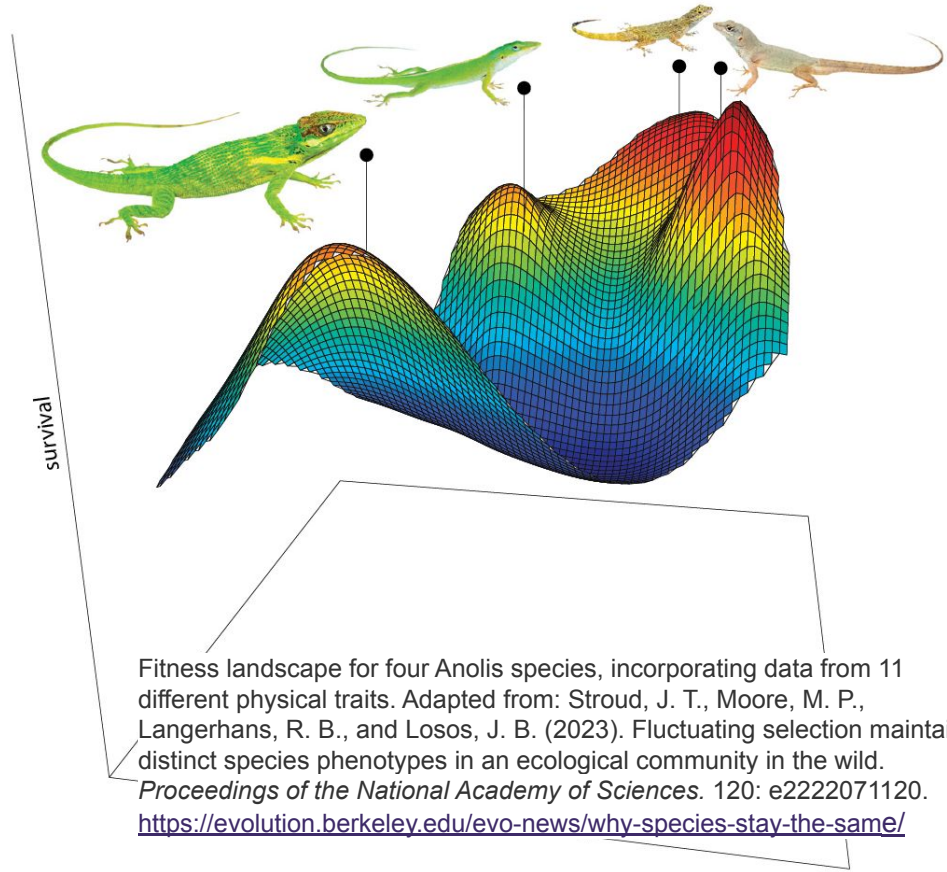


# Bio-AI Assignment 3

Xavier F. C. Sánchez Díaz



# Overview

- I. Understanding the assignment
- II. Step by step guide
  - A. Choosing data and models
  - B. Creating your tables
  - C. Visualizing the landscape
  - D. Implementation
  - E. Experimentation and Comparison
  - F. Testing instances

Understanding the assignment

# **A general idea**

# A general idea

We're **doing science** in this project, a nice prelude to your work on the MS thesis.

- You will **compare algorithms** on different problems
- You will **ask questions** and **draw conclusions** on your findings
- You will **communicate these findings** to others

# How?

Picture yourself playing Mario Kart:

- With the **data** and **models**, you will create different **racing tracks**
- By coding your **BioAI algorithms**, you will create different **karts**



© Nintendo <https://www.gamer.no/artikler/hjula-i-taket-og-bremsespor-pa-tapeten/159874>

You will **observe** a *grand prix* and **report** that!

What should I do at each step?

