that couple, the procedure has found the needed stopover of the travel that allows to reach the destination. It is important to notice that the mixed path procedure can be used to find path that need a change of mean of transportation, but also to find at least one more possibility if the firsts ones are not available due to the constraint of the capacity. The last procedure is the one that makes the system evolve and models the choosing criteria of agents, this is the Go procedure:

```
to go
  set DayTrainCount 0
  set DayFlightCount 0
  set DayNoTravelCount 0
  ask links [set passengers 0]
  ask walkers [
    ChooseDestination
    let Tinter -1
    let Minter -1
    let Finter -1
    let Price1 9999999999999
    let Time1 9999999999999
    let Comf1 9999999999999
    let Price2 9999999999999
    let Time2 9999999999999
    let Comf2 9999999999999
    let Price3 9999999999999
    let Time3 9999999999999
    let Comf3 9999999999999
    let index1 [who] of location
    let index2 [who] of destination
    if is-airport? location and is-airport? destination [
      set Finter FindWayByFlight
      if Finter != -1
      [ifelse Finter = [who] of destination
        [ask flight index1 index2[set Price2 price
          set Time2 time
          set Comf2 comfort]]
        [ask flight index1 Finter[
          set Price2 price
          set Time2 time
```

```
          set Comf2 comfort]
        ask flight Finter index2[
          set Price2 Price2 + price
          set Time2 Time2 + time
          set Comf2 Comf2 + comfort]]]]
    set Tinter FindWayByTrain
    if Tinter != -1
    [ifelse Tinter = [who] of destination
      [ask train index1 index2[
          set Price1 price
          set Time1 time
          set Comf1 comfort]]
      [ask train index1 Tinter[
        set Price1 price
        set Time1 time
        set Comf1 comfort]
      ask train Tinter index2[
        set Price1 Price1 + price
        set Time1 Time1 + time
        set Comf1 (Comf1 + comfort)]]]
    set Minter FindMixedWay
;    let DivFlag 0
    if Minter != -1 and Minter != Tinter and Minter != Finter
    [if Minter != index2
      [ifelse [out-flight-neighbor? turtle Minter] of turtle index1
        [ask flight index1 Minter[
          set Price3 price
          set Time3 time
          set Comf3 comfort]]
;          set DivFlag DivFlag + 1]]
        [ask train index1 Minter [
          set Price3 price
          set Time3 time
          set Comf3 comfort]]
      ifelse [out-flight-neighbor? turtle index2] of turtle Minter
      [ask flight Minter index2 [
        set Price3 Price3 + price
        set Time3 Time3 + time
        set Comf3 Comf3 + comfort ]]
;       set DivFlag DivFlag + 1]]
```

```
      [ask train Minter index2 [
        set Price3 Price3 + price
        set Time3 Time3 + time
        set Comf3 Comf3 + comfort]]]
    ]
;      if DivFlag != 1 [
;        set Price3 9999999999999
;        set Time3 9999999999999
;        set Comf3 9999999999999]] Without comment
;does not check whether the travel uses different mean
;of transportation, but comment allow to consider
;different path to reach the destination
;passing trough diverse cities

    ifelse Tinter = -1 and Finter = -1 and Minter = -1
    [set DayNoTravelCount DayNoTravelCount + 1]
    [let PriceList []
      let TimeList []
      let ComfList []
      set PriceList lput Price1 PriceList
      set PriceList lput Price2 PriceList
      set PriceList lput Price3 PriceList
      set TimeList lput Time1 TimeList
      set TimeList lput Time2 TimeList
      set TimeList lput Time3 TimeList
      set ComfList lput Comf1 ComfList
      set ComfList lput Comf2 ComfList
      set ComfList lput Comf3 ComfList
      if preference = 1 [
        let pos position min PriceList PriceList
        ifelse pos = 0
        [set TotExpenditure TotExpenditure + Price1
          set TotTimeTravel TotTimeTravel + Time1
          set TotComfort TotComfort + Comf1
          set DayTrainCount DayTrainCount + 1
          ifelse Tinter = index2[ask train index1 index2
          [set passengers passengers + 1]]
            [ask train index1 Tinter [set passengers passengers + 1]
              ask train Tinter index2 [set passengers passengers + 1]]]
        [ifelse pos = 1
```

```
    [set TotExpenditure TotExpenditure + Price2
      set TotTimeTravel TotTimeTravel + Time2
      set TotComfort TotComfort + Comf2
      set DayFlightCount DayFlightCount + 1
      ifelse Finter = index2[ask flight index1 index2
      [set passengers passengers + 1]]
