Introduction to Netlogo

Agent-based simulation

Netlogo
Modeling complex systems

- Programmable modeling environment for simulating natural and social phenomena
  - Well suited for modeling complex systems evolving over time
  - Hundreds or thousands of independent agents operating concurrently
  - Exploring the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals

Netlogo
Modeling complex systems

- Easy-to-use application development environment
  - creating custom models and quickly testing hypotheses about self-organized systems
  - simple scripting language
  - user-friendly graphical interface
  - runs on JVM

Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL

Netlogo

Practical Info

- Download link:
  – http://ccl.northwestern.edu/netlogo/download.shtml
  – Launch Netlogo through command line:
    - $ /{netlogo_download_folder}/netlogo.sh
- Online doc:
  – https://ccl.northwestern.edu/netlogo/docs/
- Book for agent-based modeling (special focus on Netlogo):
  – http://www.intro-to-abm.com/

Netlogo

History snapshot

- LOGO (Papert & Minsky, 1967)
  - theory of education based on Piaget's constructionism ("hands-on" creation and test of concepts)
  - simple language derived from LISP
  - turtle graphics and exploration of "micro-worlds"
- StarLogo (Resnick, 1991), MacStarLogo, StarLogoT
  - agent-based simulation language
- NetLogo (Wilensky, 1999)
  - further extending StarLogo (continuous turtle coordinates, cross-platform, networking, etc.)

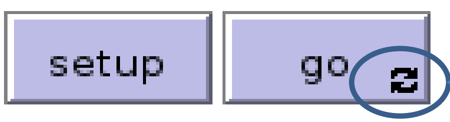
Netlogo
The world of Netlogo

- NetLogo is a 2-D world made of 4 kinds of agents:
  - *Patches* - make up the background or “landscape”
  - *Turtles* - move around on top of the patches
  - *Links* - connect two turtles
  - *The Observer* - oversees everything going on in the world

Graphical Interface

Controls

- Controls allow to run and manage the flow of execution
  - Buttons: initialize, start, stop, step through the model
    - "Once" button execute one action
    - "Forever" button repeat the same action until pressed again



  - Functions with the name of the buttons specify the action executed on click
  - Command centre: ask agents to execute specific commands "on the fly"

Graphical Interface

Settings

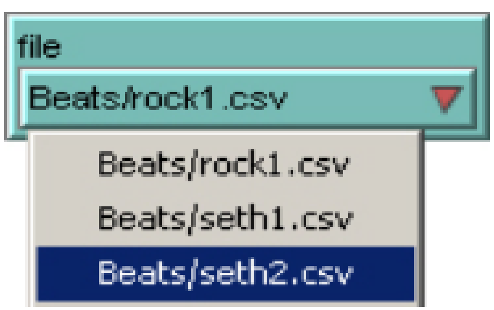
- Settings allow to modify parameters
  - Sliders: adjust a quantity from *min* to *max* by an *increment*

population = 126

  - Switches: set a Boolean variable

incentivi_installazione? = false

  - Choosers: set a value from a list

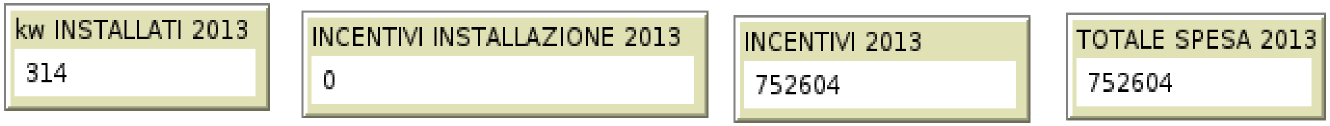


file = “Beats/seth2.csv”

