



## CIAO

# Collective Intelligence Algorithm for Optimisation

A tool for numeric optimisation of bound-constraint cost functions using collective intelligence simulation

User Guide

Version 1.24

**Sergio A. Rojas, PhD.**

**Lindsay Álvarez, PhD.**

*Universidad Distrital Francisco José de Caldas*

Bogotá, Colombia, May 2024

CIAO Version 1.25 - User guide

Copyright © 2024 The author

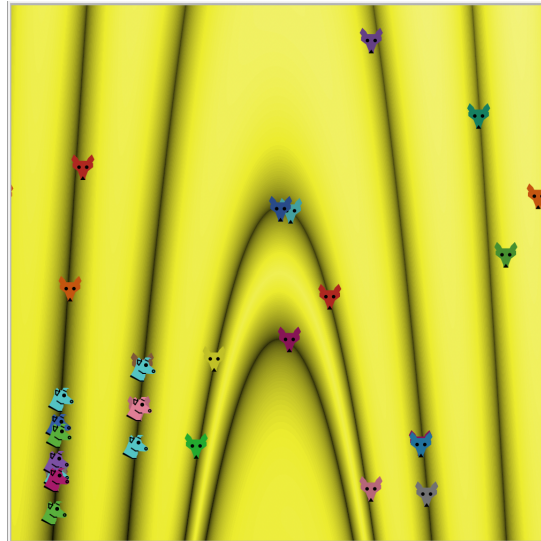
This document is distributed under the CC BY-NC-ND license (*Creative Commons Attribution-Noncommercial-NoDerivatives 3.0*). Any other unauthorised form of distribution, copying, duplication, reproduction, or sale (total or partial) of the content of this document, both for personal and commercial use, will constitute an infringement of copyright. This guide is an original work of its author, and therefore it is protected by the laws that regulate copyright and intellectual property. The opinions and points of view expressed in this document are personal to the author and do not compromise the policies, intentions, strategies, or official position of any other organism, company, organisation, service or person mentioned in it.

The author has made every effort to ensure that this guide is free from errors or omissions. However, the author accepts no responsibility for offence, damage or loss caused to any person acting or endorsing actions using the material contained in this document.

*First Edition, May 2024*  
*Bogotá, Colombia*

# Overview

**CIAO** stands for *Collective Intelligence Algorithm for Optimisation*, and is a tool designed to find approximate solutions to optimisation problems whose decision variables take numeric values in the real domain, inspired in the mechanisms of the collective intelligence genome. The software is designed for an academic audience interested in the field of metaheuristics, as a user-friendly visual tool to conduct simulation experiments with a set of benchmark optimisation problems.



The approach of solving an optimisation problem with a collective of artificial agents undergoing an adaptation process is known as a population-based metaheuristics. Instead of using mathematical analysis of aggregated variables describing the phenomena, this approach resorts to modelling the interaction of a group of agents in a simulated environment and trace the evolution of such variables as they interact during the simulation process. In this way, **CIAO** enables the visual inspection of the emerging patterns of agents' self-organisation, in response to changes in the simulation parameters, which can provide useful insights regarding the adaptability of the algorithm to the hidden properties of the problem.

**CIAO** v1.25 has been released under GNU General Public License (GPLv3); it is available online at:

[https://modelingcommons.org/browse/one\\_model/7368](https://modelingcommons.org/browse/one_model/7368)



# Contents

Overview	iii
<b>1 Description of the tool</b>	<b>1</b>
1.1 What is CIAO?	1
1.2 How it works	2
1.3 How to use it	2
1.4 Other distinctive features	5
1.5 Try it yourself	6
1.6 Extending the tool	6
1.7 List of benchmark problems	7
<b>2 Installation and execution</b>	<b>9</b>
2.1 Online (web) version	9
2.2 Desktop version	11
<b>3 Source code</b>	<b>13</b>
<b>4 Software license</b>	<b>25</b>

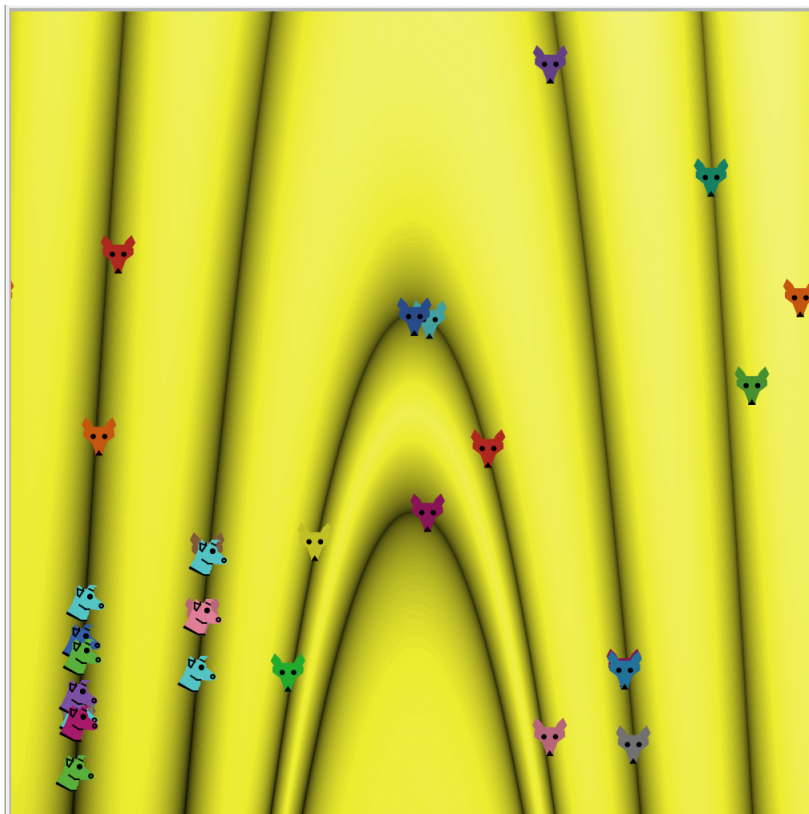


# Chapter 1

## Description of the tool

### 1.1 What is CIAO?

The **CIAO** (Collective Intelligence Algorithm for Optimisation) metaheuristic simulates a collective intelligence approach to solving unconstrained continuous optimisation problems. It involves agents known as solvers (wolves) and users (dogs), navigating a solution space to find the optimal coordinates that minimise a cost function associated with an optimisation problem. This tool implements the algorithm as an agent-based model using the Netlogo language.



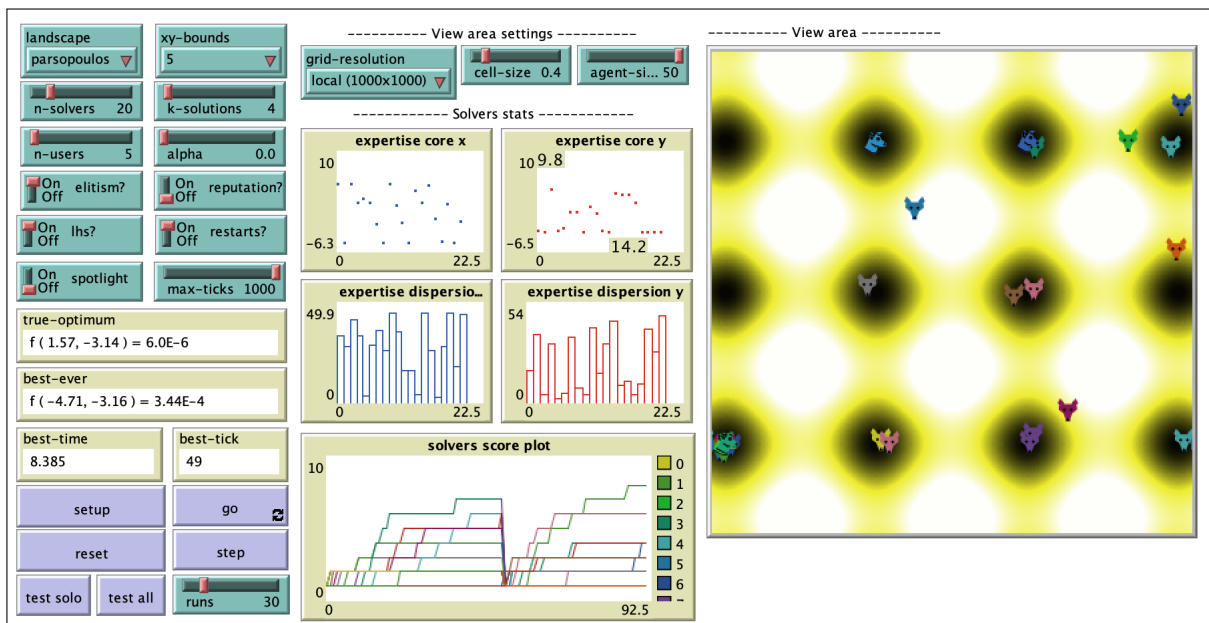
## 1.2 How it works

Solver agents maintain knowledge about promising sub-regions in the search space, represented as Gaussian distributions, involving their core expertise and their expertise dispersion. Users seek solutions from solvers, and the model incorporates learning and reputation mechanisms to refine the solver's expertise and reward effective solutions.