# Step by step guide

# Symbols used in these slides



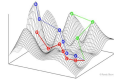
- Dataset



- Vector of Labels



- An ML Model



- A Problem Landscape



- Lookup Table



- Fitness Function

# 1: Choose datasets and ML model

- You need to choose ***d*** datasets and ***m*** ML models and are constrained to choose values for ***d*** and ***m***:
  - ***d x m = 3***, so that means
    - $d=1$  and  $m=3$ , OR
    - $d=3$  and  $m=1$
- 1 dataset and 3 ML models (more ML engineering)
  - is similar to *3 difficulties of 1 track*
- 3 datasets and 1 ML model (more data science)
  - is similar to *3 tracks of 1 difficulty*

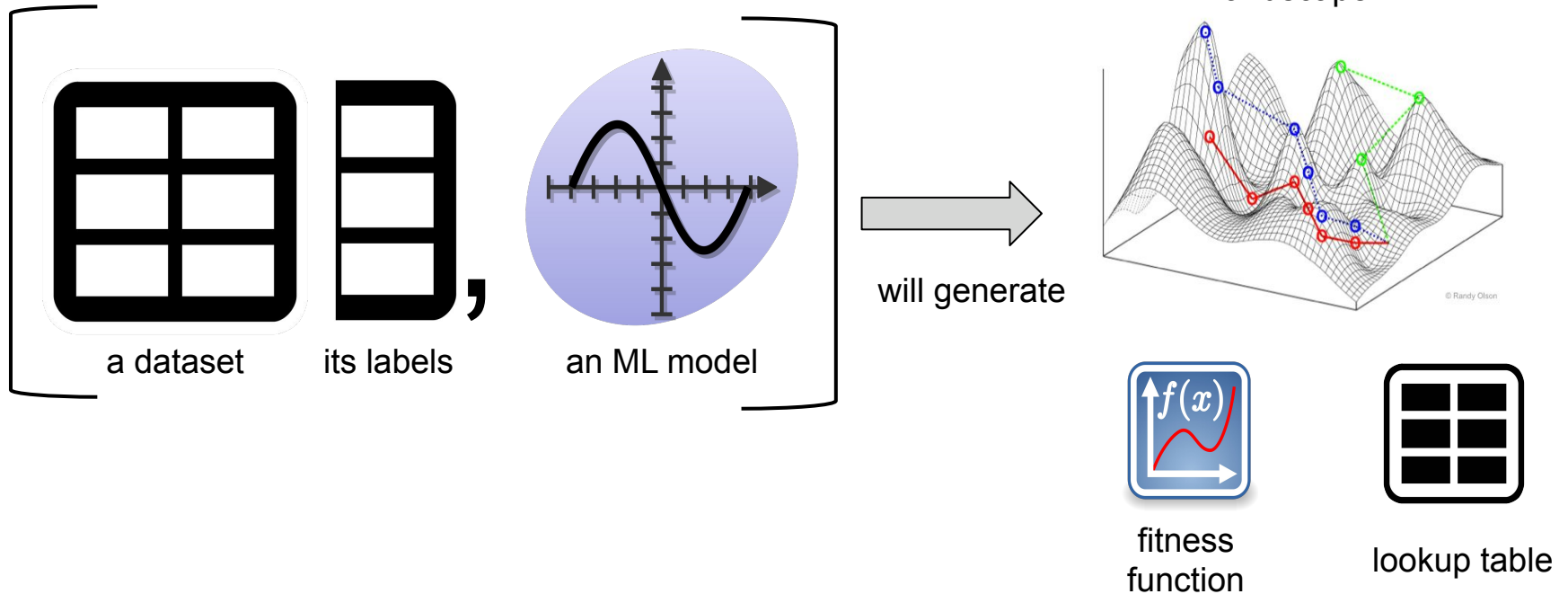




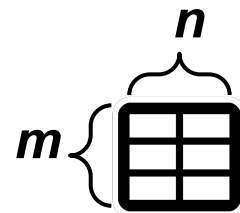
## 2: Lookup table construction

**one dataset X one ML model = one landscape**

# Here's how (conceptually)



# Here's how (conceptually)



For each combination of features ( $n$  cols) in your **dataset**:



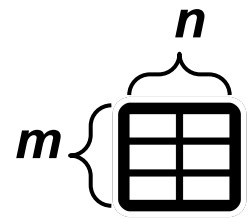
- **Train** your **ML model** using a **fraction** of the  $m$  rows
- **Test** your trained **ML model** using the remaining rows