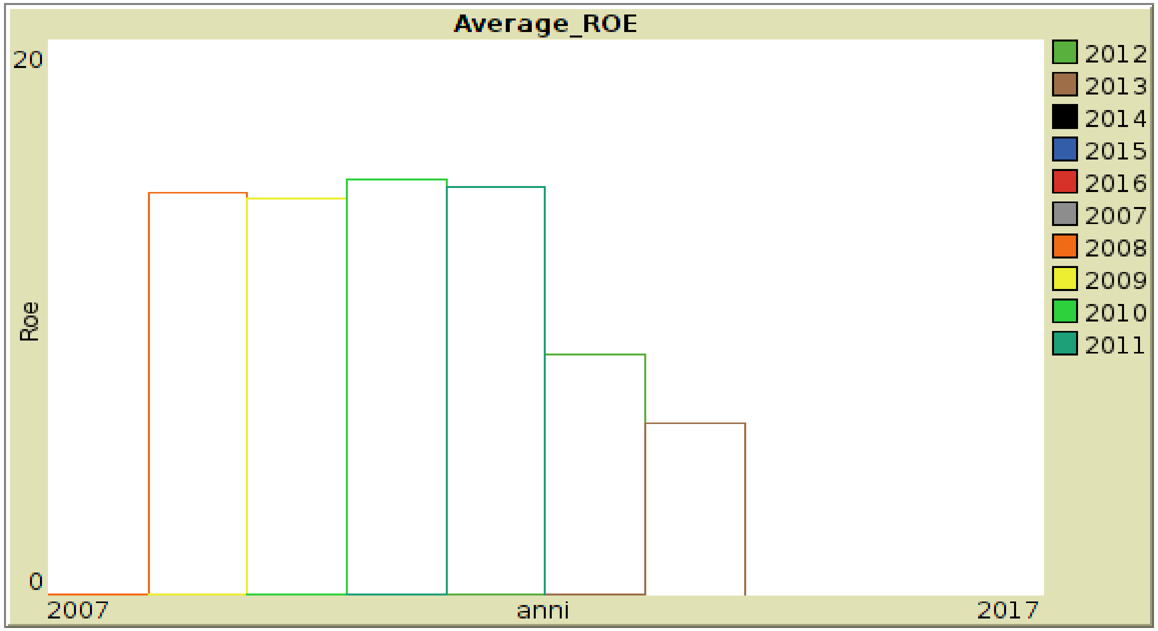
Graphical Interface

Views

- Views allow to display information
  - Monitors display the current value of variables