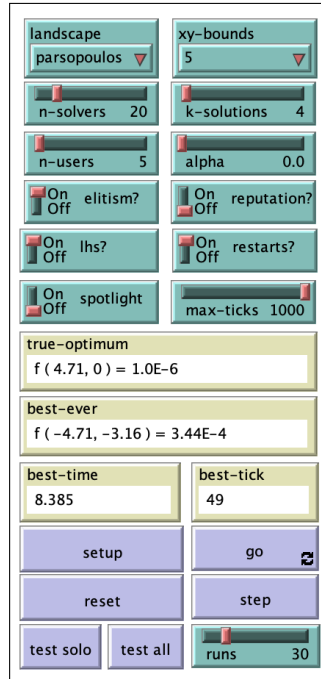
## 1.3 How to use it

Firstly, configure the simulation parameters in the simulation user interface:

- **LANDSCAPE:** Chooses the optimisation problem, visually represented in the view or world area. Selection influences the XY-BOUNDS. Refer to the APPENDIX: LIST OF BENCHMARK PROBLEMS section for a description of available benchmark functions.
- **XY-BOUNDS:** Sets the lower and upper bounds of the search space depending on the chosen landscape.
- **N-SOLVERS:** Defines the number of solver agents.







- **K-SOLUTIONS:** Defines the number of solutions a chosen solver attempts to generate.
- **N-USERS:** Defines the number of user agents.
- **ALPHA:** Sets the learning rate for solver adaptation.
- **ELITISM?** Activates the elitism mechanism, which ensures that the best solution found in the current generation is passed on as the center of expertise for one of the solvers in the next generation of the algorithm.
- **REPUTATION?:** Enables or disables choosing solvers based on their scores using roulette wheel selection.
- **LHS?:** Enables or disables Latin Hypercube Sampling of the initial solver population.
- **RESTARTS?:** Enables or disables random resets to prevent stagnation due to premature convergence to local minima.
- **SPOTLIGHT?:** Enables or disables highlighting the global minima in the view or world area.
- **MAX-TICKS:** Sets the maximum number of iterations of the algorithm main search routine.
- **GRID-SIZE:** Adjusts the resolution of the view area. Choose “*web (200x200)*” if running on the model online in the modelling commons website, as server memory constraints limit the amount of cells in the search space. Choose “*local (1000x1000)*” for a higher resolution of 1000x1000 cells if running on a desktop machine, allowing for better discretisation of the search space.

- **CELL-SIZE:** Specifies the size of each grid cell in the view area. Can be adjusted from 0.1 to 2 with a step increment of 0.1. This control enables closer or further inspection of the cells in the view area.
- **AGENT-SIZE:** Controls the size of agent representations in the view area. Adjusts from 10 to 50 with a step increment of 10.

A typical configuration of values for these parameters would be:

- N-SOLVERS: 10
- K-SOLUTIONS: 4
- N-USERS: 5
- ALPHA: 0.5
- ELITISM?: On
- REPUTATION?: On
- LHS?: On
- RESTARTS?: On
- MAX-TICKS: 1000

Next click the **SETUP** button to initialise the model with chosen parameters. And then click the **GO** button to start the simulation. Observe the movement and interaction of solvers and users in the view area of the simulator. You can control the execution of the simulation using the control panel buttons:

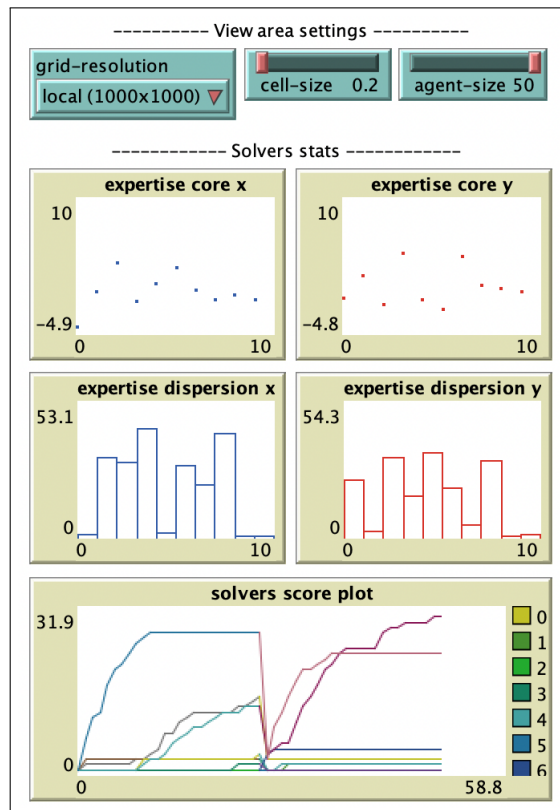
- **SETUP:** Computes and visualises the landscape and initialises agents and global variables of the simulation, according to the given parameters.
- **RESET:** Only initialises agents and global variables of the simulation, according to the given parameters. Also clears plots from previous runs.
- **GO:** Executes the main search routine until stopping conditions are met.
- **STEP:** Executes one iteration of the main search routine.

The model also features two buttons for test experimentation:

- **TEST SOLO:** Executes an experiment with the specified number of repetitions (i.e. RUNS), using the current configuration of model parameters for the selected LANDSCAPE. Results of each run are recorded as either a *hit* or a *miss* depending on whether the algorithm finds the optimum or not. The BEST-TICK, which indicates the time step at which the solution was found in each run, along with the overall success rate within the specified MAX-TICKS, is displayed in the COMMAND-CENTER panel.
- **TEST ALL:** Performs the same experiment but tests the specified number of RUNS on all benchmarks available in the LANDSCAPE list. Aggregated results showing the success rate for each experiment are displayed in the COMMAND-CENTER panel.

## 1.4 Other distinctive features

- Observe how solvers adapt their expertise to the search space, guiding users towards promising regions.
- Observe how regions with low (black) and high (yellow) values in the landscape are visualised in the view area based on the selected problem (LANDSCAPE).
- Watch the TRUE-OPTIMUM value and notice how the BEST-EVER approaches to it.
- Monitor the BEST-EVER patch, BEST-TICK, and BEST-TIME to understand when and where the best solution is discovered.
- Observe how the EXPERTISE CORE and EXPERTISE DISPERSION parameters of solvers evolve over time in the corresponding plots.
- Analyse the SOLVERS SCORE plot to see how solvers' reputation change during the optimisation process.
- Notice the periodic changes in the SOLVERS SCORE plot when RESTARTS? is enabled.



## 1.5 Try it yourself

- Experiment with different numbers of SOLVERS and USERS to observe how the collective intelligence adapts to problem complexity.
- Observe how adjusting the learning rate (ALPHA) affects the adaptation of solver expertise. Higher values (closer to 1) make solvers more resistant to exploring new solutions and cling to their currently known best solution. Lower values make them more susceptible to learning from new information and exploring alternative solutions.
- Evaluate the impact of greedy (utilising the single best new solution) and non-greedy (leveraging the average performance of multiple new solutions) learning strategies on solver adaptation using the GREEDY? switch.
- Explore the effects on solvers scores and on user decisions, of enabling or disabling reputation-based solver selection (REPUTATION?).
- Test the impact of Latin Hypercube Sampling of initial solver locations (LHS?).
- Observe the behaviour when random restarts are enabled or disabled (RESTARTS?).
- Toggle the spotlight (SPOTLIGHT?) to visually track the global minimum in the landscape, and how user agents approach to it.

As a side note, we remark that the resolution level can induce quantisation errors during the cost function sampling, therefore the optimum patch coordinates of a given LANDSCAPE can differ depending on the GRID-SIZE. For example, the optimum of ROSENBROCK'S problem is different for 1000x1000 and 200x200 resolutions.

