Metaheuristic

CS454 AI-Based Software Engineering

Shin Yoo

Teacher:
add numbers from 1 to 100!

Young Gauss:
1 + 2 + ... + 50
+ 100 + 99 + … + 51
= 101 + 101 + … + 101
= 101 * 50 = 5050

Computer: Is it 782?
Teacher: Nope.
Computer: Is it 783?
Teacher: Nope.
….
Computer: Is it 5050?
Teacher: Yes (finally…)

Let’s start with terms.

heuristic | ˌhjʊ(ə)ˈrɪstɪk |

adjective

enabling a person to discover or learn something for themselves. *a 'hands-on' or interactive heuristic approach to learning.*

• Computing proceeding to a solution by trial and error or by rules that are only loosely defined.

meta- | ˈmɛtə | (also **met-** before a vowel or h)

combining form

1 denoting a change of position or condition: *metamorphosis*.

2 denoting position behind, after, or beyond: *metacarpus*.

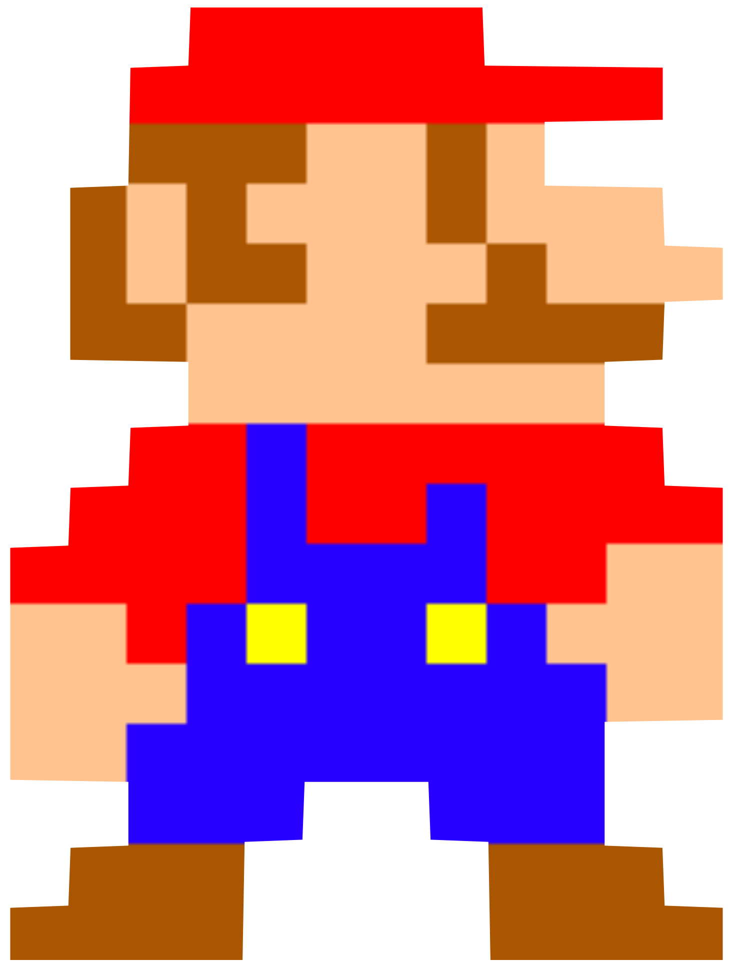
3 denoting something of a higher or second-order kind: *metalanguage* | *metonym*.

4 Chemistry denoting substitution at two carbon atoms separated by one other in a benzene ring, e.g. in 1,3 positions: *metadichlorobenzene*. Compare with ORTHO- and PARA-[1] (SENSE 2).