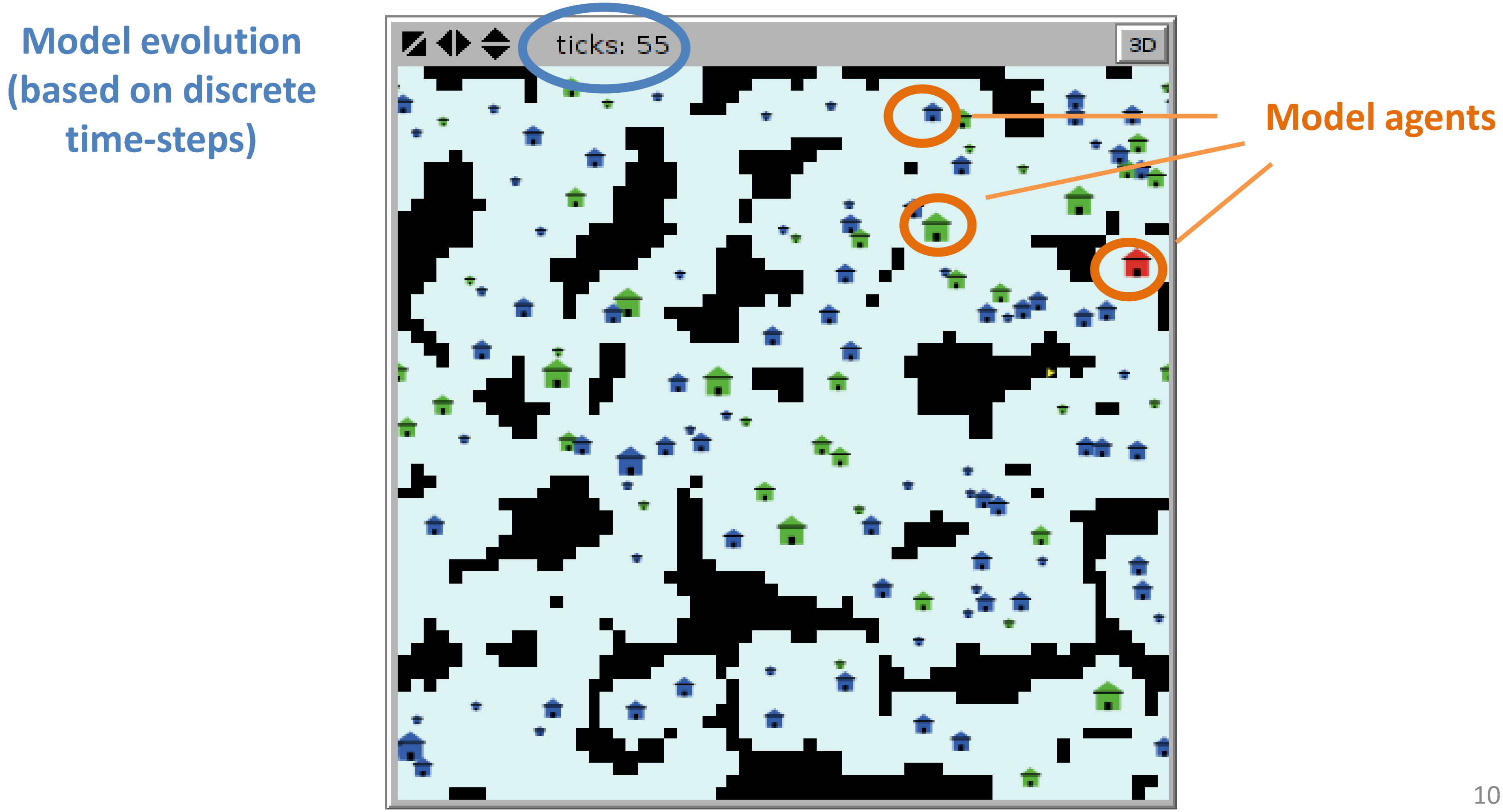
  - Plots: display the history of a variable’s value



  - Output text areas, log text info

Graphical Interface
Views

- Graphic window, the main view of the 2-D Netlogo world



Programming Concepts
Agents

- Agents carry out their activity, all simultaneously
  - Patches don't move, form a 2-D wrap-around grid, have integer coordinates (*pxcor*,*pycor*)
  - Turtles move on top of patches (not necessarily in their centre), have decimal coordinates (*xcor*,*ycor*) and orientation (*heading*)
  - Observer can create new turtles, can have read/write access to all the agents and variables

Programming Concepts

Procedures & Functions

- Commands ("to" keyword)
  - Action for the agents to carry out ("void" functions)
  - Example with 2 input arguments:

```
to draw-polygon [ num-sides size ]
      pd      ;; pen down, draw
      repeat num-sides
      [     fd size    ;; forward 'size' steps
            rt (360 / num-sides) ]   ;; rotate
end
```

Programming Concepts

Procedures & Functions

- Reporters ("to-report")
  - Report a result value
  - Example with 1 input arguments:

```
to-report absolute-value [ number ]
      if-else number >= 0
      [ report number ]
      [ report 0 - number ]
end
```

- Primitives
  - *Built-in* command or reporters
  - Some have an abbreviated form (create-turtle <--> crt)
- Procedures
  - *Custom* commands or reporters (user made)

Programming Concepts
Variables

- Variables – places to store values
  - Global variables: only one value for the variable and every agent can access it
  - Turtle and Patch variables: each turtle/patch has its own value for every turtle/patch variable
  - Local variables: defined and accessible only inside a procedure (scope=narrowest square brackets or procedure itself)

Programming Concepts
Variables

- Built-in variables
  - Ex. turtle variables: color, xcor, ycor, etc.
  - Ex. Patch variables: pcolor, pxcor, etc.
- Custom variables
  - Defining global variables

```
global [ clock ]
```

