# PUPIBinary

A tool for binary function optimisation
inspired in the behaviour of urban pigeons

User Guide

Version 1.4

**Sergio A. Rojas**, PhD.

*Universidad Distrital Francisco José de Caldas*

Bogotá, Colombia, 2020

PUPIBinary Version 1.4 - User guide.

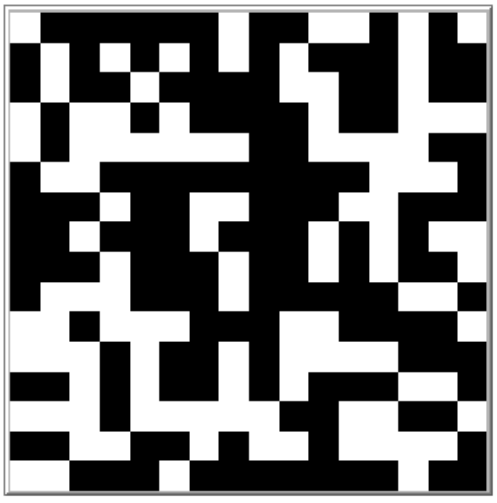


*First Edition, August 2020*
*Bogotá, Colombia*

# Overview

PUPIBinary is a software tool designed to find approximate solutions to optimisation problems whose decision variables take discrete values in the binary domain, in other words, variables taking values 0 or 1. The method used by PUPIBinary to find a solution, is derived from a recently proposed particle swarm algorithm inspired on the foraging behaviour of urban pigeons. The new variant of the algorithm is obtained by mapping the real–valued search space of the original version into a discrete binary–valued encoding. Furthermore, the extended version was adapted to the framework of an agent–based model in order to develop this software tool.

The approach to translate the original pigeon-inspired method consist of separating the search space of real–valued vector of pigeon locations from the solution space of binary-valued variables of the optimisation problem. This can be done by means of an encoding function, a technique known as genotype–to–phenotype mapping widely used in evolutionary computation algorithms. In the current version, the mapping is performed by applying a cut-off threshold to the coordinates of the pigeons' locations. In this way, the tool is able to solve high-dimensional binary problems of up to 256 variables.



PUPIBinary v1.4 has been released under GNU General Public License (GPLv3); it is available online at:

http://modelingcommons.org/browse/one_model/6400

# Contents

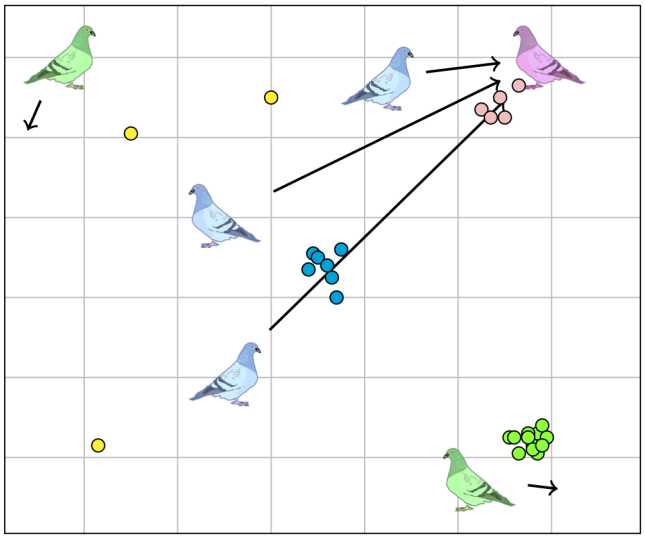
# Chapter 1

# The optimisation tool

## 1.1 What is PUPIBinary?

PUPIBinary is a software tool developed as an extension of an agent-based model for numeric real-valued unconstraint optimisation to binary domains, inspired in the foraging behaviour of urban pigeons (see [1] for details). Optimisation problems with binary domains contemplate variables taking values 0 or 1, representing decisions associated to conditions being false or true. Hence, these kind of domains are of interest in many applications in engineering and management.

The tool maintains an internal representation of the pigeons locations which is transformed to binary values via a threshold function. The movement rules of the internal representation are identical to the continuous-valued version, and the inspiration of the original method regarding foraging strategies of this species, is sustained: a leader pigeon (coloured fuchsia) located at a promising source of food is pursued by a flock of follower pigeons (coloured blue), while simultaneously other pigeons are walking around (coloured green), exploring the space for new sources of food too. Food sources in this context refers to maximal values of the binary cost function that is being optimised.