Then **record** the accuracy/error in your **lookup table**

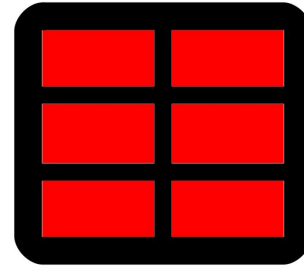
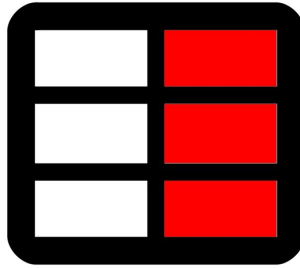
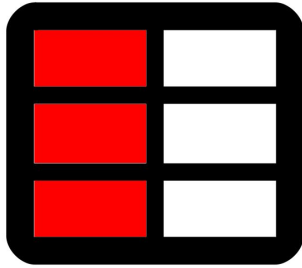
- Each combination of features is a **bitstring**
  - 1 if the model uses the feature, 0 otherwise
  - Each bitstring has an accuracy/error value



# Here's how (conceptually)



For each combination of features ( $n$  cols) in your **dataset**:



with  $n$  columns in your **dataset**, you need to do this  $2^n - 1$  times!  
so your **lookup table** will have  $2^n - 1$  rows



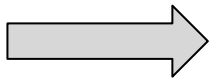
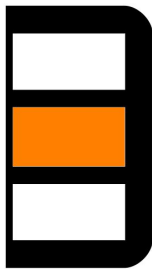
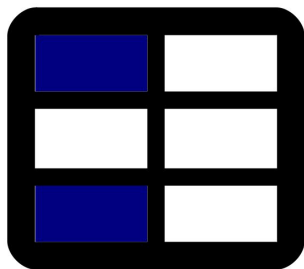
# Here's how (conceptually)



For each combination of features (cols) in your **dataset**:



- **Train** your **ML model** using a **fraction** of the rows
- **Test** your trained **ML model** using the remaining rows
- Then **record** the accuracy/error in your **lookup table**

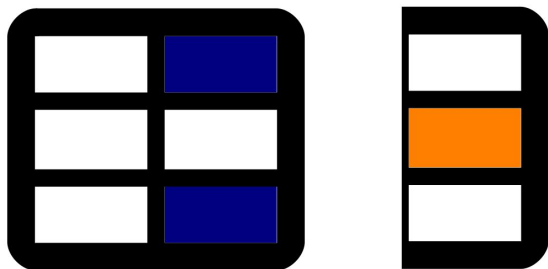
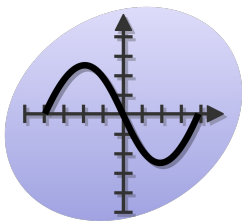


feat_subset	bitstring	error
1	01	0.345

# Here's how (conceptually)



For each combination of features (cols) in your **dataset**

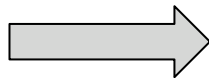
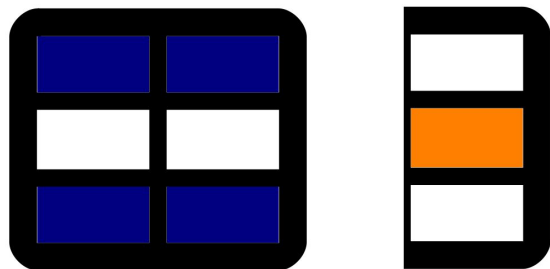
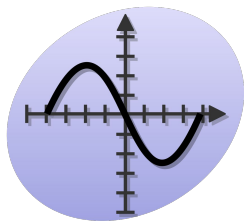


feat_subset	bitstring	error
1	01	0.345
2	10	0.489

# Here's how (conceptually)



For each combination of features (cols) in your **dataset**



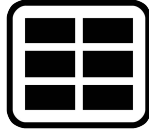
feat_subset	bitstring	error
1	01	0.345
2	10	0.489
3	11	0.12