      [ask flight index1 Finter [set passengers passengers + 1]
        ask flight Finter index2 [set passengers passengers + 1]]]
    [set TotExpenditure TotExpenditure + Price3
    set TotTimeTravel TotTimeTravel + Time3
    set TotComfort TotComfort + Comf3
    set DayTrainCount DayTrainCount + 1
    set DayFlightCount DayFlightCount + 1
    ifelse [out-flight-neighbor? turtle Minter] of turtle index1
      [ask flight index1 Minter [set passengers passengers + 1]]
      [ask train index1 Minter [set passengers passengers + 1]]
    ifelse [out-flight-neighbor? turtle index2] of turtle Minter
      [ask flight Minter index2 [set passengers passengers + 1]]
      [ask train Minter index2 [set passengers passengers + 1]]
      ]]]
if preference = 2 [
  let pos position min TimeList TimeList
  ifelse pos = 0
  [
    set TotExpenditure TotExpenditure + Price1
    set TotTimeTravel TotTimeTravel + Time1
    set TotComfort TotComfort + Comf1
    set DayTrainCount DayTrainCount + 1
    ifelse Tinter = index2[ask train index1 index2
    [set passengers passengers + 1]]
      [ask train index1 Tinter [set passengers passengers + 1]
        ask train Tinter index2 [set passengers passengers + 1]]]
  [ifelse pos = 1
    [set TotExpenditure TotExpenditure + Price2
      set TotTimeTravel TotTimeTravel + Time2
      set TotComfort TotComfort + Comf2
      set DayFlightCount DayFlightCount + 1
      ifelse Finter = index2[ask flight index1 index2
      [set passengers passengers + 1]]
      [ask flight index1 Finter [set passengers passengers + 1]
```

```
        ask flight Finter index2 [set passengers passengers + 1]]]
    [set TotExpenditure TotExpenditure + Price3
    set TotTimeTravel TotTimeTravel + Time3
    set TotComfort TotComfort + Comf3
    set DayTrainCount DayTrainCount + 1
    set DayFlightCount DayFlightCount + 1
    ifelse [out-flight-neighbor? turtle Minter] of turtle index1
      [ask flight index1 Minter [set passengers passengers + 1]]
      [ask train index1 Minter [set passengers passengers + 1]]
    ifelse [out-flight-neighbor? turtle index2] of turtle Minter
      [ask flight Minter index2 [set passengers passengers + 1]]
      [ask train Minter index2 [set passengers passengers + 1]]
      ]]]
if preference = 3 [
  let pos position min ComfList ComfList
  ifelse pos = 0
  [set TotExpenditure TotExpenditure + Price1
    set TotTimeTravel TotTimeTravel + Time1
    set TotComfort TotComfort + Comf1
    set DayTrainCount DayTrainCount + 1
    ifelse Tinter = index2[ask train index1 index2
    [set passengers passengers + 1]]
      [ask train index1 Tinter [set passengers passengers + 1]
        ask train Tinter index2 [set passengers passengers + 1]]]
  [ifelse pos = 1
    [set TotExpenditure TotExpenditure + Price2
      set TotTimeTravel TotTimeTravel + Time2
      set TotComfort TotComfort + Comf2
      set DayFlightCount DayFlightCount + 1
      ifelse Finter = index2[ask flight index1 index2
      [set passengers passengers + 1]]
      [ask flight index1 Finter [set passengers passengers + 1]
        ask flight Finter index2 [set passengers passengers + 1]]]
    [set TotExpenditure TotExpenditure + Price3
    set TotTimeTravel TotTimeTravel + Time3
    set TotComfort TotComfort + Comf3
    set DayTrainCount DayTrainCount + 1
    set DayFlightCount DayFlightCount + 1
    ifelse [out-flight-neighbor? turtle Minter] of turtle index1
      [ask flight index1 Minter [set passengers passengers + 1]]
```

```
            [ask train index1 Minter [set passengers passengers + 1]]
          ifelse [out-flight-neighbor? turtle index2] of turtle Minter
            [ask flight Minter index2 [set passengers passengers + 1]]
            [ask train Minter index2 [set passengers passengers + 1]]
            ]]]
      move-to destination
      set location destination
    ]]
  ask links [set color 6
    set thickness 0.1]
  ask trains [
    if passengers > 0.7 * capacity
    [set price price + 0.2 * price ]
    if passengers < 0.2 * capacity
    [set price price - 0.3 * price]]
  ask flights [
    if passengers > 0.7 * capacity
    [set price price + 0.2 * price ]
    if passengers < 0.2 * capacity
    [set price price - 0.3 * price]]
  ask trains with-max [passengers]
  [if passengers > 0
  [set color 47
      set thickness 0.3]]
  ask flights with-max [passengers]
  [if passengers > 0
  [set color 47
      set thickness 0.3]]
  ask trains with-max [price]
  [set color 67
    set thickness 0.3]
  ask flights with-max [price]
  [set color 67
    set thickness 0.3]
  tick
end
```