## 1.2 How it works

A benchmark of binary problems is included. Each PROBLEM consists of finding a bitstring that maximises a cost function. In the current version, the tool features three functions: "oneMax", "powSum" and "squareWave", with the following definitions:.

| Problem | Definition | Description |
|---|---|---|
| oneMax | $f(\mathbf{b}) = \sum_{i=1}^{d} b_i$ | Counts the total number of ones in the vector **b**. The optimum is: $f(\underbrace{1,1,\ldots,1}_{\leftarrow d \text{ times} \rightarrow}) = d$ |
| squareWave | $f(\mathbf{b}) = \sum_{i=1}^{d} \left(1 - \texttt{abs}\left(b_i - \left(2\left\lfloor \frac{i}{T} \right\rfloor - \left\lfloor \frac{2i}{T} \right\rfloor\right)\right)\right)$ | Computes the similarity of vector **b** to a binarised version of a sinusoidal wave with period $T = \sqrt{d}$. The optimum is: $f(\underbrace{0,0,0}_{\leftarrow \frac{d}{T} \text{ bits} \rightarrow}, \underbrace{1,1,1}_{\leftarrow \frac{d}{T} \text{ bits} \rightarrow}, \ldots, \underbrace{0,0,0}_{\leftarrow \frac{d}{T} \text{ bits} \rightarrow}, \underbrace{1,1,1}_{\leftarrow \frac{d}{T} \text{ bits} \rightarrow}) = d$ |
| powSum | $f(\mathbf{b}) = \sum_{i=1}^{d} b_i \times \log_2 2^{i-1}$ | Computes the sum of the powers corresponding to positions set to one in the binary vector **b**. The optimum is, likewise oneMax, the all-ones vector: $f(\underbrace{1,1,\ldots,1}_{\leftarrow d \text{ times} \rightarrow}) = \sum_{i=0}^{d} i = \frac{d(d+1)}{2}$ |

Pigeons are characterised by an internal array of continuous-valued coordinates of size DIM. These coordinates are mapped to binary coordinates via a cut-off threshold called TAU. This is a mechanism similar to the genotype-to-phenotype mapping widely used in evolutionary algorithms (implemented in the tool as the GPM code block). Two types of pigeon breeds were defined, namely followers and walkers. The initial population is created with an amount of pigeons given by the parameter POP-SIZE, with the subset of walkers assigned randomly in proportion to the parameter WALKERS-RATE; the remainder pigeons are assigned to the subset of followers.

At each step of the simulation, the tool performs four simple actions: find the leader, move the followers, move the walkers and update the best solution found so far. These actions correspond to the following code routines implemented in the software:

- UPDATE-LEADER: chooses as leader pigeon the one having the best fitness and updates the best fitness ever if necessary. Fitness is computed with the aforementioned cost functions depending on the chosen problem.

- FOLLOW-MOVE: moves each follower towards the leader in one randomly chosen dimension, with a step-size ALPHA plus a random shift in its orientation due to wind or collisions.

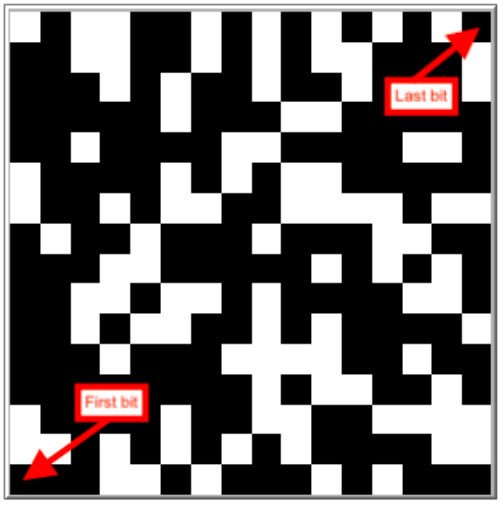
- WALK-MOVE: moves each walker around randomly with a step-size SIGMA, again in one randomly chosen dimension). The last two movement rules correspond to the exploration/exploitation mechanisms of the optimisation algorithm.

The simulation is terminated either after a maximum number of steps, MAX-STEPS, or when the truth optimal solution is found prematurely. Besides, the software was implemented using the special–purpose ABM developing platform NetLogo 6.1.0 (see [2]).

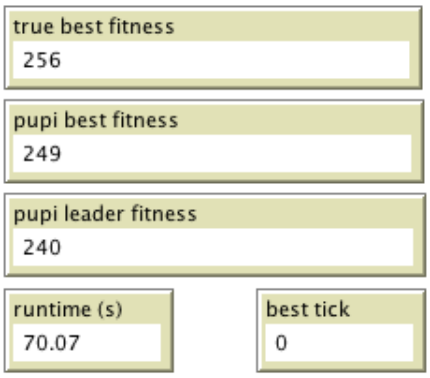
## 1.3 How to use it


Firstly choose an optimisation PROBLEM to be solved, from the pull-down list. For any of these problems, then define a particular dimension DIM. Additionally, choose the algorithm parameters POP-SIZE, WALKERS-RATE, TAU, ALPHA and SIGMA. You can also set the termination criterium MAX-TICKS. Then press SETUP, then GO.