# That's it

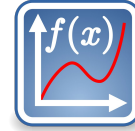
	n	ind	acc	err	err32	err16	err8
1	0	0000000	0.0	1.0	1.0	1.0	1.0
2	1	0000001	0.8811881188118812	0.1188118811881188	0.1500618811881188	0.1813118811881188	0.2438118811881188
3	2	0000010	0.7623762376237624	0.2376237623762376	0.2688737623762376	0.3001237623762376	0.3626237623762376
4	3	0000011	0.7524752475247525	0.2475247524752475	0.3100247524752475	0.3725247524752475	0.4975247524752475
5	4	0000100	0.8217821782178217	0.1782178217821782	0.2094678217821782	0.2407178217821782	0.3032178217821782
6	5	0000101	0.7524752475247525	0.2475247524752475	0.3100247524752475	0.3725247524752475	0.4975247524752475
7	6	0000110	0.7722772277227723	0.2277227722772277	0.2902227722772277	0.3527227722772277	0.4777227722772277
8	7	0000111	0.801980198019802	0.1980198019801979	0.291769801980198	0.385519801980198	0.573019801980198
9	8	0001000	0.801980198019802	0.1980198019801979	0.2292698019801979	0.260519801980198	0.323019801980198
10	9	0001001	0.7722772277227723	0.2277227722772277	0.2902227722772277	0.3527227722772277	0.4777227722772277
11	10	0001010	0.7425742574257426	0.2574257425742574	0.3199257425742574	0.3824257425742574	0.5074257425742574
12	11	0001011	0.7722772277227723	0.2277227722772277	0.3214727722772277	0.4152227722772277	0.6027227722772277
13	12	0001100	0.7227722772277227	0.2772277227722772	0.3397277227722772	0.4022277227722772	0.5272277227722773
14	13	0001101	0.7920792079207921	0.2079207920792078	0.3016707920792079	0.3954207920792079	0.5829207920792079
15	14	0001110	0.7425742574257426	0.2574257425742574	0.3511757425742574	0.4449257425742574	0.6324257425742574
16	15	0001111	0.8415841584158416	0.1584158415841585	0.2834158415841585	0.4084158415841585	0.6584158415841584
17	16	0010000	0.7920792079207921	0.2079207920792078	0.2391707920792078	0.2704207920792079	0.3329207920792079
18	17	0010001	0.8316831683168316	0.1683168316831683	0.2308168316831683	0.2933168316831683	0.4183168316831683
19	18	0010010	0.7326732673267327	0.2673267326732673	0.3298267326732673	0.3923267326732673	0.5173267326732673
20	19	0010011	0.8217821782178217	0.1782178217821782	0.2719678217821782	0.3657178217821782	0.5532178217821783
21	20	0010100	0.7722772277227723	0.2277227722772277	0.2902227722772277	0.3527227722772277	0.4777227722772277
22	21	0010101	0.7722772277227723	0.2277227722772277	0.3214727722772277	0.4152227722772277	0.6027227722772277
23	22	0010110	0.6633663366336634	0.3366336633663366	0.4303836633663366	0.5241336633663366	0.7116336633663366
24	23	0010111	0.7920792079207921	0.2079207920792078	0.3329207920792079	0.4579207920792079	0.7079207920792079
25	24	0011000	0.7524752475247525	0.2475247524752475	0.3100247524752475	0.3725247524752475	0.4975247524752475
26	25	0011001	0.8514851485148515	0.1485148514851485	0.2422648514851485	0.3360148514851485	0.5235148514851485
27	26	0011010	0.7920792079207921	0.2079207920792078	0.3016707920792079	0.3954207920792079	0.5829207920792079



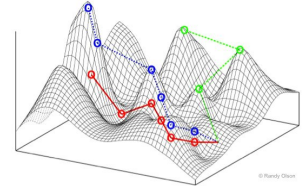
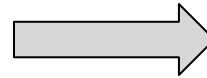
# Why?



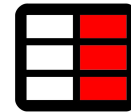
- Each row in your **table** represents one **state** (individual)
- Each **error value** represents **its fitness**
  - The fitness is the “height” of the landscape
- Your **table** is a numerical representation of **a landscape**



a race track



- Your **algorithms** will **race** towards finding **its minimum**
  - Which you can easily find by looking at your table
  - The optimum is a **subset of features**
  - Its **fitness** will be the best error

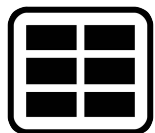


# FAQs & Tips

- Do I need to use **binary strings**?
  - For your **lookup table**, **yes**. However, you can **use real numbers** in your **algorithm representation**, and then **round** to decide if you use a feature or not. This might be helpful for continuous algorithms!
- You can **split the work** (sessions/computers/teammates) and **then concatenate**. Assume  $n=7$ , so 127 rows:
  - Calculate first 64 combinations, store. Then calculate another 63 combinations, store, and then concatenate the two tables.
- Can I use X programming language?
  - Yes, use whatever you like.