## 1.6 Extending the tool

Some possible paths for tool extensions are:

- Extend the list of landscape functions of optimisation problems.
- Implement additional user or solver behaviours to enhance the complexity of the collective intelligence dynamics. Techniques such as temporal memory, tabu lists, collaboration mechanisms, or more advanced expertise representation models like a mixture of Gaussians.
- Extend the model to incorporate alternative solver selection strategies to compare their impact on the optimisation process.
- Generalise the model to handle continuous optimisation problems in more than two dimensions ( $d > 2$ ).
- Investigate the adaptation of the model for binary domain problems, exploring how the dynamics change in this context.

## 1.7 List of benchmark problems

The **CIAO** tool offers a range of benchmark optimisation problems for users to explore the behaviour of the Collective Intelligence elements of the algorithm. Widely known in optimisation literature, these problems provide diverse challenges. Users can select from any of the 33 benchmark functions included in this version.

Here's a brief overview of the 33 benchmark problems available in the LANDSCAPE control, listed in alphabetical order:

1. **Ackley:** A smooth and well-known optimisation problem characterised by a large, deep, and nearly flat global minimum.
2. **Beale:** A multimodal problem with a relatively flat region around the global minimum.
3. **Bohachesvsky n.1:** A non-convex problem with multiple local minima and a single global minimum.
4. **Booth:** A simple, yet challenging optimisation problem with a single global minimum.
5. **Cross-in-Tray:** A problem with four symmetric global minima and an intricate landscape.
6. **Damavandi:** A complex multimodal problem with varying scales of minima.
7. **Dixon-Price:** A non-convex problem with multiple local minima.
8. **Dropwave:** A problem with a large, flat area around the global minimum, adding difficulty to optimisation.
9. **Easom:** A problem with a global minimum resembling the shape of a cosine function.
10. **Eggholder:** A challenging problem with multiple global minima, characterised by a complex landscape.
11. **Goldstein-Price:** A problem with a deep, narrow canyon leading to the global minimum.
12. **Himmelblau:** A multimodal problem with several local minima.
13. **Holder-Table:** A multimodal problem with a flat region around the global minimum.
14. **Hosaki:** A simple problem with a single global minimum and a smooth landscape.
15. **Levy:** A problem with a single global minimum and a complex, curved landscape.
16. **Matyas:** A convex problem with a single global minimum.

17. **Michalewicz:** A non-convex problem with multiple local minima.
18. **Mishra n.3:** A problem with multiple local minima and a single global minimum.
19. **Mishra n.5:** A non-convex problem with multiple local minima and a single global minimum.
20. **Mishra n.6:** A problem with several local minima and a single, more pronounced global minimum.
21. **Parsopoulos:** A complex multimodal problem with varying scales of minima.
22. **Random:** The cost function is randomly generated, simulating scenarios where optimisation landscapes lack deterministic behaviour or specific mathematical properties.
23. **Rastrigin:** A challenging optimisation problem with a highly multimodal landscape.
24. **Rastrigin Bipolar:** A bipolar version of the Rastrigin problem.
25. **Rastrigin Offset:** A variant of the Rastrigin problem with an offset in the global minimum.
26. **Rosenbrock:** A classic optimisation problem with a valley leading to the global minimum.
27. **Schaffer n.2:** A simple, yet challenging optimisation problem with a deep and narrow global minimum.
28. **Schaffer n.4:** A problem with a large and flat global minimum.
29. **Sphere:** A convex problem with a single global minimum.
30. **Sphere-Offset:** A variant of the Sphere problem with an offset in the global minimum.
31. **Three-Hump Camel:** A non-convex problem with multiple local minima.
32. **Vincent:** A problem with several local minima and a single, more pronounced global minimum.
33. **Zakharov:** A problem with a large and flat global minimum.

We encourage you to explore various benchmark functions and witness how the **CIAO** model adapts to diverse optimisation challenges, showcasing the power of Collective Intelligence in solving engineering problems.

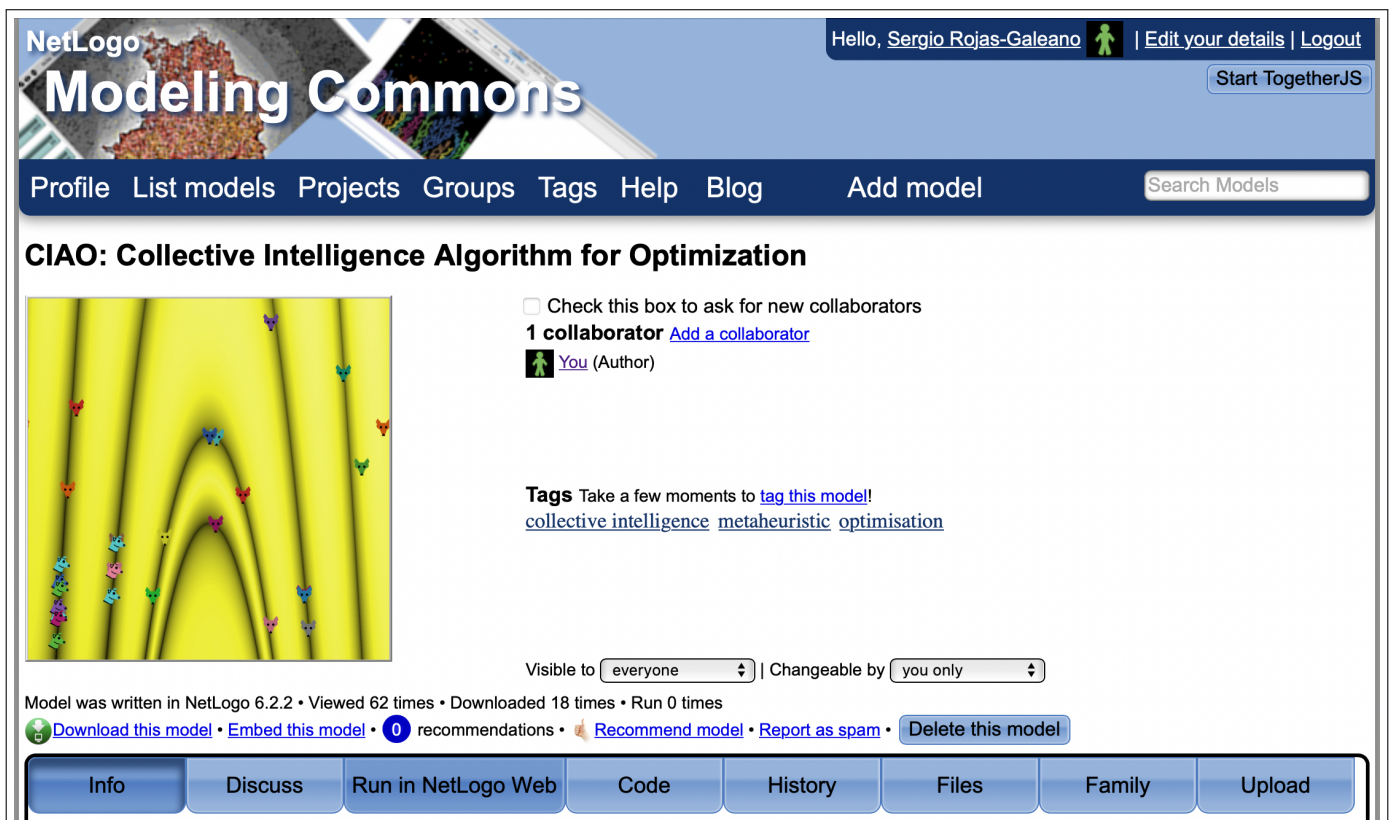
# Chapter 2

## Installation and execution

### 2.1 Online (web) version

The easiest way of experimenting with **CIAO** 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/7368](http://modelingcommons.org/browse/one_model/7368)
2. The following web page should appear:



The screenshot shows the NetLogo Modeling Commons website interface. At the top, there is a navigation bar with the site name and user information. Below that is a secondary navigation bar with links to various sections. The main content area displays the title of the model, a thumbnail image, and details about collaborators and tags. At the bottom, there are buttons for interacting with the model.