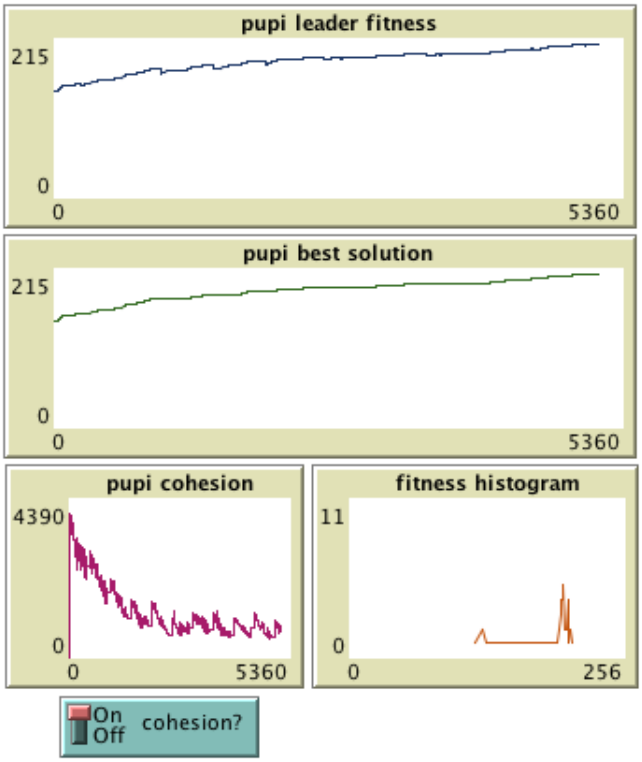
The initial genotypes of the population of pigeons will be assigned randomly and their phenotypes would be obtained. Afterwards, at each time step pigeons move according to its role, the population fitness is updated, and if needed, the leader is re-assigned. The simulation view area will display the phenotype, or bitstring, of the best pigeon found so far. This bitstring is reshaped and displayed as a 2D grid of binary cells (black cells correspond to bits set to 1 in the bitsring, whereas white cells to bits set to 0). The first bit in the bitstring is located in the bottom-left corner of the grid, and the last bit in the upper-right corner, as it is illustrated below:



The output monitors show the fitness (cost) of the true solution for the problem, the best fitness ever found by the algorithm during the simulation, and the fitness associated to the current leader. If the algorithm is able to find the true solution, the simulation stops and the BEST-TICK and RUNTIME monitors will display a "!!!" sign inserted behind their actual values. Otherwise the simulation finishes when MAX-TICKS are elapsed.




Lastly, the tool also outputs the plot of the leader fitness vs time, the plot of fitness of the best solution found vs time, the histogram of fitness of the population and the plot of flock cohesion vs time if the COHESION? switch is enabled. The latter implies an additional cost to the running time, as the software needs to compute distances between all the pigeons in the follower's flock.



## 1.4 Other distinctive features

Although the actual locations of the pigeons can not be visualised (because the problems have high-dimensional search spaces), one can notice that the leader and the flock move out from one local minima to another. This occurs because every certain number of ticks, the entire population become walkers wandering about other regions of the search space. This phenomenon is explained by the fitness variation of the leader pigeon during the simulation timeline, as it can be seen in the corresponding plot. Nonetheless, the best found ever solution always has an increasing fitness as it can be verified in its respective plot. Besides, the flock formation behaviour is suggested by the periodic patterns that appear in the cohesion plot.

Lastly, we remark that the pattern of solutions showed in the view area resemble a blacked display for the "oneMax" and "powSum" problems, and a half-and-half black and white display for the "squareWave" problem, as it was expected from their mathematical definitions. Notice that for the "powSum" problem, the last bits to be set are those in the bottom of the view (the least significant loci of the bitstring), as they contribute the least to its cost function.

## 1.5 Try it yourself

Experiment with different DIM sizes for each problem and compare if a different configuration of parameters is needed. For starters, we suggest the following settings:

- DIM = 100
- POP-SIZE = 40,
- MAX-TICKS = 40000,
- TAU = 0.5
- WALKERS-RATE = 0.2,
- ALPHA = 0.9,
- SIGMA = 0.1,

## 1.6 Extending the tool

An interesting question arising is if the convergence speed of the algorithm can be improved without compromising its simplicity for practical purposes, for example using dynamic updates of the step sizes of pigeon movements. Another related idea worth of exploring is if convergence speedup an be achieved by hybridising the model with local search techniques.