# FAQs & Tips

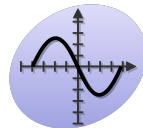
- What about the **penalty** term?
  - Since the penalty is a separate term, you can sum the penalty AFTER you have the error stored. You may want to do another column in **your table** for this.



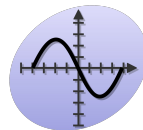
- What are good values for the penalty?
  - 0 means no penalty
  - Too much penalty results in a flatter landscape (very hard!!)
  - Too little penalty might make the landscape very spiky (hard!!)

# FAQs & Tips

- Should I tune the **hyperparameters** of my **ML Model**?
  - It is not necessary. You can use the defaults.
  - We are just looking for the minimum value in a table. It doesn't matter if the minimum is 45% error or 85% error.



- Should I fix any seeds for training my **ML Model**?
  - It is not necessary.
  - You can make it more **trustworthy** and **robust** if you run **multiple models** and then average their accuracy.



- So, let's say, 5 models per each of the  $2^n$  combinations
- Still doable if you split the work!



# 3: Visualization (optional)

Find a nice way to **visualize** your landscapes! Suggestions:

- A. A 2D or 3D plot
- B. A Local Optima Network (LON)
- C. A Hinged Bitstring Map (HBM)
- D. A Distance-Correlation map

# 3: Visualization (optional)

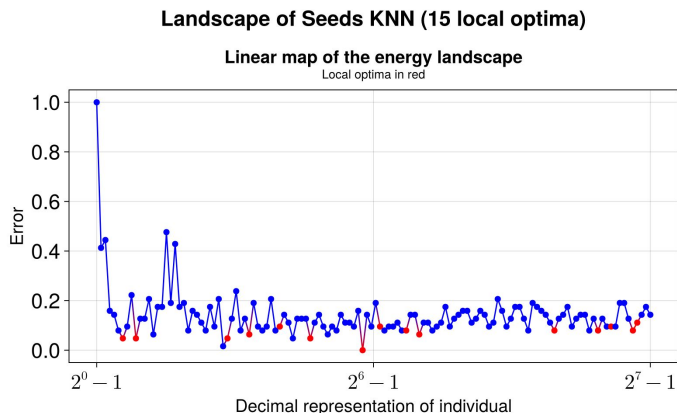
Find a nice way to **visualize** your landscapes! Suggestions:

## A. A 2D or 3D plot

Cheap!

- a. Difficult to find a mapping from **states** to **coordinates**
- b. But fun.

Enjoy!



# 3: Visualization (optional)

Find a nice way to **visualize** your landscapes! Suggestions:

**Complex and  
Expensive!**

## B. A Local Optima Network (LON)

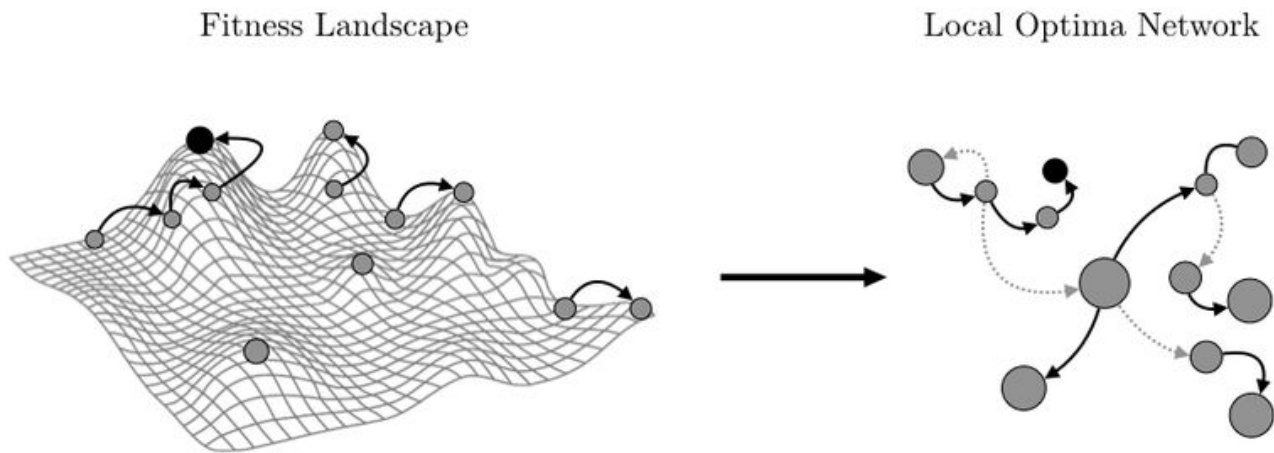
- a. From a **local optimum**, do controlled mutation (fixed *neigh\_size*)
  - i. Count **how many flips** it took *to escape* that local optimum
  - ii. Use this as an *estimator* of the “attraction” of this local optimum
  - iii. This will be the “weight” (and size) of the node
- b. Repeat with **bigger flips**, until you can reach from one optimum to another
  - i. This creates **a link** between two nodes
- c. Now repeat everything until you’re done!