5 Chemistry denoting a compound formed by dehydration: *metaphosphoric acid*.

Meta-heuristic

- Strategies that guide the search of the acceptable solution
- Approximate and usually non-deterministic
- Not problem specific
- Smart trial and error



Let's play Super Mario Bros.

http://arxiv.org/pdf/1203.1895v3.pdf

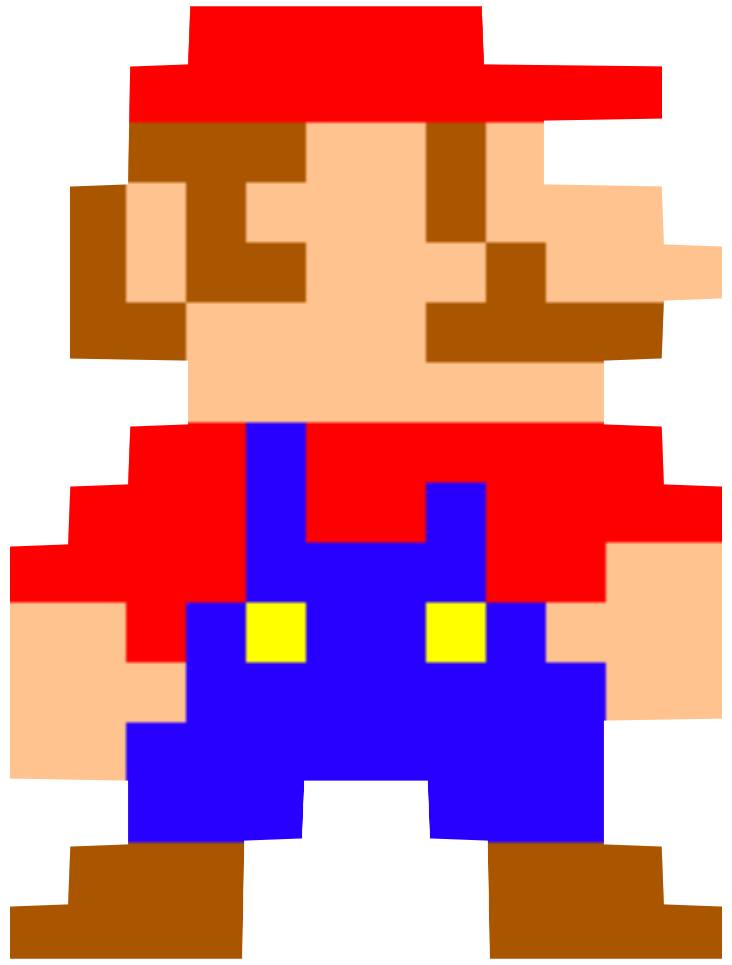
Classic Nintendo Games are (Computationally) Hard

