ABM and environmental policy

A mini-POSEIDON model in NetLogo

Nicolas Payette

Behave Summer School
Brescia, Italy
September 11, 2024

What's the plan?

- A little bit about the POSEIDON[1] fisheries ABM
- A lot about **building** a simple NetLogo version of it.
- A fair bit about **calibrating** the model and using it for **optimizing** a policy.

[1] **P**rocess-based **O**cean-system **S**imulator for **E**volving **I**ntegrated **D**omains and **O**perational **N**eeds. ... Yeah. I know.

Why model fisheries?

- It's big!
  - 96.4 million tonnes of fish caught in 2018
  - employing 59.51 million people
  - 17% of total animal protein consumed globally
- It's in trouble!
  - 34.2% of stocks are overfished

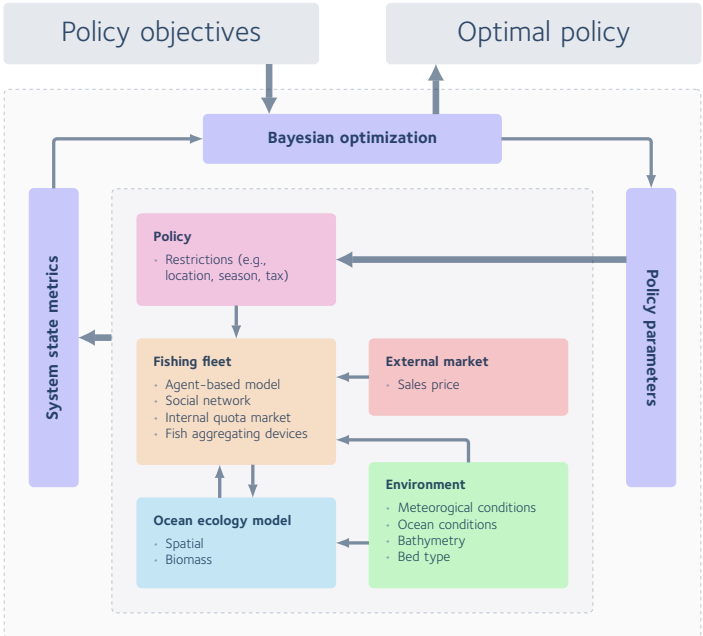
Why use ABM to do it?

- They're inherently spatial;
- they involve complex interactions,
- between smart, adaptive agents.

Source: `http://www.fao.org/3/ca9231en/CA9231EN.pdf`

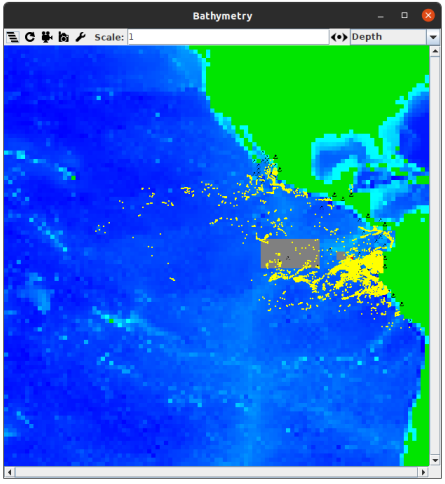
POSEIDON

Source: Ricardo André Frantz, `https://w.wiki/wzx`



POSEIDON applications

- Conceptual models
- US West Coast Groundfish
- Indonesian Deep water snapper grouper fishery
- Eastern Pacific Tuna Management
- Integration with European Digital Twin Ocean



What can we put in a minimal fisheries model?

Agents:

- A port,
- fishers,
- and fish!

Source: https://www.tourismeilesdelamadeleine.com/fr/decouvrir-les-iles/les-iles/ile-de-grande-entree

Some policy to test:

- Where to place a marine protected area (MPA)

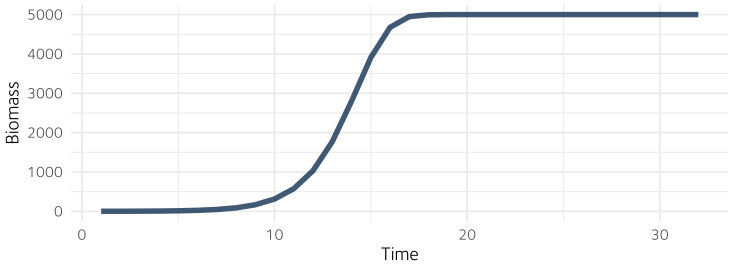
Fisher agents in POSEIDON use the **“Explore, exploit, imitate”** behaviour algorithm:

- Should I go exploring?
  - If yes, pick a random spot not too far from my favourite spot (**EXPLORE**)
  - If not, pick a friend and ask: is my favourite spot better than my friend’s favourite spot?
    - If it is, go to my favourite spot (**EXPLOIT**)
    - If it isn’t, go to my friend’s favourite spot (**IMITATE**)
- If the spot I went to was better than my favourite spot, it becomes my new favourite!