Read  
reference [6]

# 3: Visualization (optional)

Find a nice way to **visualize** your landscapes! Suggestions:

## B. A Local Optima Network (LON)





# 3: Visualization (optional)

Find a nice way to **visualize** your landscapes! Suggestions:

## c. **A Hinged Bitstring Map (HBM)** **Complex but cheap!**

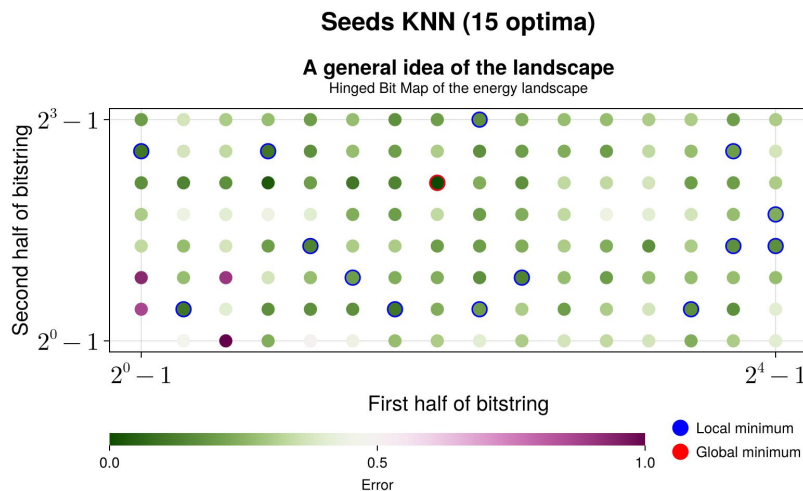
- a. Take **each bitstring** and divide in half
  - i. Convert each half to base 10
  - ii. Left half will be the x-coordinate
  - iii. Right half will be the y-coordinate
- b. Plot as a **scatter plot**, with color=fitness
- c. Add a mark for local/global optima

Read  
reference [8]

# 3: Visualization (optional)

Find a nice way to **visualize** your landscapes! Suggestions:

## c. A Hinged Bitstring Map (HBM)



### 3: Visualization (optional)

Find a nice way to **visualize** your landscapes! Suggestions:

#### d. A Distance-Correlation map

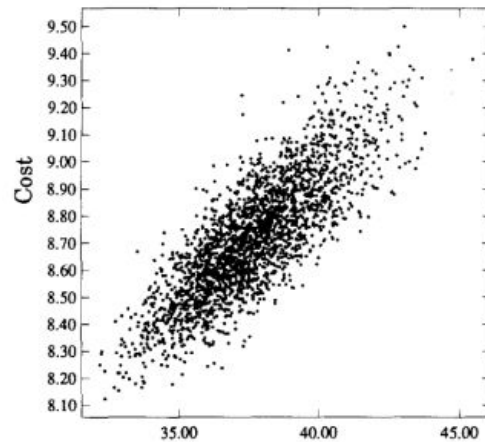
Complex!