Greg Aloupis* Erik D. Demaine† Alan Guo†‡ Giovanni Viglietta§

February 10, 2015

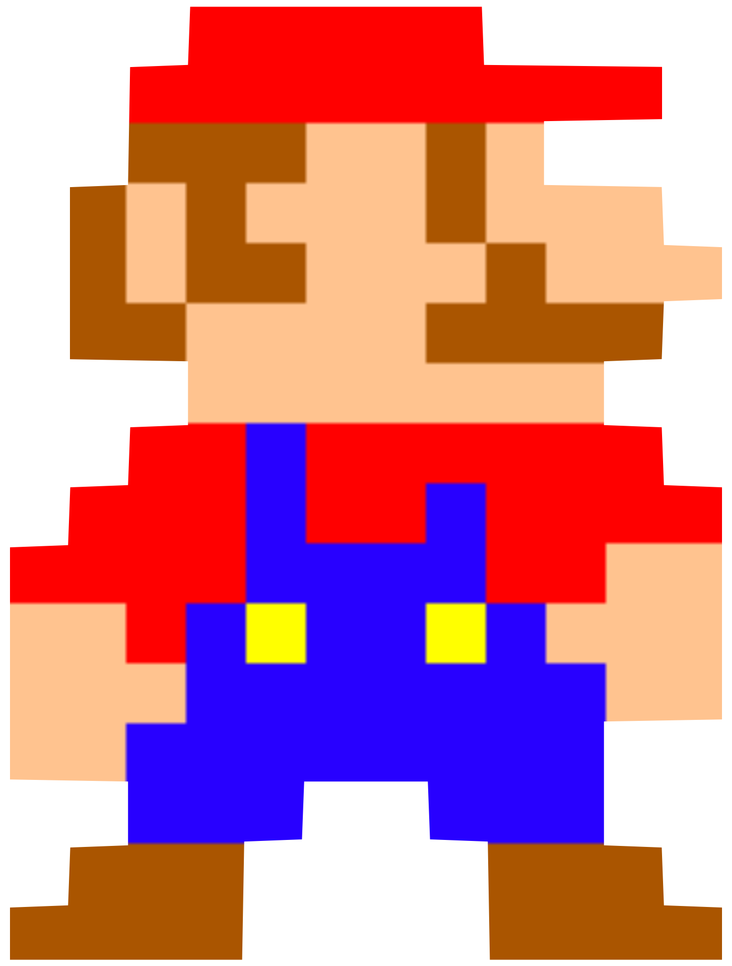
Player A

- Read the game manual to see which button does what.
- Google the level map and get familiar with it.
- Carefully, very carefully, plan when to press each button, for how long.
- Grab the controller and execute the plan.



Player B

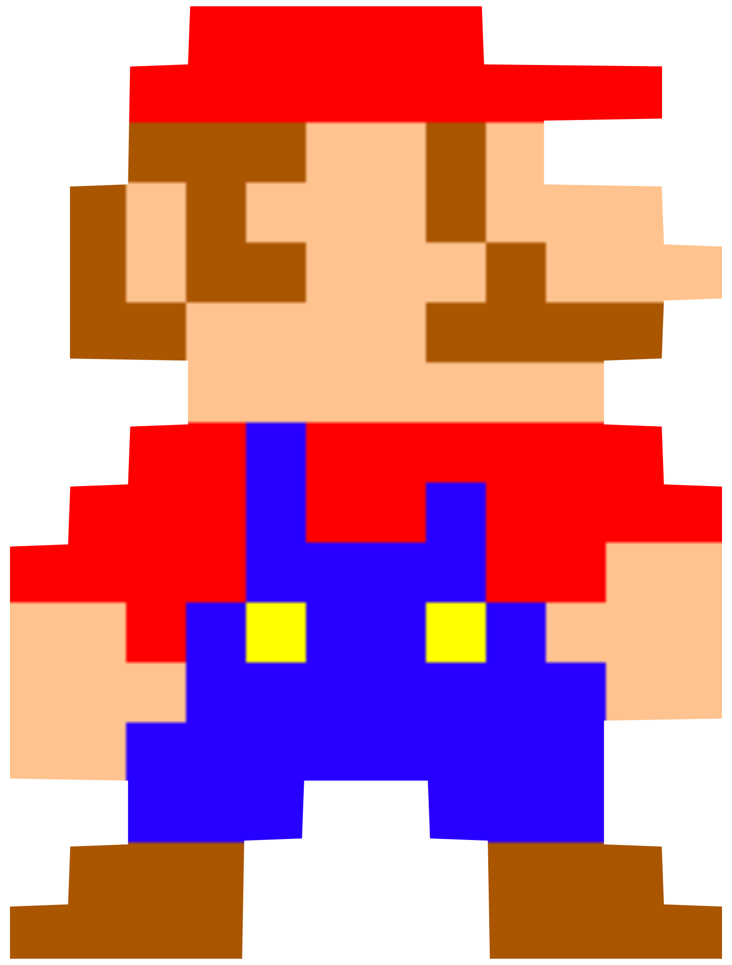
- Grab the controller.
- Play.
- Die.
- Repeat until level is cleared.