This big procedure, fixed some initial conditions, ask to all the walkers to choose their destination and if there are multiple paths to get there to choose the best way according to their preference, updating the daily counters that model the observable variables and finally updating the price of trains and flights.

More in details for each agent are created the needed variables, then if location and destination are airports is invoked the procedure FindWayByFlight if it finds a path to get to the destination (being both a direct flight or more flights with stopover) it memorizes the price and other properties of the travel. Then is invoked the FindWayByTrain procedure searching for competing path through railway and if it exists the properties of the travel are also memorized. In the end is invoked the FindMixedWay to search for mixed path or available alternative, memorizing also data related to that trip.
Later through a nested and multiple ifelse is maked the walker to choose the best path for him depending if it wants to minimize the expenditure, or the time spent travelling or prefers to maximize the Comfort. During this step are updated the counters according to the choice of the traveller.
In the end are updated the price of all flights and trains proportionally to the number of passengers that travelled on that particular line and are coloured links with more passengers (yellow) and links more expensive (green).

5 Results

5.1 The robustness of the network

The first interesting aspect of the simulation is about the network generated. By construction the network upon which we make move the walkers is a connected network that is a network such that each node can be reached from any other node (maybe through many links), but in our simulation walkers can choose only between travel with at most one stopover. This implies that to walkers the network appear not connected.
This is simply verifiable setting a low number of trains and flights connecting respectively a rail station with other rail stations and a airport with other airports, and then trying to launch the simulation: in the first plot will be showed that a large portion of the walkers cannot reach their destination.
This property appears quite stable modifying other parameters of the simulation like number of walkers, their preference, and initial price of trains and flights. In any case the portion of walkers that can not reach the assigned destination is about the 75%.
It appears rather interesting that the percentage above is less sensible to augmenting the number of flights than to augmenting the number of trains, this is explainable by the fact that the number of airports is lower than the number of rail stations so a higher fraction of walkers is setted onto a rail station and also the probability that the destination is a rail station is higher. The results are showed in the Table 5.1.
Moreover it seems to be critical the value of 7 trains for each station to reduce to 0 the percentage of non-reachable destinations, this consideration allows the conclusion that a network of airport even strongly connected is rather irrelevant if its airports are not reachable, that is quite intuitive but relevant in the following section about mean prices.

5.2 The importance of capacity

All the results about connectedness are obtained without considering the finite amount of passengers that can be hosted on a train or plane. This is to explore the characteristics proper of the network, avoiding the introduction of parameters which are proper of the process that we develop on the network. It is now interesting to highlight how the parameter of the capacity of trains and flights modifies that results, indeed it is intuitive that the constrain of a limited amount of passengers on a link every day makes unreachable some destinations to some passengers since that train/flight is full. So it should not seem strange if modifying the capacities sliders will grow up the portion of unreachable destinations.

# Trains	# Flights	% Unreachable
1	1	78%
1	2	65%
1	3	60%
2	1	55%
2	2	45%
2	3	40%
3	1	33%
3	2	28%
3	3	20%

Table 1: 5.1: Table of percentage of unreachable destination depending on number of trains that link every rail station and on number of flight that connect every airport.This results are obtained fixing as unlimited the capacity of both trains and flights.