NetLogo  
**Modeling Commons**

Hello, Sergio Rojas-Galeano | Edit your details | Logout  
Start TogetherJS

Profile List models Projects Groups Tags Help Blog Add model Search Models

### CIAO: Collective Intelligence Algorithm for Optimization

Check this box to ask for new collaborators  
**1 collaborator** [Add a collaborator](#)  
You (Author)

**Tags** Take a few moments to [tag this model!](#)  
[collective intelligence](#) [metaheuristic](#) [optimisation](#)

Visible to: everyone | Changeable by: you only

Model was written in NetLogo 6.2.2 • Viewed 62 times • Downloaded 18 times • Run 0 times  
[Download this model](#) • [Embed this model](#) • 0 recommendations • [Recommend model](#) • [Report as spam](#) • [Delete this model](#)

Info Discuss Run in NetLogo Web Code History Files Family Upload

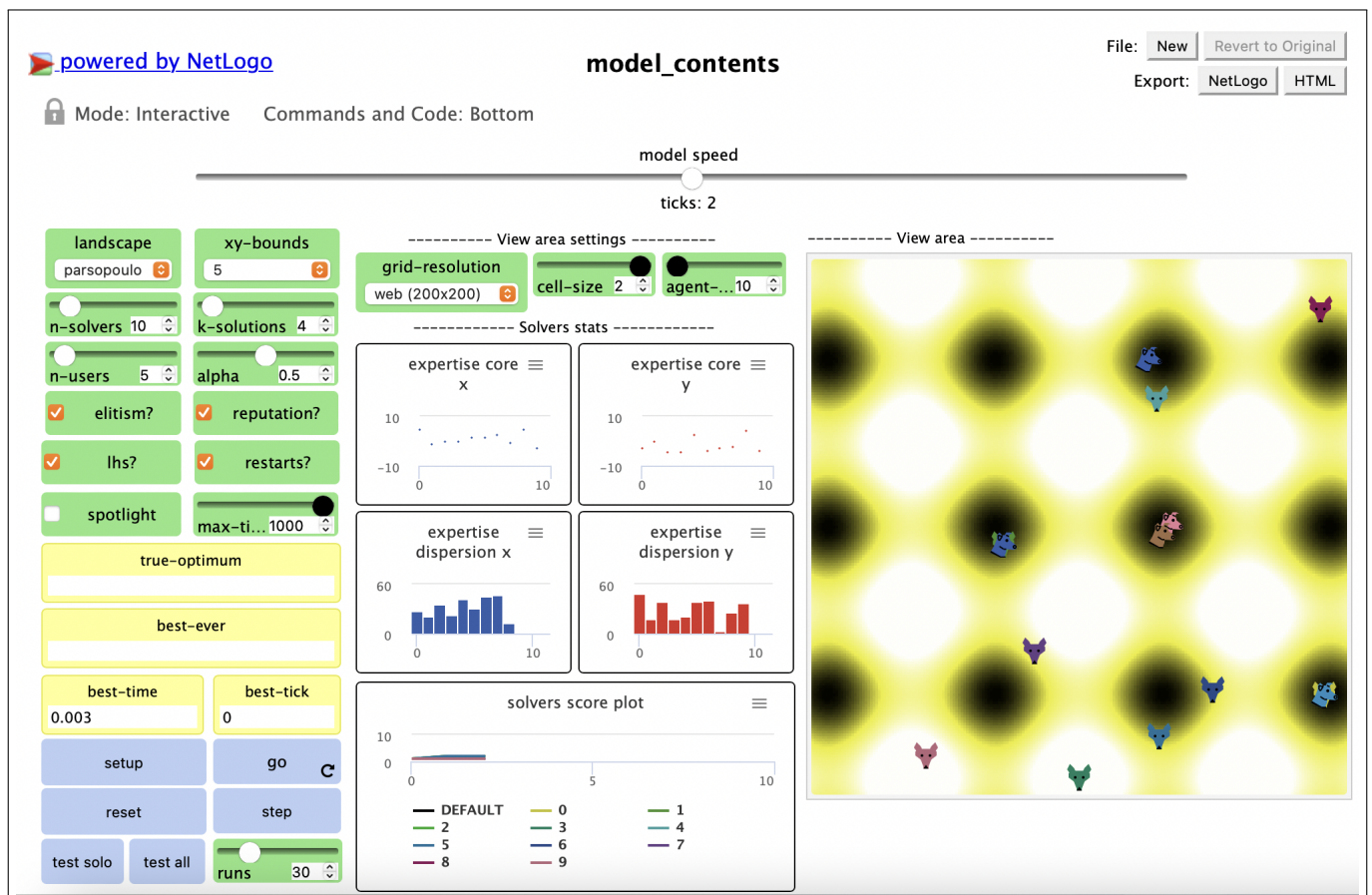
3. From the toolbar, choose the “Run in NetLogo Web” tab:



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:



The screenshot shows the NetLogo Web interface for the 'model\_contents' model. The interface is divided into several sections:

- Top Bar:** Includes 'powered by NetLogo', 'model\_contents', and file management options (File: New, Revert to Original; Export: NetLogo, HTML).
- Control Panel (Left):** Contains various sliders and buttons for model configuration, including 'landscape' (parsopoulo), 'xy-bounds' (5), 'grid-resolution' (web (200x200)), 'cell-size' (2), 'agent-...' (10), 'n-solvers' (10), 'k-solutions' (4), 'n-users' (5), 'alpha' (0.5), 'elitism?' (checked), 'reputation?' (checked), 'lhs?' (checked), 'restarts?' (checked), 'spotlight', 'max-ti...' (1000), 'true-optimum', 'best-ever', 'best-time' (0.003), 'best-tick' (0), 'setup', 'go', 'reset', 'step', 'test solo', 'test all', and 'runs' (30).
- Monitors and Plots (Center):** Displays 'model speed' (ticks: 2), 'View area settings' (grid-resolution, cell-size, agent-...), 'Solvers stats' (expertise core x, expertise core y, expertise dispersion x, expertise dispersion y), and 'solvers score plot'.
- View Area (Right):** Shows a simulation of agents (represented by small icons) on a landscape (represented by a grid of yellow and black cells).

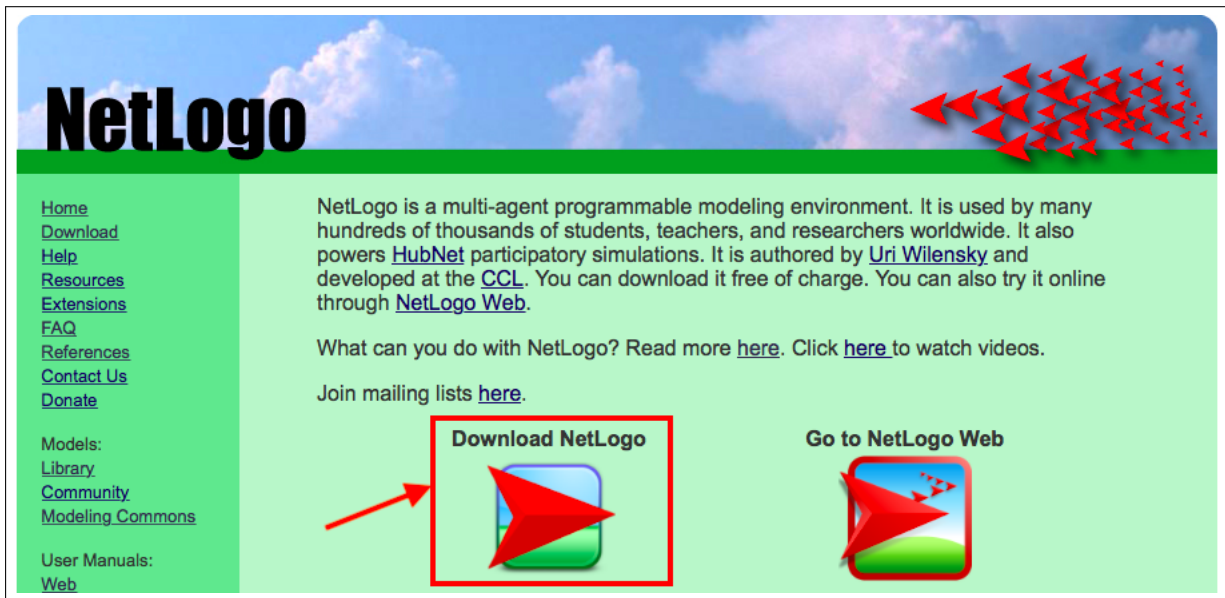
6. That’s it! Select the running parameters in the control panel, click on SETUP, and then on GO! You’ll observe the emergence of collective intelligence solving optimisation problems in the simulation view area. Performance indicators of the CIAO algorithm will be displayed in the monitors and plots. Alternatively, you can run the simulation multiple times using the QUICK TEST button and RUNS slider.