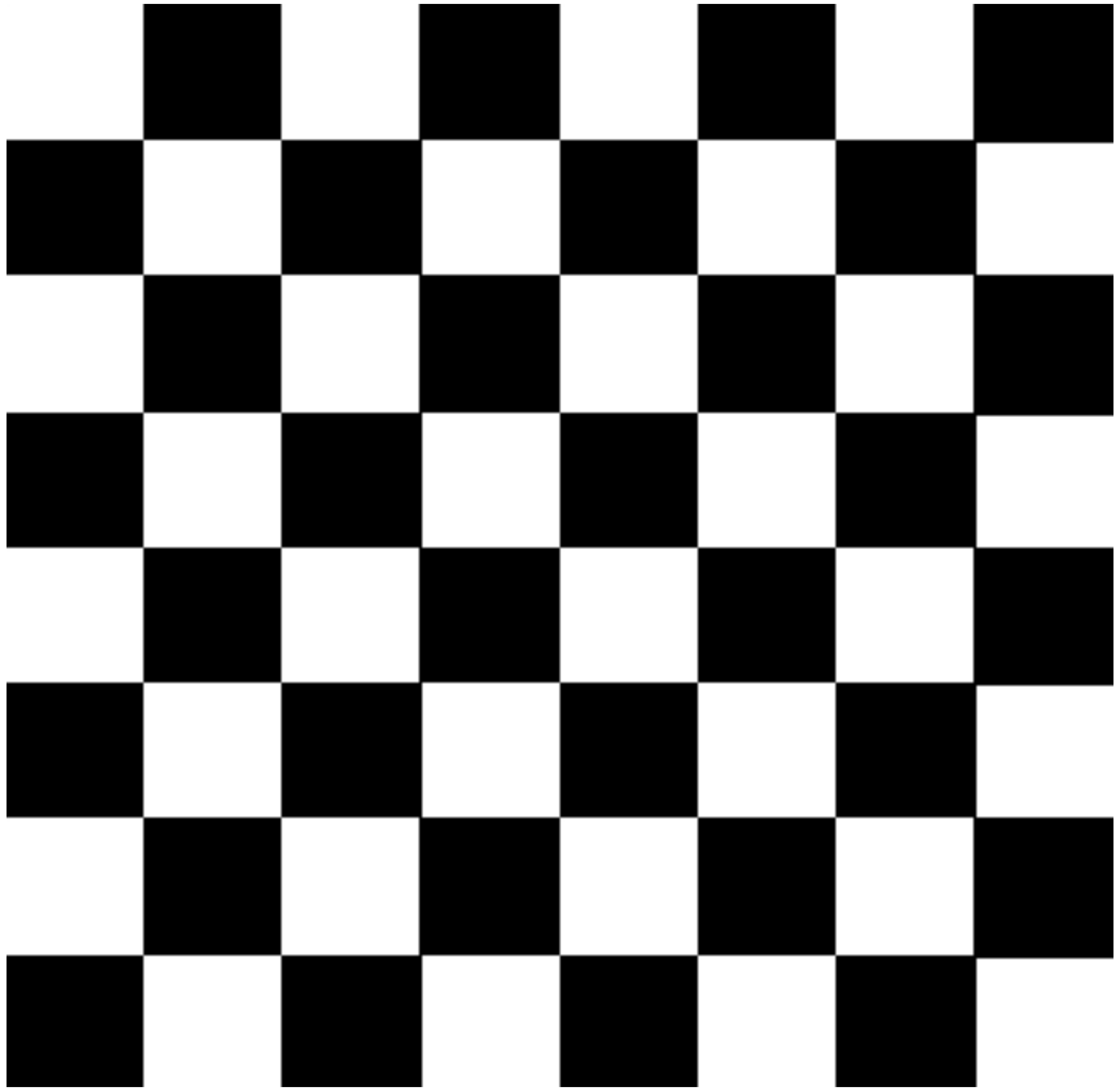
Intelligence lies in how differently you die next time.

Mario analogy goes a long way

- You make small changes to your last attempt ("okay, I will press A slightly later this time").
- You combine different bits of solutions ("Okay, jump over here, but then later do not jump over there").
- You accidentally discover new parts of the map ("Oops, how did I find this secret passage?")