5.3 The role of the total number of walkers

The total number of walkers is a quantity that obviously influences the system, but it does in particular if it is considered with the capacity of means of transportation. Indeed the amount of walkers is irrelevant when the capacity of trains and flights is very large, in this case we always obtain that prices of both trains and flights drop down exponentially, the portion of travellers who choose to fly is about 33% and the portion of those who use the train is 66%.
More interesting are the cases when we fix the capacity at values such that represent real constrains to the flow of passengers.
For example for fixed initial condition (50% of people prefer to minimize expenditure and the 50% the time, train capacity = 16, flight capacity = 9), but varying the number of walkers we find something interesting for value higher than 3000, indeed for lower values are repeated the following output:

- prices drop to zero exponentially as much rapidly as smaller is number walkers.
- the fractions of people flying and train travelling are respectively 25% and 75%.
- the average amount of passengers for each mean of transportation grows if number of walkers grow but the one related to trains is always higher, and both are lower than any capacity.

For values higher that 3000 we find something different:

- prices no more drop to zero, but the flight higher price grows up exponentially.
- the behaviours remain the same, so we have a partition like 75% of people choose train, and 25% flight.
- the average of passengers of the same order of the capacity for flights whereas for trains is still much lower than the capacity.

To complete the analysis and to highlight the evolution of a overfilled system we tested one more critical situation leaving 5000 the number of walkers, but reducing the capacity of trains and flights bringing them to 6 for trains and 7 for flights obtaining the following results:

- average prices of both trains and flights explode this time reaching the same order, whereas until now average trains' prices were much lower than flights'.
- average passengers for both means of transportation are close to the capacity.
- the partition of behaviours remains the same, but we have that about 15-20% of travellers can not more reach the destination.

5.4 Different scenarios

Starting from previous results it appears reasonable to fix all the parameters but preference of walkers as follows:

- NumberTrainForStation = 14, and NumberFlightForAirport = 8.
- TrainPrice = 101, and FlightPrice = 376.
- TrainCapacity = 6, and FlightCapacity = 7.
- NumberWalkers = 2051.

This setting appear good since as in real world from a train station is possible to reach more destinations then from an airport, price of train travel is on average lower than the price of a flight, and in the end the capacity are setted to be comparable to the average on links of the passengers that results for that particular value of NumberWalker, and reflecting the fact that the capacity of trains is often higher than the capacity of a airplane.
All the following results are obtained by simulations that have all the same initial conditions.

5.4.1 All people want to minimize expenditure

If all walkers as set to prefer to minimize the expenditure we get that after a transient almost the 70% of travellers do it by train, and only 30% decide to fly. The average passengers on each trains after the same transient stabilize and display values like 4 passengers for trains and 3 passengers for plane.
In the end the evolution of average prices is interesting to understand the transient, in fact we see that in the first days the flight prices collapse because no one chooses to fly, and become lower than the train prices, then both means of transportation are chosen so the prices start to grow up. In the Figure 5.4.1 are reported the graphics.

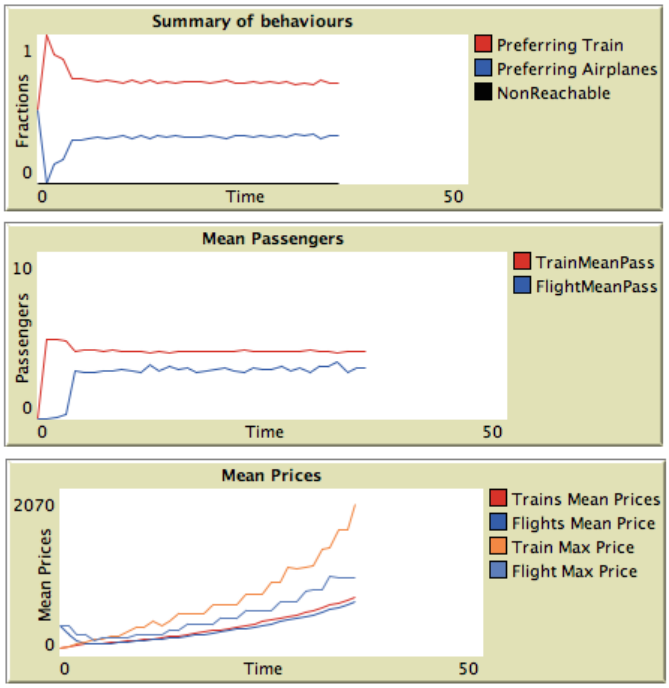


Figure 1: Results obtained if all people prefer to minimize expenditure.