A bit of biology

We do not simulate individual fish. We keep track of the total biomass and apply yearly **logistic growth**.

$$\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right)$$



Pierre-François Verhulst (1804–1849).
Source: `https://w.wiki/x2o`.

Get the model skeleton from GitHub

https://github.com/
nicolaspayette/mini-poseidon

Explore, exploit, imitate...

```
to pick-destination ; fisher procedure
  ifelse random-float 1 < exploration-probability [
    ; explore:
    let r 1 + random-poisson exploration-radius
    set trip-destination [ one-of fishable-patches in-radius r ] of favourite-destination
  ] [
    let other-fisher one-of other fishers
    let their-profits [ profits-at-favourite-destination ] of other-fisher
    ifelse profits-at-favourite-destination >= their-profits [
      ; exploit:
      set trip-destination favourite-destination
    ] [
      ; imitate
      set trip-destination [ favourite-destination ] of other-fisher
    ]
  ]
  set current-destination trip-destination
end
```

Let's go!

```
to go
  ask fishers [
    set trip-costs trip-costs + hourly-costs
    ifelse patch-here = current-destination [
      ifelse any? ports-here [ dock ] [ fish ]
    ] [
      face current-destination
      forward speed
    ]
  ]
  update-biology
  tick
end
```

Fishing!

```
to fish ; fisher procedure
  set pcolor red
  let biomass-caught biomass * catchability
  set biomass biomass - biomass-caught
  set biomass-in-hold biomass-in-hold + biomass-caught
  set current-destination [ patch-here ] of one-of ports
end
```

Docking at the port

```
to dock ; fisher procedure
  let revenues biomass-in-hold * price-of-fish
  set biomass-in-hold 0
  let profits revenues - trip-costs
  set trip-costs 0
  set bank-balance bank-balance + profits
  (ifelse
    trip-destination = favourite-destination [
      set profits-at-favourite-destination profits
    ]
    profits > profits-at-favourite-destination [
      set favourite-destination trip-destination
      set profits-at-favourite-destination profits
    ]
  )
  pick-destination
end
```

Updating the biology

$$\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right)$$

```
to update-biology
  diffuse biomass diffusion-rate
  recolor-patches
  if ticks mod (15 * 24) = 0 [ ; every 15 days
    ask patches [
      set biomass biomass + (
        growth-rate * biomass * (1 - (biomass / carrying-capacity))
      )
    ]
  ]
end
```

Scenario

We have observed the real-world fishery for one year.

Fishers ended up with a mean bank balance of £775,000.

We want to:

- Use this information to estimate the EEI algorithm parameters.
- Simulate the fishery for three more years under the "business as usual" scenario.
- Figure out the best location for an MPA.

Estimating EEI parameters

Parameter Specification:

```
["exploration-probability" [0 "C" 1]]
["exploration-radius" [1 "C" 11]]
```

Measure:

```
abs (775000 - mean [bank-balance] of fishers)
```

Stop If: [blank]