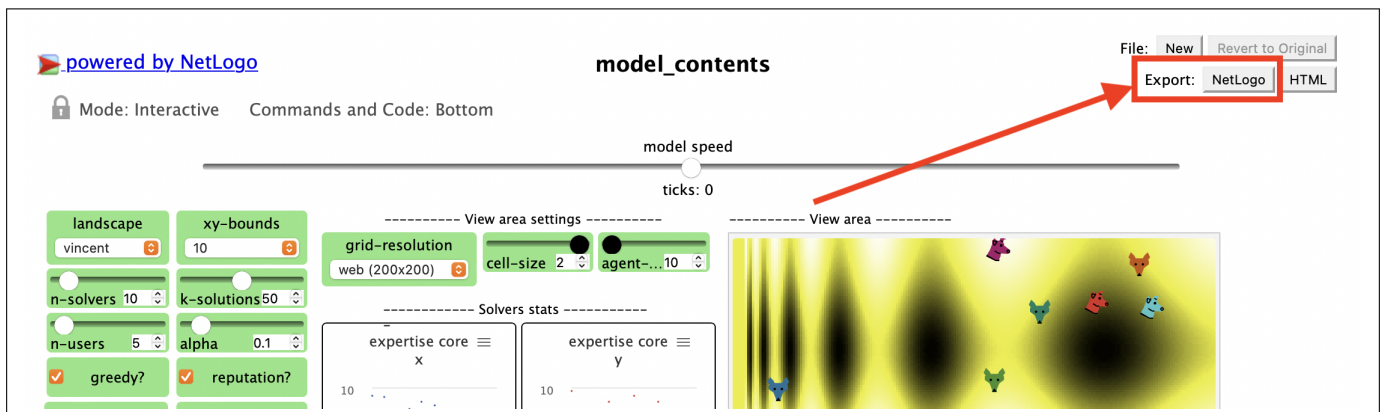
## 2.2 Desktop version

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 resolution for the view area. For this purpose, **CIAO** runs over the NetLogo desktop simulation platform. 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 installation instructions:

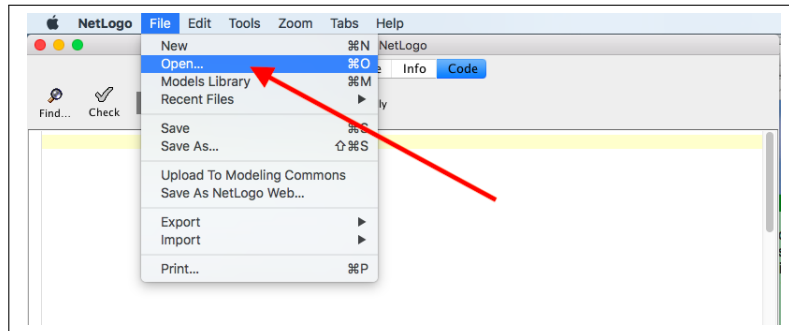


2. Download the **CIAO** model file from the model webpage, using the “Export” button:



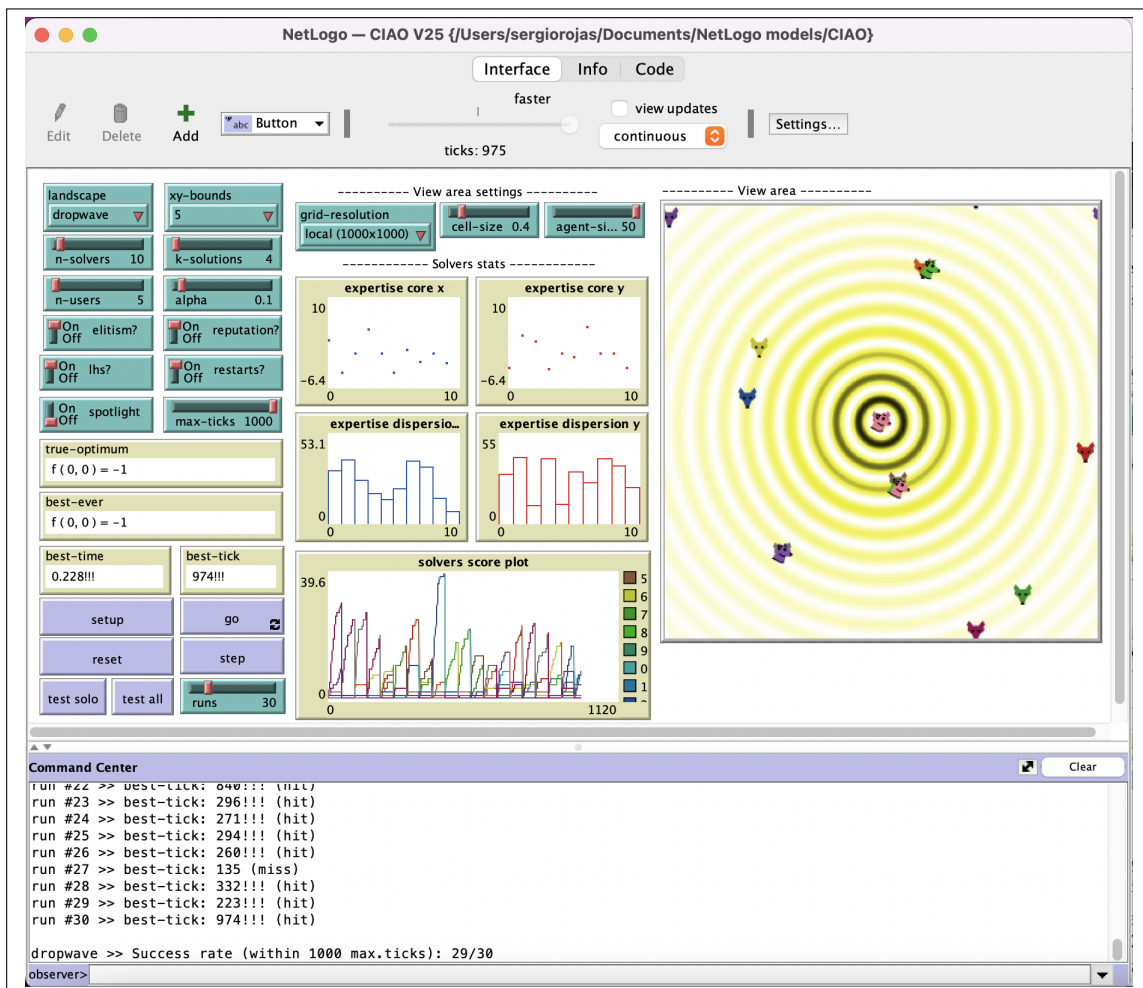
A file named *model\_contents.nlogo* (or the name you specify) will be downloaded to your local drive.

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



Locate the file named *model\_contents.nlogo* (or the name you specified) that you downloaded previously and open it.

4. The CIAO desktop screen will show up:



That's it! Choose the running parameters in the control panel, click on SETUP, and then on GO! You'll observe the emergence of collective intelligence solving optimisation problems in the simulation view area. Performance indicators of the CIAO algorithm will be displayed in the monitors and plots. Alternatively, you can run the simulation multiple times using the QUICK TEST button and RUNS slider.

# Chapter 3

## Source code

```
-----  
; CIAO: Collective Intelligence Algorithm for Optimization  
-----  
; This NetLogo code implements CIAO, a novel metaheuristic  
; algorithm that leverages the collective intelligence of  
; solvers and users to explore and optimize the solution  
; space of a given optimization problem.  
;  
; Authors: Sergio Rojas-Galeano, Lindsay Álvarez, Martha Garzón  
; v1.25 Copyright (c) 2024 The authors  
; License: GNU General Public License (GPLv3)  
; See Info tab for full copyright and license.  
-----  
  
; Global variables to store information about the best solution found  
globals [  
  true-best-patch ; Patch corresponding to the ground-truth optimum  
  best-ever       ; Patch corresponding to the best solution found so far  
  best-tick       ; Tick when best-ever was found  
  best-time       ; Time when best-ever was found  
]  
  
; Patch attributes to represent landscape and cost function of the optimization  
  problem  
patches-own [  
  x ; Patch x-coordinate, associated with a point in the solution space  
    within the defined bounds  
  y ; Patch y-coordinate, associated with a point in the solution space  
    within the defined bounds  
  value ; Patch value corresponding to the cost function evaluated at its (x, y)  
    coordinates  
]  
  
; Agent breeds for different types of agents in the simulation  
breed [solvers solver]  
breed [users user]  
breed [solutions solution]  
  
; Solver agents' attributes  
solvers-own [  
  bx by ; Knowledge of the best solution location found so far  
  mx my ; Core (mu) component of solver expertise  
  sx sy ; Breadth (sigma) component of solver expertise  
  score ; Solver's rating given by users  
]
```

```

; User agents' attributes
users-own [
  own-best ; User's best solution value found so far
]

; Execute a single iteration of the main loop.
to go
  ; Reset the timer on the first tick
  if ticks = 0 [ reset-timer ]

  ; Apply search operators
  search-solutions
  update-best
  tick

  ; Restart agents at regular intervals if restarts are enabled
  if (restarts? and ticks mod 50 = 0) [ restart ]

  ; Stopping conditions: maximum number of ticks reached or true-best-patch found
  if (ticks > max-ticks) or (any? true-best-patch with [value = [value] of
    best-ever]) [ stop ]
end

; Search for solutions based on user interactions
to search-solutions
  ask users [

    ; Choose a solver and spawn new candidate solutions according to its
    expertise model
    let my-solver choose-solver
    hatch-solutions k-solutions [
      set xcor random-normal [mx] of my-solver [sx] of my-solver
      set ycor random-normal [my] of my-solver [sy] of my-solver
    ]

    ; Save user current location and move it to the best among the new solutions
    let old-patch patch-here
    move-to min-one-of solutions [value]

    ; Check if the best among the new solutions is better than the user's own best
    ifelse value < own-best [

      ; If so, update the user's own best and reward the chosen solver using the
      new location
      set own-best value
      reward-solver my-solver patch-here

    ] [

      ; Otherwise, preserve the previous own best location
      move-to old-patch
    ]

    ; Update solver's expertise and forget candidate solutions
    update-solver my-solver
    ask solutions [ die ]

    ; Apply elitism if switched on
    if elitism? [

      ; Choose the best performing solver and center it around the best ever
      location
      ask min-one-of solvers [ value ] [
        move-to best-ever
        set mx xcor set my ycor
      ]
    ]
  ]
]