- a. Calculate the pairwise **distance** between **local optima** (!!!)
- b. Plot the **distance** from each local optima to its nearest **global optimum** vs their **fitness**
  - i. This works both as a **scatter plot** or a **hexbin plot**

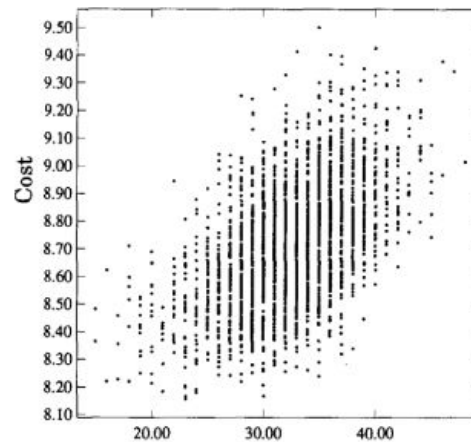
# 3: Visualization (optional)

Find a nice way to **visualize** your landscapes! Suggestions:

## D. A Distance-Correlation map



Ave distance from other local minima



Distance to best local minimum

# FAQs & Tips

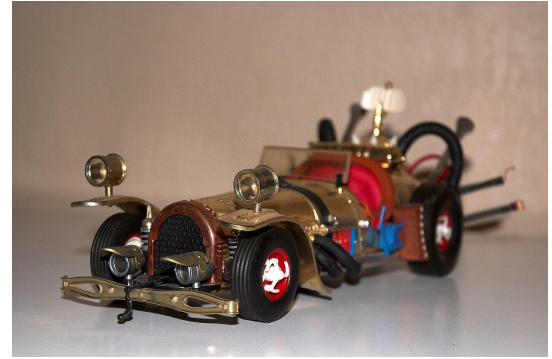
- This takes some work!
  - Yes, but it is optional. You don't need to do it.
- Does it matter which visualization we use?
  - No.
- Can I use X programming language?
  - Yes, use whatever you like.

# 4: Implementation

Now you need to **implement your 3 BioAI algorithms!**

- One **single-objective**
- One **multi-objective**
- One from **swarm intelligence**

These are your **racing cars**



You will use these on both Step 5 and Step 6

# FAQs & Tips

- Can I use my previous EAs/GAs?
  - Yes. You can use your previous algorithms (although they will need some modifications).
- Can I use the ones in EvoLP.jl?
  - Absolutely. However both the EA and the GA in EvoLP won't do any good in Step 6. Those are toy algorithms!
- How can I be sure they will be good for Step 6?
  - If they do well in your 3 landscapes, they *might* work well :)


# FAQs & Tips

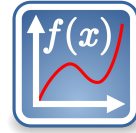
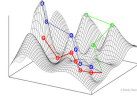
- What are the different **objectives** for the **multi-objective** case?
  - We recommend using the accuracy/error as one objective, and the penalty as a separate, second objective
- Can I use advanced versions of swarm intelligence algorithms for multiple objectives?
  - Yes, go ahead



# 5: Experimentation and Comparison

Experimentation is easy. It's  **racing time**!

- Run your **BioAI algorithms**...
  - several times each (to compensate unlucky runs!)
  - on each **landscape** using the **table values** as **fitness**



- Record your results
- **Calculate statistics** about the experiments
- Ask **questions\*** and try to find the answers

# FAQs & Tips

- \*What kind of questions should I ask?
  - *Who won? Why?*
  - *Is one problem more difficult than the others?*
  - *What happens if you increase/reduce the penalty?*
  - *What are the hyper-parameters your algorithms need to find the solution faster? What helped the most?*
- What kind of statistics should I compute?
  - Min, average and max fitness per generation, per run, at least.
  - Average and Standard deviation of best fitness throughout runs

# **Check the slides from March 27!**

They contain info about working with  
EAs and running experiments!

# 6: Test instances

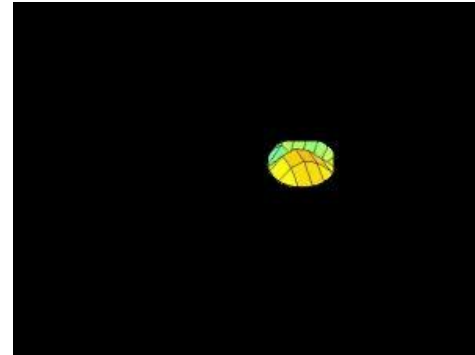
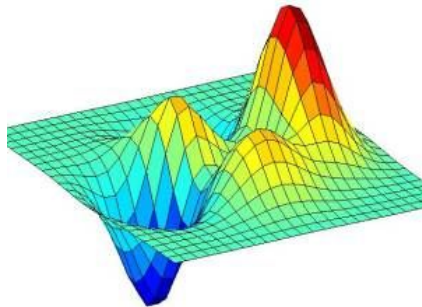
You will receive **3 test “tasks”** *close* to the demo day.