Finally, it would be appealing to include additional binary problems and try to solve them with this tool. For this purpose, users only need to add a code block that computes the cost function of the new problem, based on the bitstring phenotype of the pigeons, in addition to the TRUE-BEST-FITNESS value in the SETUP-PROBLEM procedure. We suggest for example, studying combinatorial problems such as the N-queens or the Knapsack problem.

## 1.7 References

[1] Rojas-Galeano, S. (2019). Binary optimisation with an urban pigeon-inspired swarm algorithm. In: Workshop on Engineering Applications. Springer CCIS Series. https://link.springer.com/chapter/10.1007/978-3-030-31019-6_17.

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

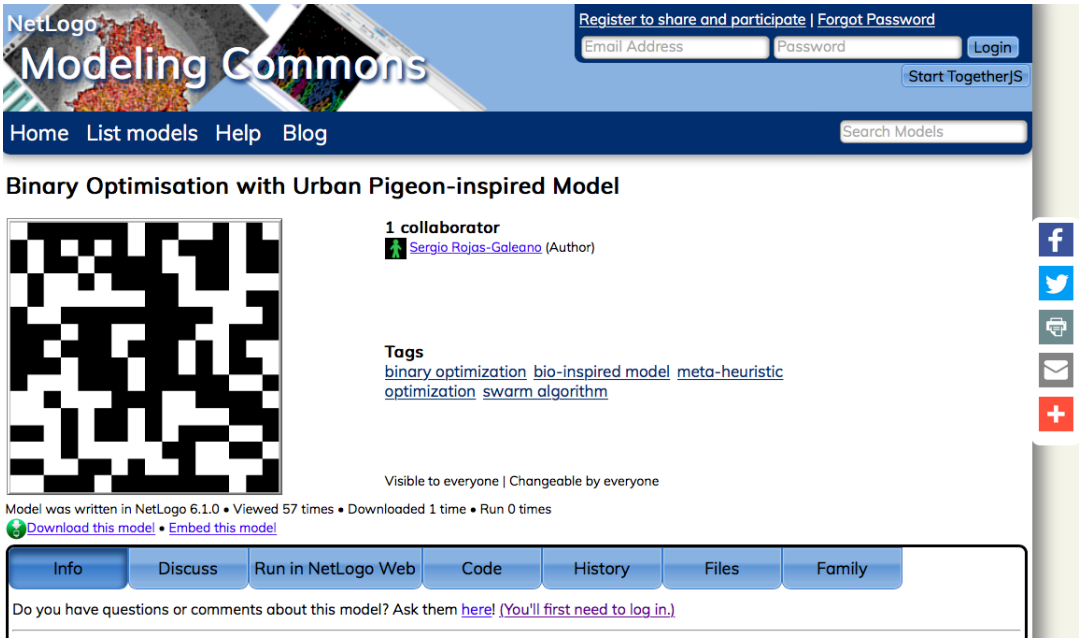
# Chapter 2

# Installation and execution

## 2.1 Online version

The easiest way of experimenting with PUPIBinary is by using its online version. The software is available at the ModellingCommons website. So, you just need to follow these steps:

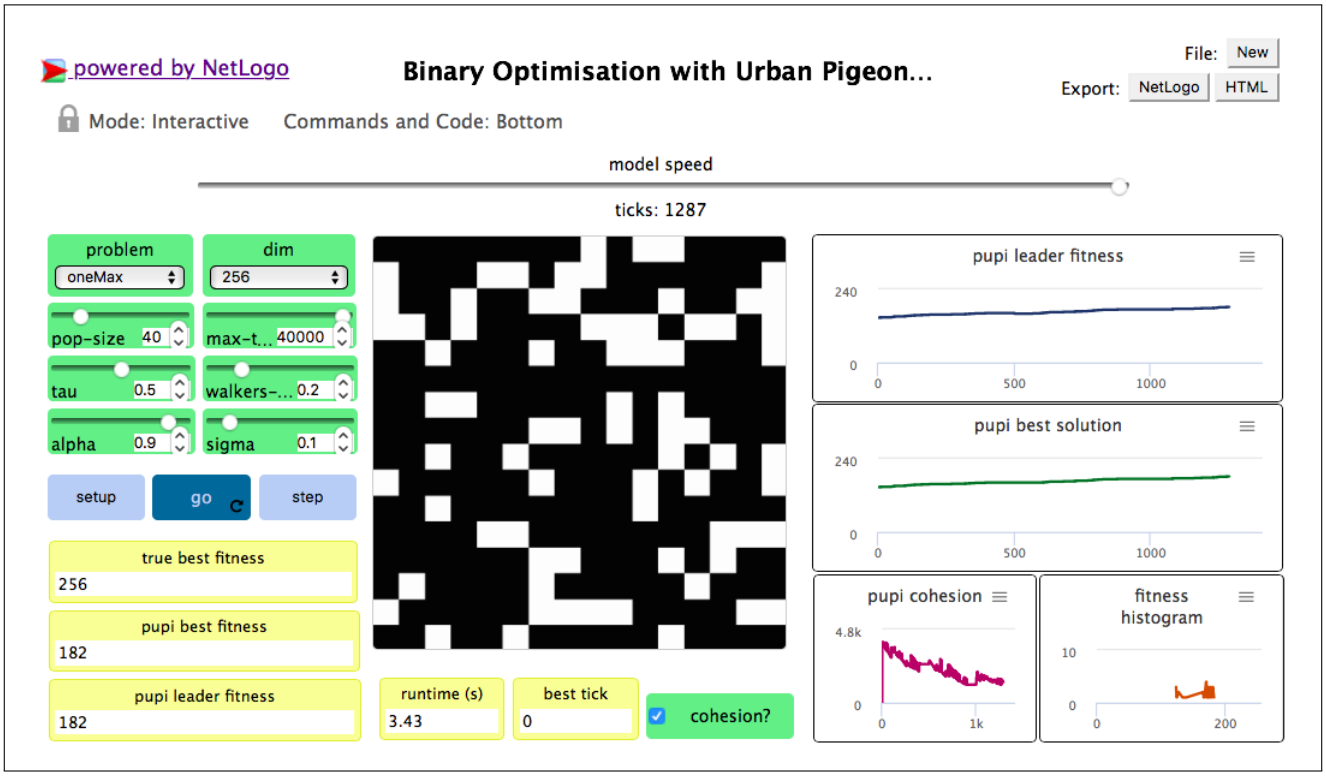
1. Open your favourite Internet browser and point it to the following URL: http://modelingcommons.org/browse/one_model/6400
2. The following web page should appear:




3. From the toolbar, choose the "Run in Netlogo Web" tab:

4. A grey area in the middle of the screen is shown. Do “Click to Run Model”:


5. The model main screen will show up:

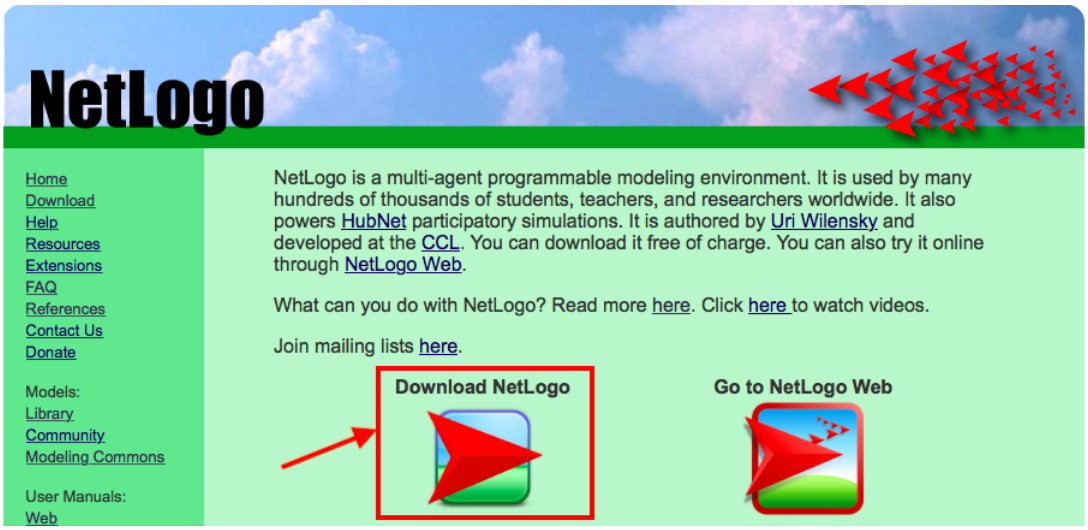



6. That’s all! Choose the running parameters in the control panel, click SETUP and then GO! The flock of pigeons will start the search in the genotype space in order to solve the chosen problem, while the view area will display the phenotype of the best solution found ever and the monitors and plots will show the performance indicators of the adaptation process to the corresponding binary cost function.