  - Defining turtle/patch variables

```
turtles-own [ energy speed ]
patches-own [ friction ]
```

Programming Concepts

Variables

- Custom variables
  - Defining global variables
  - Defining turtle/patch variables
  - Defining local variables:
    - **`let variable value`**
    - Creates a new local variable and gives it the desired value

```
to swap-colors [ t1 t2 ]
    let temp color-of t1
```

  - Setting a variable values (after its definition):
    - **`set variable value`**

Programming Concepts
Ask

- *Ask* – specify commands to be run by turtles or patches
  - Asking all turtles

```
ask turtles [ … ]
```

  - Asking all patches
  - Asking *N* turtles

```
ask n-of N turtles [ … ]
```

- Observer code *cannot* be inside any “ask” block

Programming Concepts

Variables

- Setting variables
  - Setting the color of all turtles

    ```
    ask turtles[ set color red ]
    ```

  - Setting the color of all patches

    ```
    ask patches[ set pcolor red ]
    ```

  - Setting the color of the patches under the turtles

    ```
    ask turtles [ set pcolor red ]
    ```

  - Setting the color of one turtle (identify by ID)

    ```
    ask turtle 5 [ set color green]
    ```

  - or

    ```
    set color-of turtle 5 green
    ```

  - Setting the color of one patch (identified with coordinates)

    ```
    ask patch 2 3 [ set pcolor green ]
    ```

Programming Concepts
Agent sets

- Agent set, definition of a subset of agents (not a keyword)
  - All blue turtles

    ```
    turtles with [ color = blue ]
    ```
  - All blue turtles on the patch of the current caller (patch or turtle)

    ```
    turtles-here with [ color = blue ]
    ```
  - All turtles less than 5 patches away from caller

    ```
    turtles in-radius 3
    ```
  - The 4 patches to the east, north, west and south of the caller

    ```
    patches  at-points   [[1 0] [0 1] [-1 0]
    [0 -1]]
    ```

Programming Concepts

Agent sets

- Using agent sets
  - Ask such agents to execute a command

```
ask <agentset> [ … ]
```

  - Check if there are such agents

```
show any? <agentset>
```

  - Count such agents

```
show count <agentset>
```

- Ex. - remove the richest turtle (with the maximum “assets” value):

```
ask max-one-of turtles [sum assets][die]
```

agentset: `max-one-of turtles [sum assets]`

command/action: `[die]`

Programming Concepts

Breeds

- *Breed*, a “natural” kind of agent set (other species than turtle)

```
breed [ wolves sheep ]
```

- A new breed comes with automatically derived primitives:

```
create-<breed>
create-custom-<breed>
<breed>-here
<breed>-at
```