Step Limit: 8760

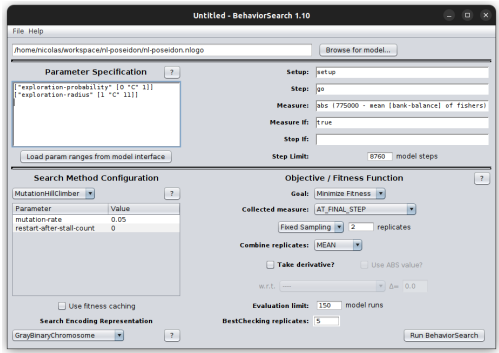
Search Method: MutationHillClimber

Goal: Minimize Fitness

Fixed Sampling: 4 replicates

Evaluation limit: 150 model runs

BestChecking replicates: 8

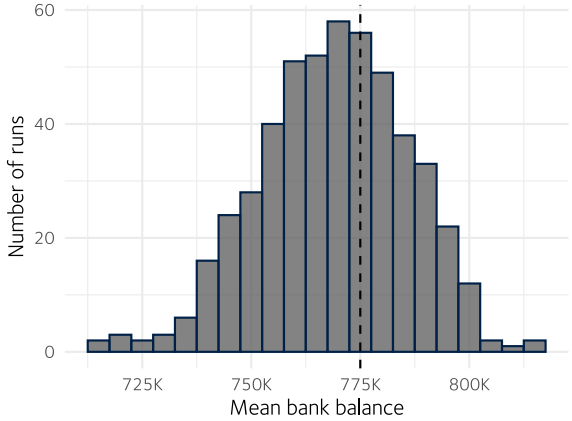


Loading BehaviorSearch results automatically

```
to load-bsearch-min [ prefix ]
  let rows csv:from-file word prefix ".finalCheckedBests.csv"
  let headers item 0 rows
  ; Grab the row with the best (i.e. lowest) fitness:
  let data first sort-by [ [row1 row2] ->
    last row1 < last row2
  ] but-first rows
  foreach range length headers [ index ->
    ; the headers ending with "*" indicate model parameters:
    if last item index headers = "*" [
      ; remove the "*" from the header:
      let param but-last item index headers
      let value item index data
      run (word "set " param " " value)
    ]
  ]
end
```

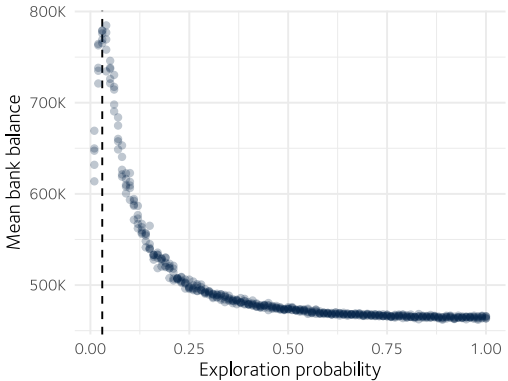
Goodness of fit

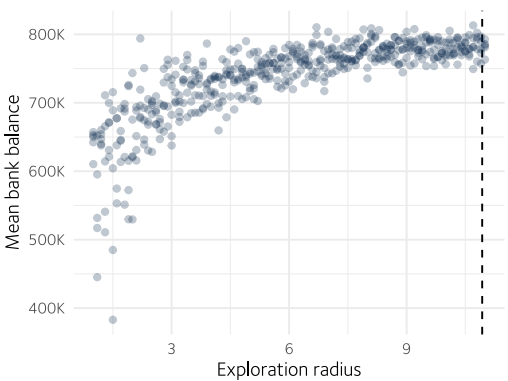
```
to check-goodness-of-fit
  reset-parameters
  load-bsearch-min "fit-eei"
  let balances [ [ "balance" ] ]
  foreach range 500 [ i ->
    print word "Run " i
    setup
    repeat 365 * 24 [ go ]
    let result mean [ bank-balance ] of fishers
    set balances lput (list result) balances
  ]
  csv:to-file "goodness-of-fit.csv" balances
end
```



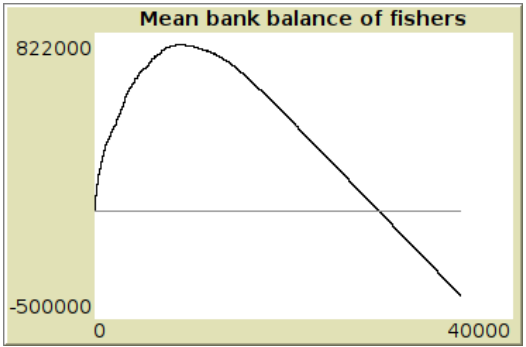
Testing sensititivy of EEI parameters

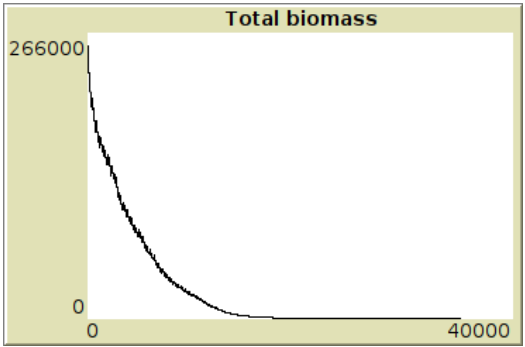
"One-at-time sensitivity": vary one parameter while keeping others fixed.





Not looking good for the fishery...