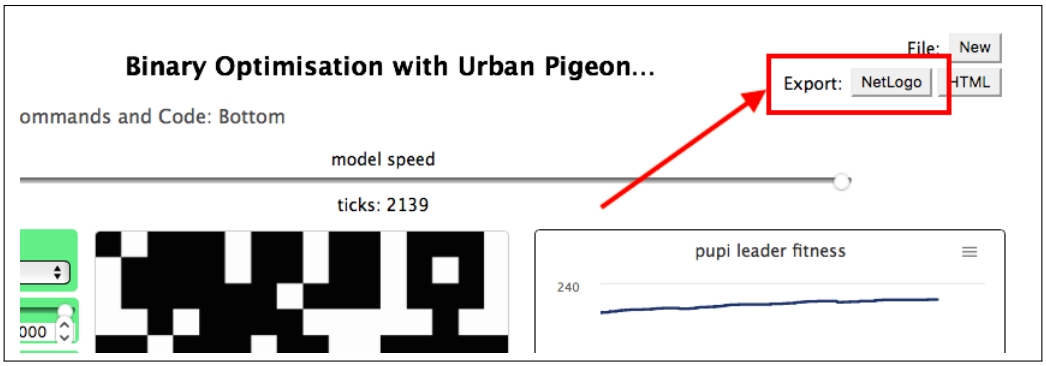
## 2.2 Desktop version

The desktop version is recommended if you want to try heavy experimentation, such as parameter tuning, average behaviour of multiple runs or simulations with large populations. For this purpose, `PUPIBinary` runs over the NetLogo desktop simulation platform [2]. In this case, you need to go through the following steps:

1. Download and install the NetLogo desktop software. For this purpose, go to `http://ccl.northwestern.edu/netlogo/`, click in "Download NetLogo" and follow the instructions:

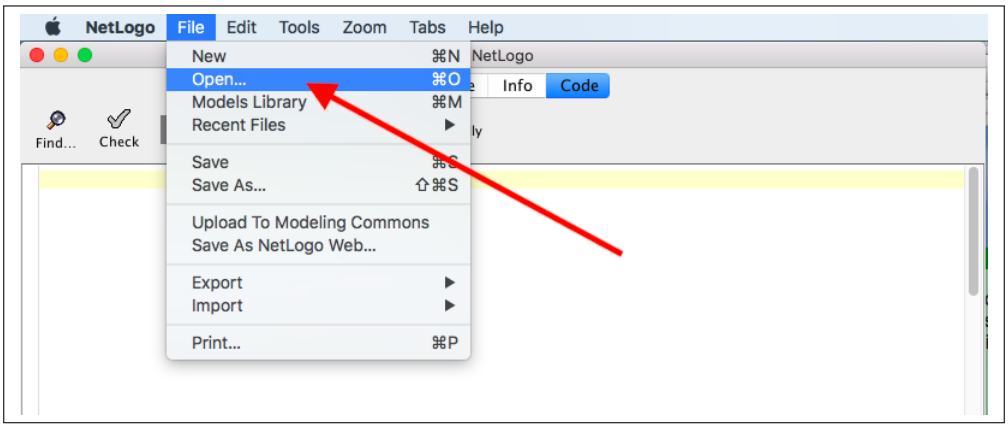



2. Download the `PUPIBinary` model file from the tool webpage, using the "Export" button:



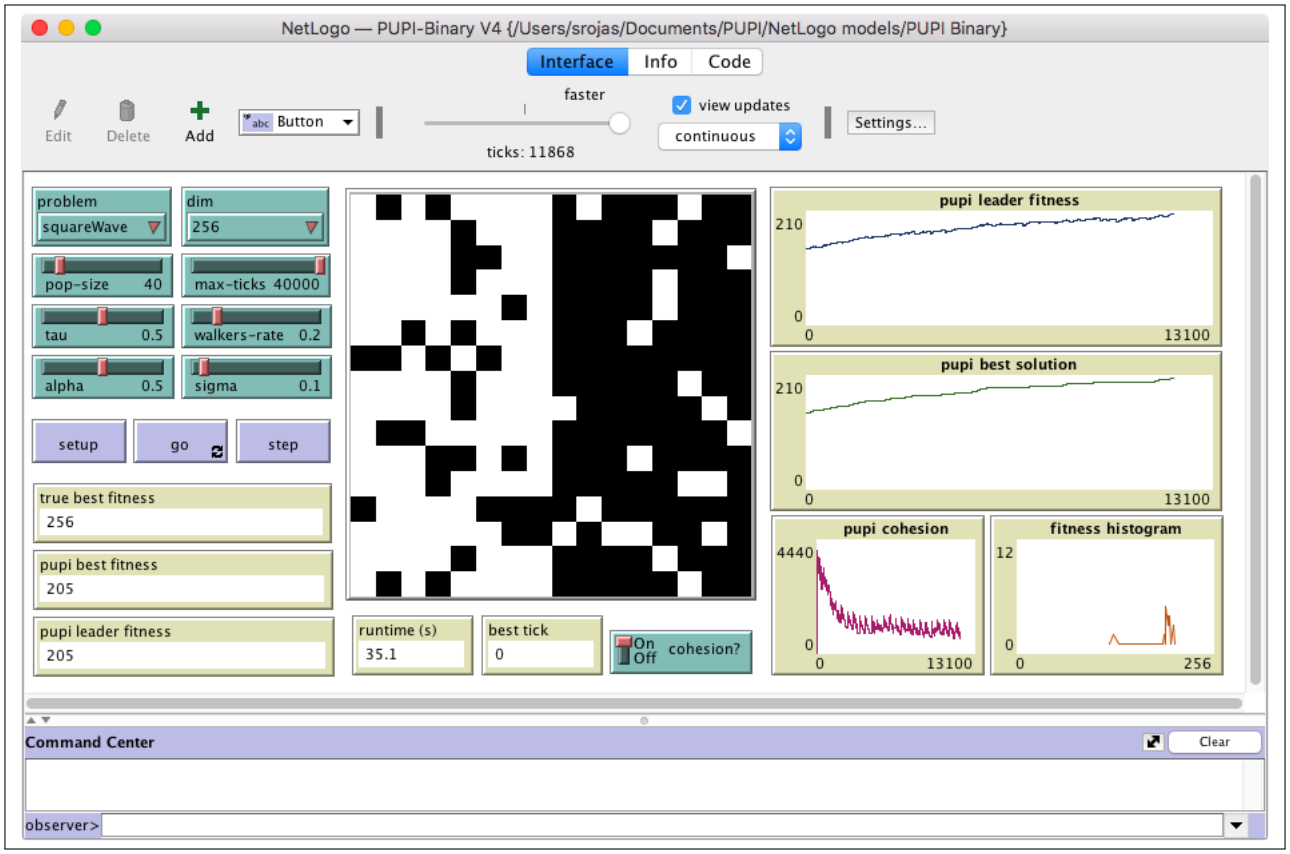

A file called *Binary Optimisation with Urban Pigeon-inspired Model.nlogo* would be downloaded to your local disk.

3. Run NetLogo on your computer. Choose the menu option *File* → *Open*:




Locate the `PUPIBinary` .nlogo file that you downloaded previously and open it.

4. The PUPIBinary main screen will show up:




That's it! Choose the running parameters in the control panel, click SETUP and then GO! The flock of pigeons will start the search in the genotype space in order to solve the chosen problem, while the view area will display the phenotype of the best solution found ever and the monitors and plots will show the performance indicators of the adaptation process to the corresponding binary cost function.