- The breed is a turtle variable

```
ask turtles 5 [ if breed=sheep ]
```

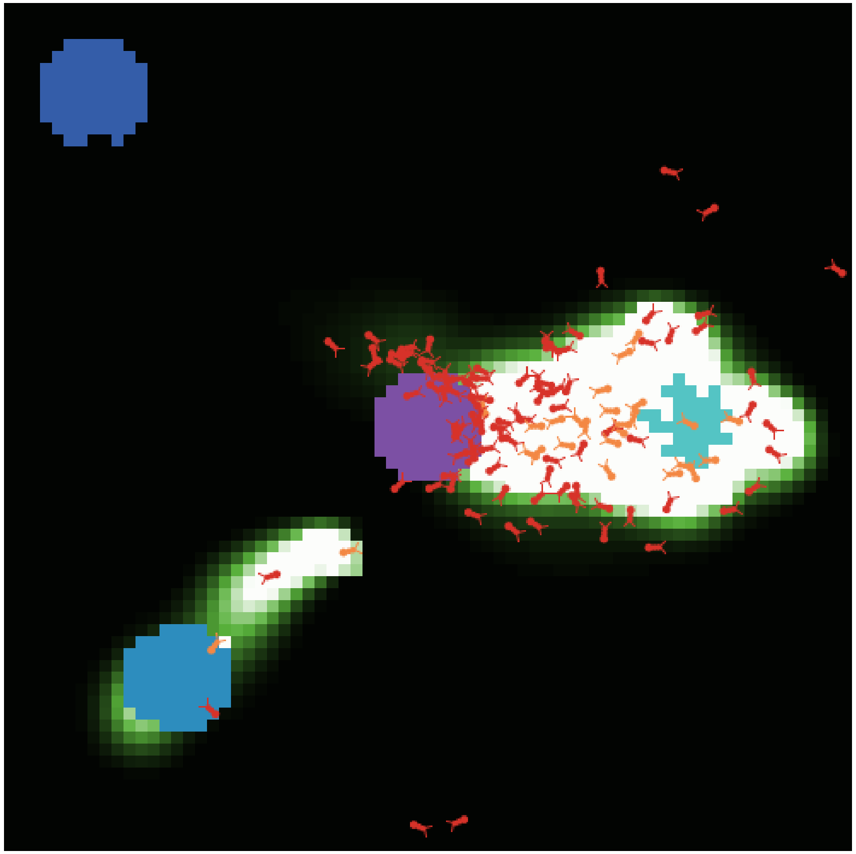
- A turtle agent can change breed

```
ask turtles 5 [ set breed sheep ]
```

Exercise 1

Basic Ants Model

- Very simple model as a first "hands-on" experience
- A colony of ants forages for food
  - Tough each ant follows a set of simpler rules, the colony as a whole act in a sophisticated way



Wilensky, U. (1997). NetLogo Ants model. http://ccl.northwestern.edu/netlogo/models/Ants. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Exercise 1

Ants Model

- When an ant finds a piece of food, it carries the food back to the nest, dropping a chemical as it moves
- When other ants “sniff” the chemical, they follow the chemical toward the food
- As more ants carry food to the nest, they reinforce the chemical trail

Exercise 1

Model Usage

- Click the SETUP button to set up the ant nest (in violet, at center) and three piles of food then click the GO button to start the simulation.
  - The chemical is shown in a green-to-white gradient.

- The EVAPORATION-RATE slider controls the evaporation rate of the chemical. The DIFFUSION-RATE slider controls the diffusion rate of the chemical.
- If you want to change the number of ants, move the POPULATION slider before pressing SETUP

Exercise 1

Things to notice

- The ant colony generally exploits the food source in order, starting with the food closest to the nest, and finishing with the food most distant from the nest
- It is more difficult for the ants to form a stable trail to the more distant food, since the chemical trail has more time to evaporate and diffuse before being reinforced

Exercise 1
Things to notice

- Once the colony finishes collecting the closest food, the chemical trail to that food naturally disappears, freeing up ants to help collect the other food sources.
  – The more distant food sources require a larger "critical number" of ants to form a stable trail.
- The consumption of the food is shown in a plot.
  – The line colors in the plot match the colors of the food piles.