```

```

]
end

; Reward the solver for obtaining a better solution
to reward-solver [the-solver better-patch]
  ask the-solver [
    set score score + 1 ; Increase the solver's reputation
    move-to better-patch ; Move to the better solution's location
    set bx xcor ; Update the solver's best solution's x-coordinate
    set by ycor ; Update the solver's best solution's y-coordinate
  ]
end

; Update the solver's knowledge as they search for new solutions
to update-solver [the-solver]
  ask the-solver [
    let new-x 0 let new-y 0 ; Initialize variables for new coordinates

    set new-x [xcor] of min-one-of solutions [value]
    set new-y [ycor] of min-one-of solutions [value]

    ; Update solver's core expertise based on the learning rate
    set mx (alpha * bx) + ((1 - alpha) * new-x)
    set my (alpha * by) + ((1 - alpha) * new-y)
    setxy mx my ; Move the solver to the updated location

    ; Narrow down the solver's expertise dispersion
    set sx sx * exp (-.001 * (ticks mod 50))
    set sy sy * exp (-.001 * (ticks mod 50))
  ]
end

; Check if any user has improved the best solution found so far
to update-best
  let best-now min-one-of users [value]

  if [value] of best-now < [value] of best-ever [
    ; Record the location, tick, and time when a newer best-ever solution was
    found.
    ask best-now [ set best-ever patch-here ]
    set best-tick ticks
    set best-time timer
  ]
end

; Perform the initial setup for the simulation
to setup
  clear-all ; Clear the world and all agents
  setup-search-landscape ; Set up the landscape and cost function
  setup-solvers ; Set up solver agents
  setup-users ; Set up user agents
  setup-globals ; Set up global variables
end

; Reset the simulation for a new run (without computing the landscape again)
to reset
  clear-all-plots ; Clear plot monitors from the previous run
  setup-solvers ; Reinitialize solver agents
  setup-users ; Reinitialize user agents
  setup-globals ; Reinitialize global variables
end

; Set up solver agents
to setup-solvers

```

```

; Clear previous solver agents
ask solvers [ die ]

; Create solver agents
create-solvers n-solvers [

; Assign visual attributes
set shape "wolf_7"
set color 4 + (10 * who)
set size agent-size

; Initialize expertise model and score
set mx random-xcor
set my random-ycor
set sx random-float 50
set sy random-float 50
set bx mx
set by my
set score 1

; Set initial random location
setxy mx my
]

; If enabled, change locations with Latin Hypercube Sampling (lhs)
if lhs? [ latin-hypercube-sampling ]
end

; Set up user agents
to setup-users
; Clear previous user agents
ask users [ die ]

; Create new users
create-users n-users [
; Assign visual attributes
set shape "dog"
set size agent-size

; Assign initial own best solution
move-to one-of patches
set own-best value
]
end

; Assign initial values for global variables
to setup-globals
set best-tick 0
set best-time 0
set best-ever max-one-of patches [value]
reset-ticks
end

; Restart agents to prevent stagnation
to restart
setup-users
setup-solvers
end

; Choose a solver either randomly or guided by reputation
to-report choose-solver
ifelse (reputation?) [
; Choose a solver based on their reputation using a roulette wheel
let score-list map [ [the-solver] -> [score] of the-solver ] sort solvers
let id-list map [ [the-solver] -> [who] of the-solver ] sort solvers

```

```

; Compute cumulative distribution from the histogram of scores
let hist fput (list (first score-list)) (but-first score-list)
let aggs reduce [ [cumul next] -> sentence cumul ((last cumul) + next) ] hist

; Use a roulette wheel to choose a solver according to the cumulative
distribution
let pockets map [ p -> p / last aggs ] aggs ; Compute wheel pockets by
normalizing cumulative sum
let ball random-float 1 ; Roll the ball, then check the
winner pocket
let winner first filter [ [index] -> ball <= item index pockets ] range
length pockets
report solver item winner id-list
] [
; Otherwise choose any solver at random uniformly
report one-of solvers
]
end

; Perform Latin Hypercube Sampling for initial solver locations
to latin-hypercube-sampling
; Compute the width of location slots
let width 2 * max-pxcor / n-solvers

; Split each dimension into non-overlapping slots (1 per solver) and sample
random locations within
foreach range 2 [ index ->
let coordinates shuffle n-values n-solvers [ slot -> width * (slot +
random-float 1) ]

; Assign coordinate locations for each agent in an orderly manner, ensuring
bound constraints
(foreach sort solvers coordinates [
[the-solver coordinate] ->
ask the-solver [
ifelse index = 0 [
; x-coordinate
set mx (- max-pxcor + coordinate)
set bx mx
set xcor mx
][
; y-coordinate
set my (- max-pycor + coordinate)
set by my
set ycor my
]
]
])
end

; Reporter to display solver's score
to-report show-scores [the-agent]
report (word "s_" who "=" score "u|u")
end

; Reporter to display agent's location
to-report locations [the-agent]
report (word "u_" who ":u(" precision x 1 ",u" precision y 1 ")|")
end

; Define view area settings when loading the model
to startup
set grid-resolution "web_u(200x200)"

; Startup with cell size 1 to prevent view area distortions when resizing
(NetLogo bug)
set cell-size 1
set agent-size 10

```

```

end

; Routine for quick testing of success rate of a number of repetitions
to quick-test [verbose]
  print "-----"
  type (word landscape "␣>>␣Testing␣" runs "␣runs" ifelse-value verbose ["␣
    (successful␣runs␣marked␣as␣'!!!'):␣\n"] [":"])
  setup
  set max-ticks round ( (ifelse-value grid-resolution = "web␣(200x200)" [ 8000 ]
    [ 20000 ]) / (n-users * k-solutions))

  let n 0 let i 1
  repeat runs [
    ifelse landscape = "random" [ setup ] [ reset ]
    let success false
    while [ ticks < max-ticks and not success] [
      go
      set success (any? true-best-patch with [value = [value] of best-ever])
    ]
    ifelse verbose [
      print (word "run␣#" i "␣>>␣best-tick:␣" best-tick ifelse-value success [
        "!!!␣(hit)" ] [ "␣(miss)" ])
    ] [
      type "."
    ]
    set n n + ifelse-value success [ 1 ] [ 0 ]
    set i i + 1
  ]
  print (word "\n" landscape "␣>>␣Success␣rate␣(within␣" max-ticks "␣max.ticks):␣
    " n "/" runs)
end

;----- DEFINITION OF BENCHMARK OPTIMIZATION PROBLEM LANDSCAPES
;-----
; This procedure computes the landscape of the chosen cost function and
; visualizes it.
; Source: Jamil, M., & Yang, X. S. (2013). A literature survey of benchmark
; functions for global
; optimization problems. International Journal of Mathematical Modelling and
; Numerical Optimisation.
; To add new problems, simply insert the mathematical expression of their cost
; function as new cases.

to setup-search-landscape
  clear-all

  ; Setup world and patch size
  set-patch-size cell-size
  ifelse grid-resolution = "web␣(200x200)" [
    (ifelse
      landscape = "damavandi" or
      landscape = "hosaki" or
      landscape = "michalewicz" or
      landscape = "vincent" [ resize-world 0 200 0 200 ]
      landscape = "zakharov" [ resize-world -50 150 -50 150 ]
      ; All other optimisation problems are defined over the four quadrants of
      the solution space
      [ resize-world -100 100 -100 100 ]
    )
  ] [
    (ifelse
      landscape = "damavandi" or
      landscape = "hosaki" or
      landscape = "michalewicz" or
      landscape = "vincent" [ resize-world 0 1000 0 1000 ]
      landscape = "zakharov" [ resize-world -250 750 -250 750 ]
      ; All other optimisation problems are defined over the four quadrants of
      the solution space
      [ resize-world -500 500 -500 500 ]
    )
  ]