# Chapter 3

# Source code

```
;; ------------------------------------------------------------
;; Binary version of PUPI optimiser
;;
;; A model by Sergio Rojas-Galeano
;; v1.4 Copyright (c) July 2020 The author
;; Correspondance email: srojas@udistrital.edu.co
;; Universidad Distrital Francisco Jose de Caldas, Bogota,
    Colombia
;;
;; An extension of Urban Pigeon-inspired Model for Unconstraint
    Optimisation
;; v1.16 (2020) by Sergio Rojas-Galeano and Martha Garzon
;; http://modelingcommons.org/browse/one_model/6390
;;
;; This program is free software: you can redistribute it and/or
    modify
;; it under the terms of the GNU General Public License (GPLv3)
;; (see license at: https://www.gnu.org/licenses/gpl-3.0.txt)
;;
;; The model is made publicly available in the hope that it will
    be useful
;; to modelers, but WITHOUT ANY WARRANTY whatsoever (see license
    for details).
;; ------------------------------------------------------------

;; Definition of global variables
globals[
  ;; PUPI globals
  pigeons               ; agentset of all pigeons
     (walkers+followers)
  pupi-leader           ; best pigeon in current iteration
  pupi-best-solution    ; best solution found by PUPI ever
  pupi-best-fitness     ; fitness of best solution found ever
  pupi-best-tick        ; tick where best solution was found
  pupi-best-time        ; runtime where best solution was found
  pupi-runtime          ; overall algorithm runtime (ms)
  pupi-cohesion         ; flock cohesion

  ;; Problem globals
```

```
  true-best-fitness     ; the ground-truth fitness of optimum
      solution
  wave-signal
]

;; PUPI breeds
breed [walkers walker]
breed [followers follower]

;; Pigeon attributes for binary problems
turtles-own [
  xcors    ; a d-dimensional list of coordinates (i.e "location"
      of the pigeon)
  xbits    ; a binary mapping of the coordinates
      (genotype-to-phenotype representation)
  fitness  ; value of cost function for the pigeon
]

;; Execute one iteration of the optimisation procedure
to go
  reset-timer
;    ifelse ticks mod 1000 > 800 [
  ifelse ticks mod 500 > 400 [
    ;; PUPI wild search (starvation move)
    ask pigeons [ walk-move ]
  ][
    ;; PUPI normal foraging mode
    ask pigeons [ compute-fitness ]
    update-leader
    ask followers [ follow-move ]
    ask walkers [ walk-move ]
  ]
  set pupi-runtime pupi-runtime + timer

  ;; The following steps do not count as runtime; they are
      ancilliary to the algorithm
  if cohesion? [ set pupi-cohesion sum map [ bits ->
      hamming-distance [xbits] of pupi-leader bits ] [xbits] of
      followers ]
  update-display
  tick
  if (ticks > max-ticks) or ((pupi-best-tick > 0) ) [ stop ]
end

;; Compute fitness of pigeon depending on selected problem
to compute-fitness
  gmp
  run problem     ; call the procedure corresponding to problem
      (see benchmarks below)
end

;; Genotype-to-phenotype mapping
to gmp
  set xbits map [x -> ifelse-value (x < tau) [0] [1] ] xcors   ;
      apply binarisation according to \tau threshold
end
```

```
;; Find current leader and update best solution ever
to update-leader
  set pupi-leader max-one-of pigeons [fitness]

  ;; update solution and statistics if the new leader is fitter
     than ever
  if [fitness] of pupi-leader > pupi-best-fitness [    ;
     maximises by default. Change to "<" to minimise instead
    set pupi-best-solution [xbits] of pupi-leader
    set pupi-best-fitness [fitness] of pupi-leader

    ;; verify stopping condition
    if [fitness] of pupi-leader = true-best-fitness [
      set pupi-best-tick ticks
      set pupi-best-time pupi-runtime
    ]
  ]
end

;; Move followers towards pigeon leader
to follow-move
  let index random dim  ; choose a random coordinate to modify
  let xi item index xcors
  let delta-xi (item index [xcors] of pupi-leader) - (item index
     xcors)
  set xcors replace-item index xcors ( (xi + alpha * delta-xi) +
     sigma * random-normal 0 0.1 ) ;mod 1
end

;; Move walkers around
to walk-move
  let index random dim  ; choose a random coordinate to modify
  let xi item index xcors
  set xcors replace-item index xcors ((xi + sigma * random-normal
     0 1) mod 1)
end

;; Get ready to go
to setup
  clear-all
  setup-problem
  setup-display
  create-walkers pop-size * walkers-rate       ;; create walkers
     accroding walkers-rate
  create-followers pop-size - count walkers    ;; create
     followers accroding walkers-rate
  set pigeons (turtle-set followers walkers)   ;; populate
     pigeons agentset
  ask pigeons [
    set xcors n-values dim [ random-float 1 ]  ;; set pigeon
       initial random location (coordinates)
    compute-fitness                            ;; set pigeon
       initial fitness value
    hide-turtle                                ;; no need to show
       turtles
```

```
  ]
  initial-solution
  update-leader
  update-display
  reset-ticks
end

;; Define true optimal cost and other global variables
to setup-problem
  set true-best-fitness ( ifelse-value
    problem = "oneMax" [ dim ]                    ; the number of
       bits that must be set on
    problem = "powSum" [ dim * (dim + 1) / 2 ]    ; in a all-ones
       bitstring, the sum of n powers is equals to n(n+1)/2
    problem = "squareWave" [ dim ]                ; the number of
       bits that should be identical to those in the wave signal
  )
  if problem = "squareWave" [
    let root sqrt dim
    let index range dim
    set wave-signal map [ i -> -1 * (2 * int (i / root) - int (2
       * i / root)) ] index
  ]
end

;; Resize view area according to problem dimension
to setup-display
  let side-size sqrt dim
  resize-world 0 (side-size - 1) 0 (side-size - 1)
  set-patch-size 300 / side-size
end

;; Initialise best solution
to initial-solution
  let anyone one-of pigeons       ;; choose any pigeon as initial
     solution
  set pupi-best-solution [xbits] of anyone
  set pupi-best-fitness [fitness] of anyone
end

;; Show current best solution on display
to update-display
  ask patches [
    let index  (max-pxcor + 1) * pycor + pxcor  ; obtain linear
       location of bit corresponding to patch coordinates
    set pcolor ifelse-value (item index pupi-best-solution = 0)
       [white] [black]
  ]
end

;; Compute hamming distance between bitstrings
to-report hamming-distance [xbits1 xbits2]
  report (length remove true (map [ [b1 b2] -> b1 = b2 ] xbits1
     xbits2))
end
```

```
;;;;;;;; Benchmark problems ;;;;;;;;;

;; oneMax computes the sum of the bits
to oneMax
  set fitness (sum xbits)
end

;; powSum computes the sum of power exponents where bits are on
to powSum
  let powers (range 1 (dim + 1) )
  set fitness sum (map [ [power bit] -> power * bit ] powers
     xbits)
end

;; squareWave computes the number of bits identical to those in
   the wave-signal of size dim
to squareWave
  set fitness sum (map [ [wave bit] -> ifelse-value wave = bit
     [1][0] ] wave-signal xbits)
end
```

# Chapter 4

# Software license

PUPIBinary version 1.4