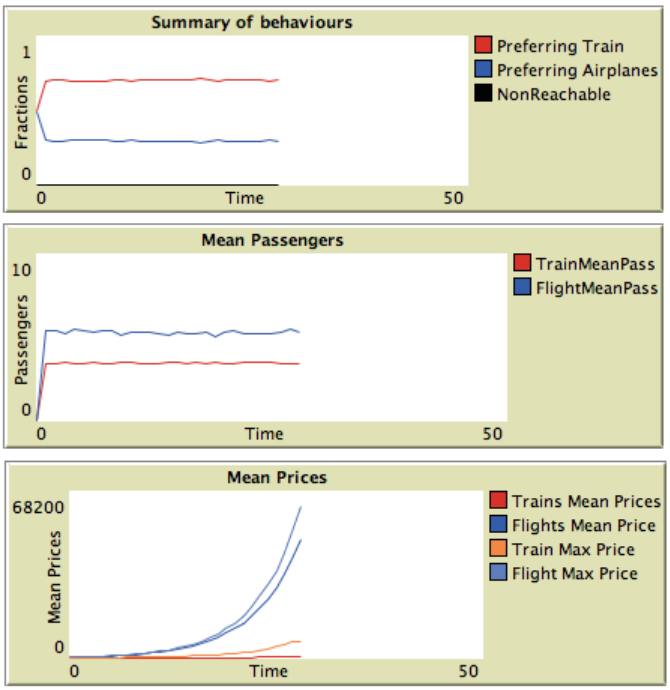


Figure 2: Results obtained if all people prefer to minimize time spent travelling.

5.4.2 All people want to minimize time

When all people want to minimize the time spent travelling we have that flights average passengers are close to the capacity of the plane, so the prices of flights grows up immediately, we can also highlight as the mean price of flight is close to the maximum price of flight. About the train prices (maximum and average) we can say that they also grows, but so much slower, this can be explained both by the facto that so much people prefer plane to train, and by the fact that initial price of train is lower than the one related to plane, thus the increment (of 20%) for plane is higher than the increment for train. The last thing to say is that this scenario is the first one in which we have an average number of passengers which fly that is higher than the average passengers on train, all these observations can be compared to the plots in Figure 5.4.2.

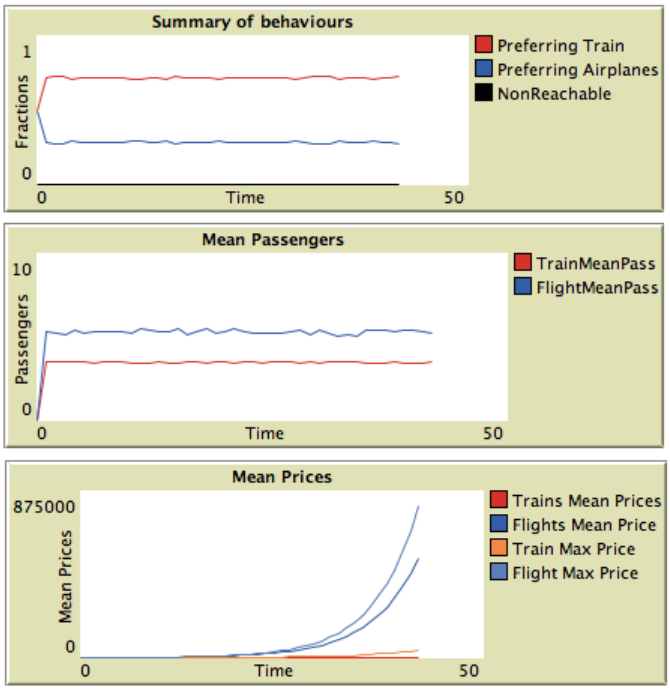


Figure 3: Results obtained if all people prefer to maximize comfort that it means that prefer to minimize stopover.