```



```

)
]

; Setup range of variable coordinates for each problem
set xy-bounds (ifelse-value
  landscape = "ackley" [ 32 ]
  landscape = "beale" or
  landscape = "michalewicz" or
  landscape = "parsopoulos" [ 5 ]
  landscape = "bohachesvsky_n.1" [ 100 ]
  landscape = "booth" or
  landscape = "cross-in-tray" or
  landscape = "dixon-price" or
  landscape = "holder-table" or
  landscape = "hosaki" or
  landscape = "levy" or
  landscape = "matyas" or
  landscape = "mishra_n.3" or
  landscape = "mishra_n.5" or
  landscape = "mishra_n.6" or
  landscape = "vincent" or
  landscape = "zakharov" [ 10 ]
  landscape = "damavandi" [ 14 ]
  landscape = "eggholder" [ 512 ]
  landscape = "goldstein-price" [ 2 ]
; For the following problems, we use the range [-32, 32] instead of the
  original [-100, 100], to reduce discretization errors
  landscape = "easom" or
  landscape = "schaffer_n.4" or
  landscape = "schaffer_n.2" [ 32 ]
; For any other problem use a [-5, 5] default range
[ 5 ]
)

; Evaluate the cost function for each patch in the landscape
ask patches [
  set x pxcor * (xy-bounds / max-pxcor )
  set y pycor * (xy-bounds / max-pycor )

; Note: Trigonometric functions require input in degrees, not radians; thus,
  a conversion factor (180 / pi) was used
set value
(ifelse-value
  landscape = "ackley" [
    -20 * exp(-0.2 * sqrt(0.5 * (x ^ 2 + y ^ 2))) - exp(0.5 * (cos((180 / pi)
      * (2 * pi) * x) + cos((180 / pi) * (2 * pi) * y))) + 20 + e
  ]
  landscape = "beale" [
    ((1.5 - x + (x * y)) ^ 2) + ((2.25 - x + (x * (y ^ 2))) ^ 2) + ((2.625 -
      x + (x * (y ^ 3))) ^ 2)
  ]
  landscape = "bohachesvsky_n.1" [
    (x ^ 2) + 2 * (y ^ 2) - (0.3 * (cos((180 / pi) * 3 * pi * x))) - (0.4 *
      cos ((180 / pi) * 4 * pi * y)) + 0.7
  ]
  landscape = "booth" [
    (x + (2 * y) - 7) ^ 2 + ((2 * x) + y - 5) ^ 2
  ]
  landscape = "cross-in-tray" [
    -0.0001 * (((abs(sin((180 / pi) * x) * sin((180 / pi) * y) * exp(abs(100
      - ((sqrt((x ^ 2) + (y ^ 2)))) / pi)))) + 1) ^ 0.1)
  ]
  landscape = "damavandi" [
;
  ifelse-value ( x = 2 ) or ( y = 2 ) [ 100 ]
;
  ifelse-value ( x = 2 ) and ( y = 2 ) [ 0 ]
  ifelse-value ( abs(x - 2) < 0.0001 ) or ( abs(y - 2) < 0.0001 ) [ 100 ]
  ifelse-value ( abs(x - 2) < 0.0001 ) and ( abs(y - 2) < 0.0001 ) [ 0 ]
  [(1 - (abs(((sin((180 / pi) * pi * (x - 2))) * (sin((180 / pi) * pi * (y
    - 2)))) / ((pi ^ 2) * (x - 2) * (y - 2)))) ^ 5 ) * ((2 + (x - 7)
    ^ 2) + (2 * (y - 7) ^ 2))]

```

```

]
landscape = "dixon-price" [
  (x - 1) ^ 2 + 2 * ((2 * y ^ 2) - x) ^ 2
]
landscape = "dropwave" [
  -1 * (((1 + (cos((180 / pi) * 12 * sqrt((x ^ 2) + (y ^ 2)))))) / (0.5 *
    ((x ^ 2) + (y ^ 2) + 2)))
]
landscape = "easom" [
  -1 * (cos((180 / pi) * x) * cos((180 / pi) * y)) * exp(-((x - pi) ^ 2
    + (y - pi) ^ 2))
]
landscape = "eggholder" [ ; note that degrees, not radians, are needed for
  sin function
  ((- x) * sin((180 / pi) * sqrt(abs(x - (y + 47)))))) - (y + 47) * sin
    ((180 / pi) * sqrt(abs((x / 2) + (y + 47))))
]
landscape = "goldstein-price" [
  (1 + ((x + y + 1) ^ 2) * (19 - (14 * x) + (3 * (x ^ 2) - (14 * y) + (6 *
    x * y) + (3 * (y ^ 2)))) *
    (30 + (((2 * x) - (3 * y)) ^ 2) * (18 - (32 * x) + (12 * (x ^ 2) + (48
    * y) - (36 * x * y) + (27 * (y ^ 2))))))
]
landscape = "himmelblau" [
  ((x ^ 2) + y - 11) ^ 2 + (x + (y ^ 2) - 7) ^ 2
]
landscape = "holder-table" [
  -1 * abs(sin((180 / pi) * x) * cos((180 / pi) * y) * exp(abs(1 - (sqrt(x
    ^ 2 + y ^ 2) / pi))))
]
landscape = "hosaki" [
  (1 - (8 * x) + (7 * (x ^ 2)) - ((7 / 3) * x ^ 3) + (0.25 * (x ^ 4))) *
    ((y ^ 2) * exp((- y)))
]
landscape = "levy" [
  (sin((180 / pi) * pi * (1 + (x - 1) / 4)) ^ 2)
  + (((1 + (x - 1) / 4) - 1) ^ 2) * (1 + 10 * (sin((180 / pi) * (pi * (1 +
    (x - 1) / 4) + 1)) ^ 2)
  + (((1 + (y - 1) / 4) - 1) ^ 2) * (1 + (sin((180 / pi) * (2 * pi * (1 +
    (y - 1) / 4)))) ^ 2)
]
landscape = "matyas" [
  (0.26 * ((x ^ 2) + (y ^ 2))) - (0.48 * (x * y))
]
landscape = "michalewicz" [
  -1 * (sin((180 / pi) * x) * (sin((180 / pi) * 1 * (x ^ 2) / pi)) ^ (2 *
    10))
  - (sin((180 / pi) * y) * (sin((180 / pi) * 2 * (y ^ 2) / pi)) ^ (2 *
    10))
]
landscape = "mishra_n.3" [
  sqrt(abs(cos((180 / pi) * sqrt(abs((x ^ 2) + y)))))) + 0.01 * (x + y)
]
landscape = "mishra_n.5" [
  (((sin((180 / pi) * (cos((180 / pi) * x) + cos((180 / pi) * y)) ^ 2)) ^
    2
  + (cos((180 / pi) * (sin((180 / pi) * x) + sin((180 / pi) * y)) ^ 2))
    ^ 2 + x) ^ 2)
  + (.01 * x) + (.1 * y)
]
landscape = "mishra_n.6" [
  -1 * ln(((sin((180 / pi) * (cos((180 / pi) * x) + cos((180 / pi) * y)) ^
    2)) ^ 2
  - (cos((180 / pi) * (sin((180 / pi) * x) + sin((180 / pi) * y)) ^ 2))
    ^ 2 + x) ^ 2)
  + .1 * ((x - 1) ^ 2 + (y - 1) ^ 2)
]
landscape = "parseopoulos" [
  cos((180 / pi) * x) ^ 2 + sin((180 / pi) * y) ^ 2
]
]