Adding a protected area after one year

```
to setup-mpa
  set-default-shape xs "x"
  ask patches with [
    pxcor >= min-mpa-x and
    pxcor <= max-mpa-x and
    pycor >= min-mpa-y and
    pycor <= max-mpa-y and
    not any? ports-here
  ] [
    sprout-xs 1 [ set color [ 0 0 0 50 ] ]
  ]
  set fishable-patches fishable-patches with [ not any? xs-here ]
end
```

Modify go

```
to go
  if ticks = 365 * 24 [ setup-mpa ]
  ask fishers [
    set trip-costs trip-costs + hourly-costs
    ifelse patch-here = current-destination [
      ifelse any? ports-here [ dock ] [ fish ]
    ] [
      face current-destination
      forward speed
    ]
  ]
  update-biology
  tick
end
```

Modify dock

```
to dock ; fisher procedure
  let revenues biomass-in-hold * price-of-fish
  set biomass-in-hold 0
  let profits revenues - trip-costs
  set trip-costs 0
  set bank-balance bank-balance + profits
  (ifelse
    trip-destination = favourite-destination [
      set profits-at-favourite-destination profits
    ]
    profits > profits-at-favourite-destination [
      set favourite-destination trip-destination
      set profits-at-favourite-destination profits
    ]
  )
  if [ any? xs-here ] of favourite-destination [
    set favourite-destination min-one-of fishable-patches [
      distance [ favourite-destination ] of myself
    ]
  ]
  pick-destination
end
```

Searching for the optimal MPA

Parameter Specification:

```
["exploration-probability" 0.021]
["exploration-radius" 10.578]
["min-mpa-x" [-5 1 5]]
["max-mpa-x" [-5 1 5]]
["min-mpa-y" [-5 1 5]]
["max-mpa-y" [-5 1 5]]
```

Measure:

```
mean [bank-balance] of fishers
```

Stop If: [blank]

Step Limit: 35040

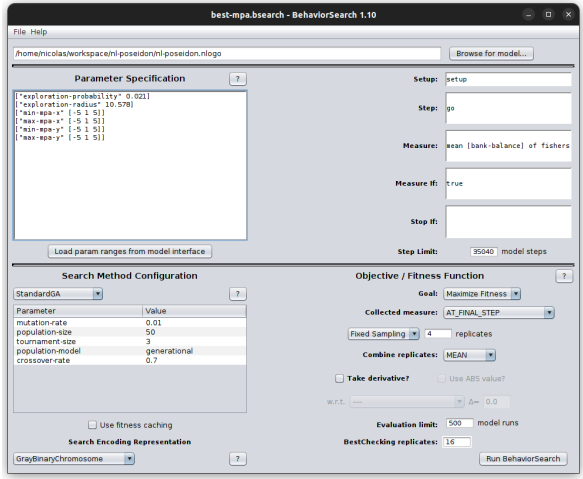
Search Method: StandardGA

Goal: Maximize Fitness

Fixed Sampling: 4 replicates

Evaluation limit: 300 model runs

BestChecking replicates: 8



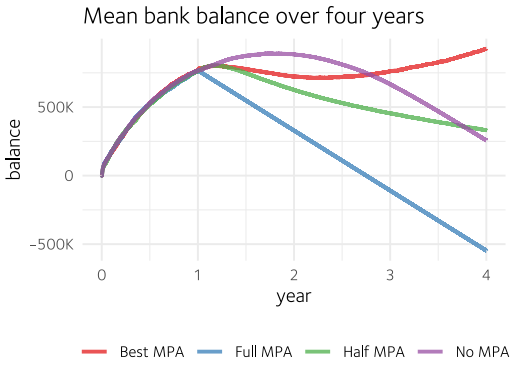
Generalizing BehaviorSearch load procedure

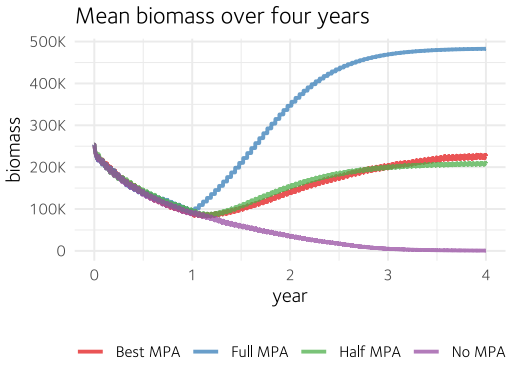
```
to load-bsearch [ prefix sort-criteria ]
  let rows csv:from-file word prefix ".finalCheckedBests.csv"
  let headers item 0 rows
  ; Grab the row with the best fitness according to criteria:
  let data first sort-by sort-criteria but-first rows
  foreach range length headers [ index ->
    ; the headers ending with "*" indicate model parameters:
    if last item index headers = "*" [
      ; remove the "*" from the header:
      let param but-last item index headers
      let value item index data
      run (word "set " param " " value)
    ]
  ]
end

to load-bsearch-min [ prefix ]
  load-bsearch prefix [ [row1 row2] -> last row1 < last row2 ]
end

to load-bsearch-max [ prefix ]
  load-bsearch prefix [ [row1 row2] -> last row1 > last row2 ]
end
```

Comparing four different scenarios





So, what is the optimal MPA?