5.4.3 All people want to maximize comfort

This scenario is probably the less interesting because our model is not perfectly soft tuned, even if we estimated the loss of comfort due to stopover such that between two flights and only one train trip is to be preferred the train trip, the dynamic appear almost similar to the dynamic of previous scenario, giving the same output. See Figure 5.4.3 for benchmark.

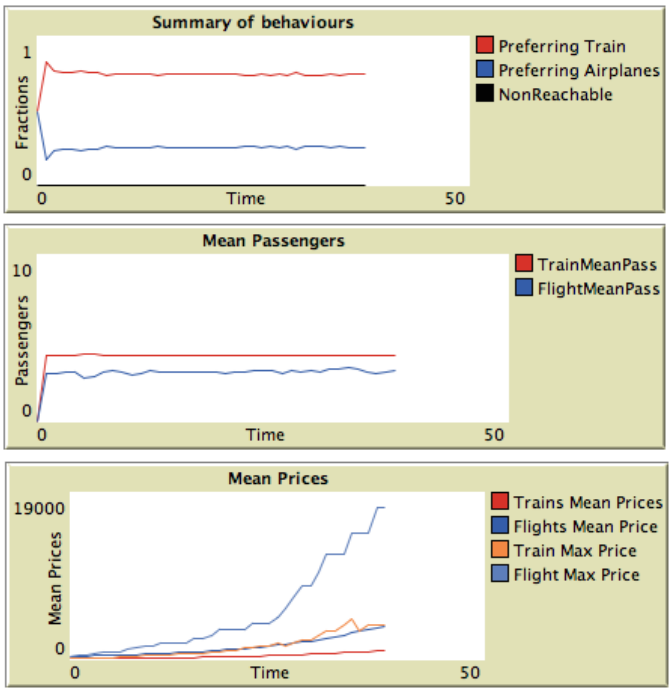


Figure 4: Results obtained if half people prefer to minimize expenditure and half prefer to minimize time.

5.4.4 Half people want to minimize expenditure and half time

The first of two intermediate scenarios that we studied is the one in which the population is divided in 2 categories, half people prefer to minimize the expenditure, and the left prefer to minimize time spent. In this case we can see that the behaviours plots show two lines more separated with respect to the first scenario (it means that a lower fraction of travelling people prefer to fly), and again the average of passengers of trains is higher than the average of passengers on plane. Moreover the prices curves are more similar, in Figure 5.4.4 we can notice how curves related to both means of transportation grows, the curves of maximum prices grows quite rapidly, whereas the curves of mean prices show little differences that are caused by the fact that the average passengers on plane is constantly over the threshold of 70% of capacity that makes higher the price, whereas the average passengers on plane is close to that threshold, but non always over. Plots are reported in Figure 5.4.4.