These tasks will look something similar to the following:

- Using these **specific datasets**
  - And this **specific ML model**
    - with these **specific hyperparameters**
    - and **averaging X copies** of the same **model+feature combination**
    - with this **specific penalty values**
- Try to find the **best feature combination** represented as a bitstring.

# 6: Test instances

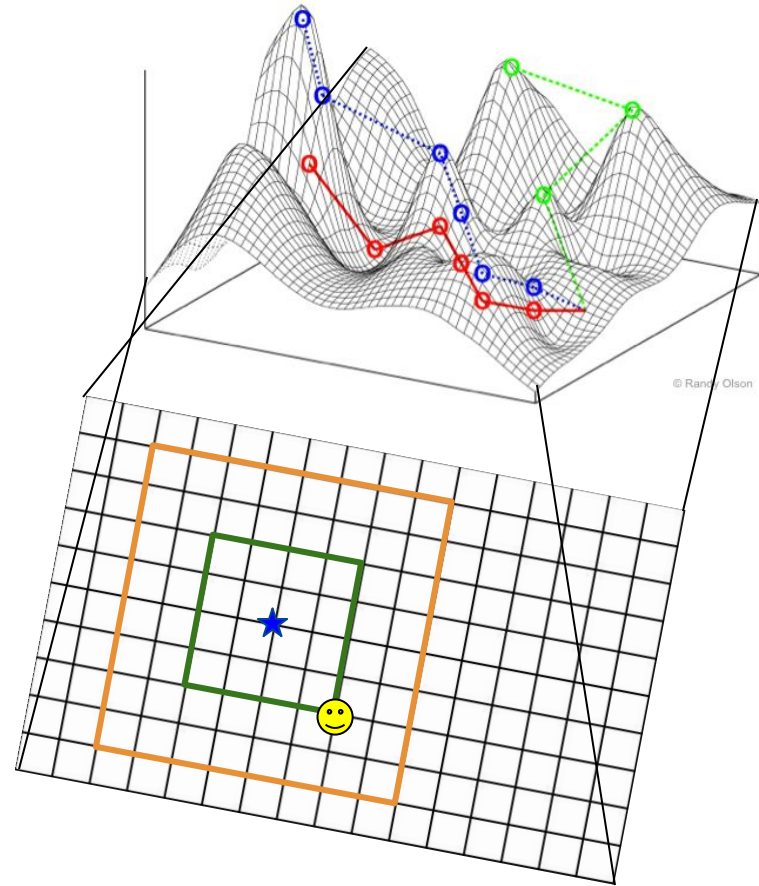
- Try to find the best feature combination (a bitstring)
  - Using any of your **BioAI algorithms**
  - Setting their parameters as you see fit
  - You don't have the full table
  - Your algorithms will **race** through *thick fog*
  - Each step is very expensive



# 6: Test instances



We know the **solution**, and will calculate the **Hamming distance** between that, and your *best attempt*.



Illustrative picture. The landscape is not like this :)

Modelling and Implementation (Theoretical tips)

# **Reminder from our previous assignment lecture**

# Make it Parallel

Your laptop has **multiple cores**. Use all of them!

- **Difficulty** depends on communication complexity:
  - Multithreading: easy (and can easily halve the running time of many functions)
  - Distributed: medium (for example combining the powers of your computers and trying an island model)
  - Message Passing: hard (can create a lot of new problems)



# Use a surrogate model

If fitness **evaluation** is **expensive**, make a simpler, faster model that serves as an *approximator*, e.g.:

1. Run multiple simulations (get a good sample)
2. Create a table
3. Train an ML model to approximate the table
4. Use the ML model as the fitness evaluation

**Difficulty:** hard (and time consuming), unless you have experience with the particular ML model

Example: Evaluating a trained small neural network is actually just a matrix calculation.