Exercise 1

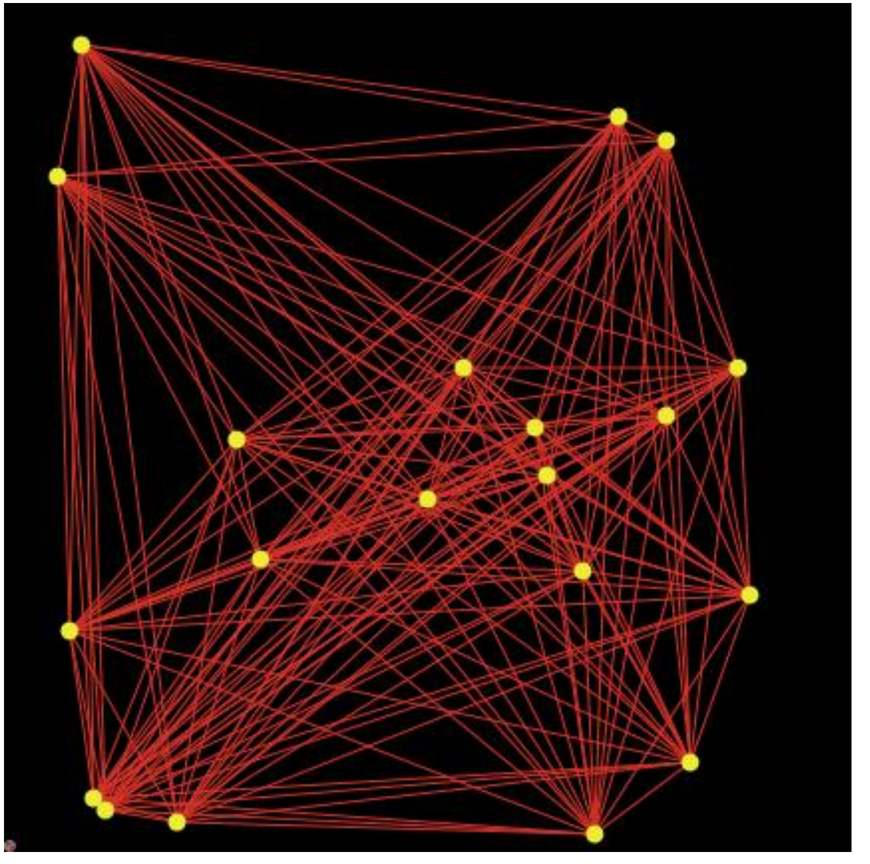
Model Extensions

1. Try different placements for the food sources
   – What happens if two food sources are equidistant from the nest?
2. In this project, the ants use a “trick” to find their way back to the nest: they follow the “nest scent.”
   – Real ants use a variety of different approaches to find their way back to the nest.
   – Try to implement some alternative strategies.
3. In the uphill-chemical procedure, the ant “follows the gradient” of the chemical. That is, it “sniffs” in three directions, then turns in the direction where the chemical is strongest.
   – Try variants of the uphill-chemical procedure, changing the number and placement of “ant sniffs.”

Exercise 2

Ant Colony Optimization e TSP

- Goal: implementing the Ant System algorithm[1] and use it to solve the Traveling Salesman Problem
- Based on the observation of ants behaviour
  - Positive feedback based on pheromone tracks which reinforce the best solution components



[1] *Dorigo, M., Maniezzo, V., and Colorni, A., The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics, Vol. 26, No. 1. (1996), pp. 29-41. http://citeseer.ist.psu.edu/dorigo96ant.html*

Exercise 2
Ant Colony

- As you already know
  - Ants leave a pheromone trail while going from the nest to food sources (and vice versa)
  - Ants tend to choose (with higher probability) routes with greater amount of pheromones
  - Cooperative interaction which leads to an emergent behaviour, that is finding the shortest path

Exercise 2
ACO

- Probabilistic model (*pheromone* model) used to recreate the pheromone trails left by ants
- Ants incrementally build the component of a solution
- Ants perform stochastic steps on a fully connected graph (*construction graph*)
- Constraints used to obtain a feasible solution

Exercise 2
ACO and TSP

- A possible model for TSP:
  - The graph nodes are the city to visit (the components of a solution)
  - The edges are the connections between the cities
  - A solution is a Hamiltonian circuit in the graph
  - Constraints are used to avoid loops, so that an ant can visit a city exactly once

Exercise 2

Information sources

- Edges or vertexes (or both) have two information:
  - Pheromone τ, which stands in for natural trail left by ants and represents the long term memory of ants in relation to the global search process
  - Heuristic value η, i.e. the a priori knowledge on the problem

Exercise 2
ACO System

- The ants follow a path on the construction graph and build a solution
- They used a transition (probabilistic) rule to choose the next node to visit
- Both pheromone and heuristic are taken into account
- Pheromone values are adjusted based on the quality of the solution found

Introduction to Netlogo

End

Additional info, questions, etc..

andrea.borghesi3@unibo.it