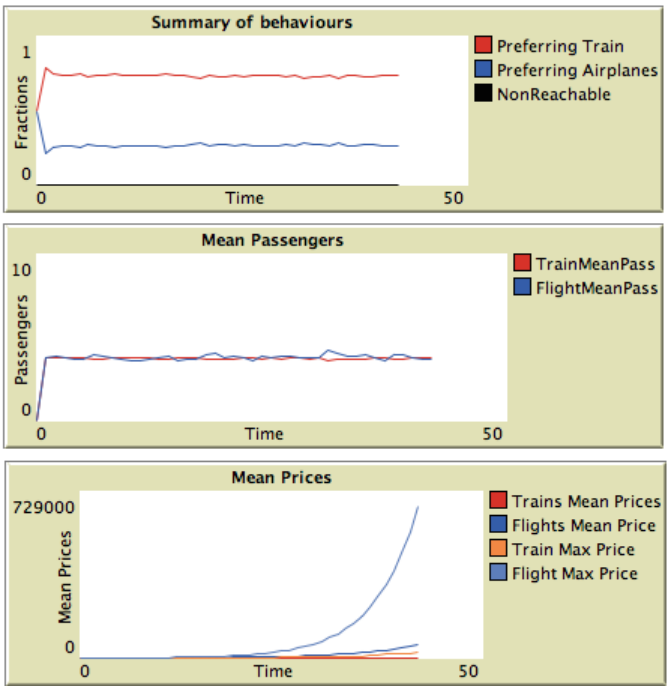


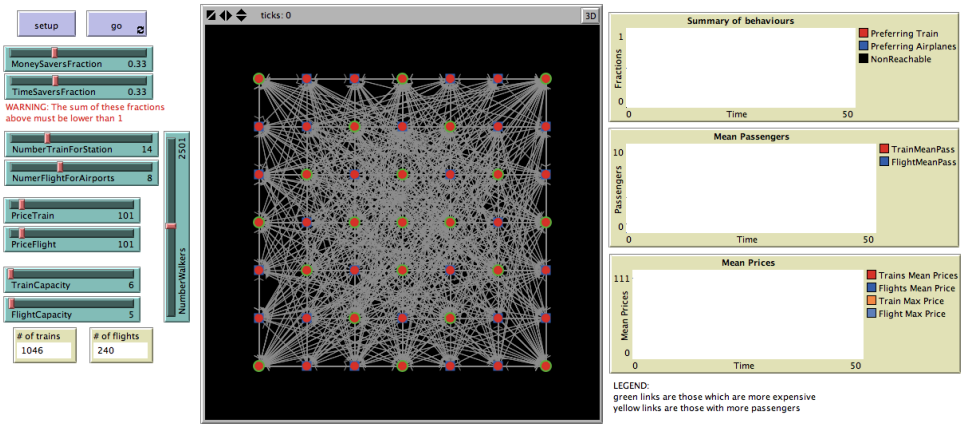
Figure 5: Results obtained if all people are equipartitioned between the 3 possible preferences.

5.4.5 All people equipartitioned in the three categories

The last scenario considered is the one in which all the population is equipartitioned in the 3 possible categories of preference. This scenario gives us results quite similarly to the previous one, the only interesting aspect is that the averages of passengers related to the two means of transportation are very close to each other as showed in 5.4.5.

6 How to enhance the project

The first big step ahead of this project is to enhance the ability of walkers to find different path linking their location to their destination, for example allowing them to compare more than 1 travel for each type (only train, only flight, 1 flight and 1 train or 1 train and 1 flight) until the possibility to compare all the possible combinations of links that allows to get to the destinations including those that need more than 2 links, which is now a limitation.
Moreover must be considered that until now what we modelled is an artificial and simplified version of the world, but nothing binds this to the real world, thus should be interesting to set the positions of the railStations and of the airports including in the model spatial properties like distances that walkers have to go through maybe including this parameter in the choice behaviour of people.
An other interesting development should be to set prices of trains and flights in order to reproduce the real offer range available, this should be also able to model the fact that some destination are more frequent than others.
Also about modeling the behaviours of travellers the project can be improved, since in our model we fix the preference and once fixed the walkers are not able to consider border line situation, like the case of competing path to reach a destination such that price of a flight is just a little more expensive than a train travel but the time gain is relevant. This task can be probably fixed through a sort of *Utility function* that takes as argument price time and number of stopover and gives a real number. In the end must be considered that in our model all prices are updated daily and there are no possibility of predetermination of travel, so no booking are considered it means that prices are updated only depending on their last amount of passengers, whereas in real world price of travel change as long as the departure is coming allowing people who book in advance to save money. All this stuff are oriented to more accurate simulations of the real mobility.


setup
go
MoneySaversFraction 0.33
TimeSaversFraction 0.33
WARNING: The sum of these fractions above must be lower than 1
NumberTrainForStation 14
NumerFlightForAirports 8
PriceTrain 101
PriceFlight 101
TrainCapacity 6
FlightCapacity 5
# of trains 1046
# of flights 240
NumberWalkers 2501
ticks: 0
3D
Summary of behaviours
Fractions
Time
Preferring Train
Preferring Airplanes
NonReachable
Mean Passengers
Passengers
TrainMeanPass
FlightMeanPass
Mean Prices
Trains Mean Prices
Flights Mean Price
Train Max Price
Flight Max Price
LEGEND:
green links are those which are more expensive
yellow links are those with more passengers