```

```

landscape = "rastrigin" [
  20 + ((x ^ 2) - 10 * cos((180 / pi) * (2 * pi) * x)) + ((y ^ 2) - 10 *
    cos((180 / pi) * (2 * pi) * y))
]
landscape = "rastrigin_offset" [
  20 + (((x - 1.123) ^ 2) - 10 * cos((180 / pi) * (2 * pi) * (x - 1.123)))
    + (((y - 1.123) ^ 2) - 10 * cos((180 / pi) * (2 * pi) * (y - 1.123)))
]
landscape = "rastrigin_bipolar" [
  20 + (((x + 1) ^ 2) - 10 * cos((180 / pi) * (2 * pi) * (x + 1))) + (((y
    - 1) ^ 2) - 10 * cos((180 / pi) * (2 * pi) * (y - 1)))
]
landscape = "rosenbrock" [
  100 * (y - (x ^ 2)) ^ 2 + (1 - x) ^ 2
]
landscape = "schaffer_n.2" [
  0.5 + (((sin((180 / pi) * (x ^ 2 - y ^ 2)) ^ 2) - 0.5) / (1 + (0.001 * (x
    ^ 2 + y ^ 2))) ^ 2)
]
landscape = "schaffer_n.4" [
  0.5 + (((cos((180 / pi) * sin((180 / pi) * abs(x ^ 2 - y ^ 2)))) ^ 2 -
    0.5) / (1 + (0.001 * (x ^ 2 + y ^ 2))) ^ 2)
]
landscape = "sphere" [
  x ^ 2 + y ^ 2
]
  landscape = "sphere-offset" [
    (x - 3) ^ 2 + (y + 3) ^ 2
  ]
landscape = "three-hump_camel" [
  (2 * (x ^ 2)) - (1.05 * (x ^ 4)) + ((x ^ 6) / 6) + (x * y) + (y ^ 2)
]
landscape = "vincent" [
  ifelse-value x < 0.25 or y < 0.25 [ 0 ]
; [-1 * sin((180 / pi) * 10 * (log(x) 10)) - sin((180 / pi) * 10 *
(log(y) 10))]
  [-1 * sin((180 / pi) * 10 * ln(x)) - sin((180 / pi) * 10 * ln(y)) ]
]
landscape = "zakharov" [
  (x ^ 2 + y ^ 2) + ((0.5 * x) + (0.5 * 2 * y)) ^ 2 + ((0.5 * x) + (0.5 * 2
    * y)) ^ 4
]
; Otherwise, use a random landscape
[ random-normal 0 500 ]
)
]

; Smooth out random landscape for better visualization and search efficiency
if landscape = "random" [
  ask min-n-of 4 patches [value] [ ask patches in-radius 30 [ set value value -
    random-float 300 ] ]
  repeat 10 [ diffuse value 1 ]
]

; Find the true best patch (global minima) based on the chosen landscape
(ifelse
; Functions with 2 global minima
landscape = "dixon-price" [ set true-best-patch min-n-of 2 patches [value] ]

; Functions with 4 global minima
landscape = "cross-in-tray" or
landscape = "holder-table" or
landscape = "schaffer_n.4" [ set true-best-patch min-n-of 4 patches [value] ]

; Himmelblau has 4 global minima, but 5 emerge due to discretization errors
landscape = "himmelblau" [ set true-best-patch min-n-of 5 patches [value] ]

; Functions with 12 global minima
landscape = "parsopoulos" [ set true-best-patch min-n-of 12 patches [value] ]

```

```

; Functions with 6^2 global minima
landscape = "vincent" [ set true-best-patch min-n-of 36 patches [value] ]

; All other cost functions have a single global minima
[ set true-best-patch patch-set min-one-of patches [value] ]
)

;; Scale patches color within min and max values limits for visualisation purposes
let min-val min [value] of patches
let max-val max [value] of patches
  ask patches [
    (ifelse
      ; Problems better visualised using linear color scale
      landscape = "ackley" or
      landscape = "bohachesvsky_n.1" or
      landscape = "cross-in-tray" or
      landscape = "damavandi" or
      landscape = "dropwave" or
      landscape = "schaffer_n.2" or
      landscape = "schaffer_n.4" or
      landscape = "vincent"
      [ set pcolor scale-color yellow value min-val max-val]

      ; Problems better visualised using square rooted color scale
      landscape = "easom" or
      landscape = "eggholder" or
      landscape = "holder-table" or
      landscape = "michalewicz" or
      landscape = "mishra_n.6" or
      landscape = "parsopoulos" or
      landscape = "zakharov"
      [ set pcolor scale-color yellow value min-val sqrt max-val ]

      ; Problems better visualised using logarithmic scale
      landscape = "beale" or
      landscape = "dixon-price" or
      landscape = "goldstein-price" or
      landscape = "booth" or
      landscape = "rosenbrock"
      [ set pcolor scale-color yellow value min-val log max-val 1.01 ]

      landscape = "rastrigin" or
      landscape = "rastrigin_offset" or
      landscape = "rastrigin_bipolar"
      [ set pcolor scale-color yellow value min-val log max-val 1.05 ]

      landscape = "himmelblau" or
      landscape = "levy" or
      landscape = "matyas" or
      landscape = "mishra_n.3" or
      landscape = "sphere" or
      landscape = "sphere-offset" or
      landscape = "three-hump_camel"
      [ set pcolor scale-color yellow value min-val log max-val 1.1 ]

      landscape = "hosaki" or
      landscape = "mishra_n.5"
      [ set pcolor scale-color yellow value min-val log max-val 10.1 ]

      ; For any other problem use a logarithmic default scale
      [ set pcolor scale-color yellow value min-val log abs (max-val + 0.001)
        1.05 ]
    )
  ]

;; Set spotlight on or off
if spotlight = true [ watch one-of true-best-patch ]
end

; Shorter setup procedure for some representative landscape problems

```

```

to setup-search-landscapes-short
  ; Set up world and patch size
  resize-world -500 500 -500 500
  set-patch-size cell-size
  display

  ; Create the 2D landscape according to the chosen cost function and bound
  constraints
  set xy-bounds ifelse-value landscape = "eggholder" [ 512 ] [ 6 ]
  ask patches [
    set x pxcor * (xy-bounds / max-pxcor)
    set y pycor * (xy-bounds / max-pycor)

    set value (ifelse-value
      landscape = "sphere" [
        ifelse-value (x = 5) or (y = 5)
          [ 100 ]
          [ x ^ 2 + y ^ 2 ]
      ]
      landscape = "sphere-offset" [
        (x - 300 * (xy-bounds / max-pxcor)) ^ 2 + (y + 300 * (xy-bounds /
          max-pxcor)) ^ 2
      ]
      landscape = "rastrigin" [ ; Note that degrees, not radians, are needed for
        the cos function
        20 + ((x ^ 2) - 10 * cos ((180 / pi) * (2 * pi) * x)) + ((y ^ 2) - 10 *
          cos ((180 / pi) * (2 * pi) * y))
      ]
      landscape = "rosenbrock" [
        100 * (y - (x ^ 2))^ 2 + (1 - x)^ 2
      ]
      landscape = "himmelblau" [
        ((x ^ 2) + y - 11) ^ 2 + (x + (y ^ 2) - 7)^ 2
      ]
      landscape = "eggholder" [ ; Note that degrees, not radians, are needed for
        the sin function
        ((- x) * sin ((180 / pi) * sqrt (abs (x - (y + 47)))))) - (y + 47) * sin
          ((180 / pi) * sqrt (abs ((x / 2) + (y + 47))))
      ]
      [ random-normal 0 500 ] ; The last case is a random landscape
    )
  ]

  if landscape = "random" [
    ; Smooth out the random landscape
    ask min-n-of 4 patches [value] [ ask patches in-radius 30 [ set value value -
      random-float 300 ] ]
    repeat 10 [ diffuse value 1 ]
  ]

  ; Find the true best location
  ifelse landscape = "himmelblau" [
    ; "himmelblau" exhibits 4 global minima (5 emerge due to discretisation
    errors)
    set true-best-patch min-n-of 5 patches [value]
  ] [
    ; All other cost functions have a single global minima
    set true-best-patch patch-set min-one-of patches [value]
  ]

  ; Scale patches color within value limits
  let min-val min [value] of patches
  let max-val max [value] of patches
  ask patches [ set pcolor scale-color yellow value min-val log abs max-val 1.05 ]

  ; Set spotlight if switched on
  if spotlight = true [ watch one-of true-best-patch ]
end

;;;;; END OF FILE ;;;;;;

```



# Chapter 4

## Software license

**CIAO** version 1.25

Copyright © 2024 Sergio Rojas-Galeano, Lindsay Álvarez, Martha Garzón.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, you can download it from:

<https://www.gnu.org/licenses/gpl-3.0.en.html>.