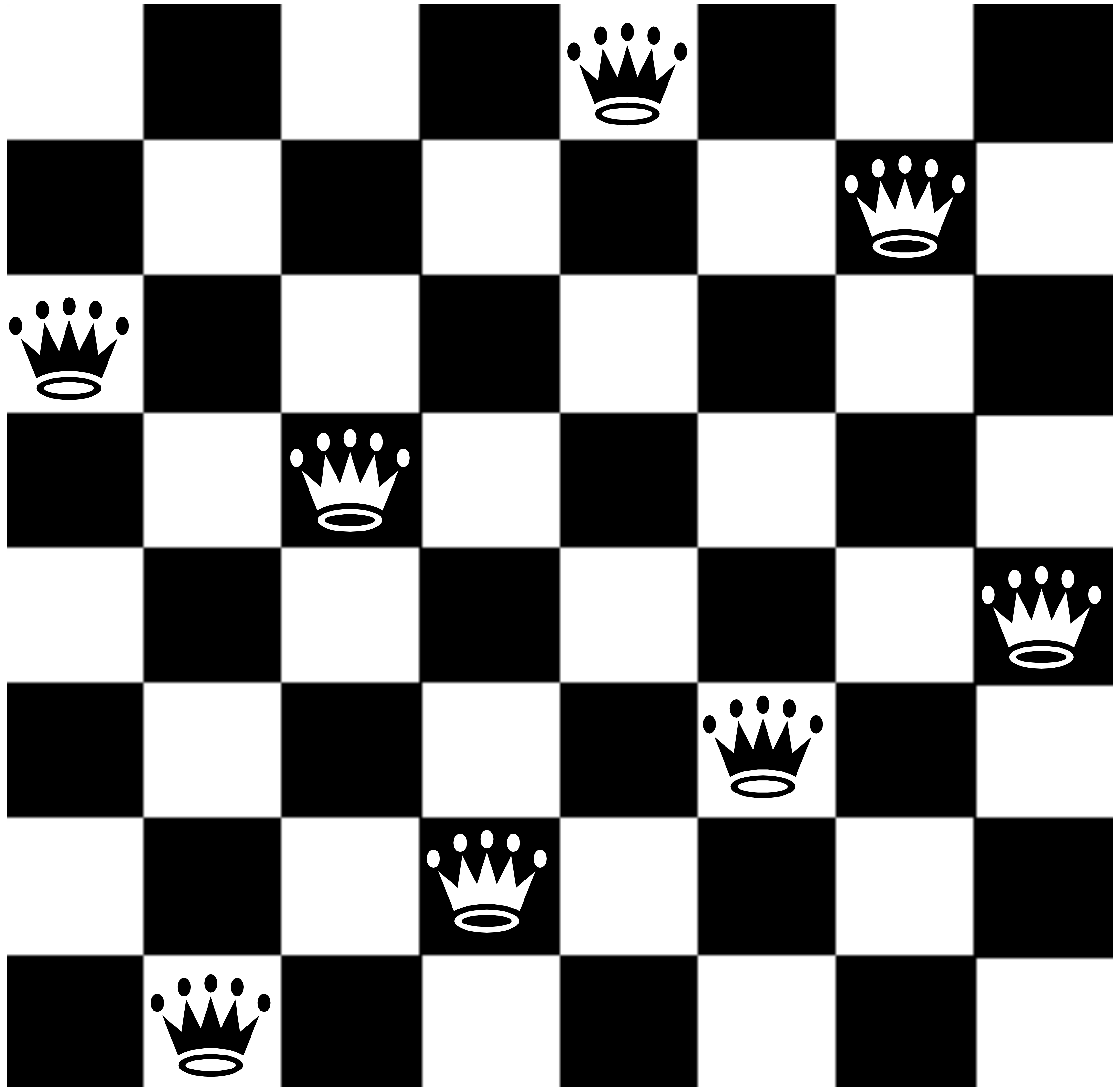


Eight Queens Problem



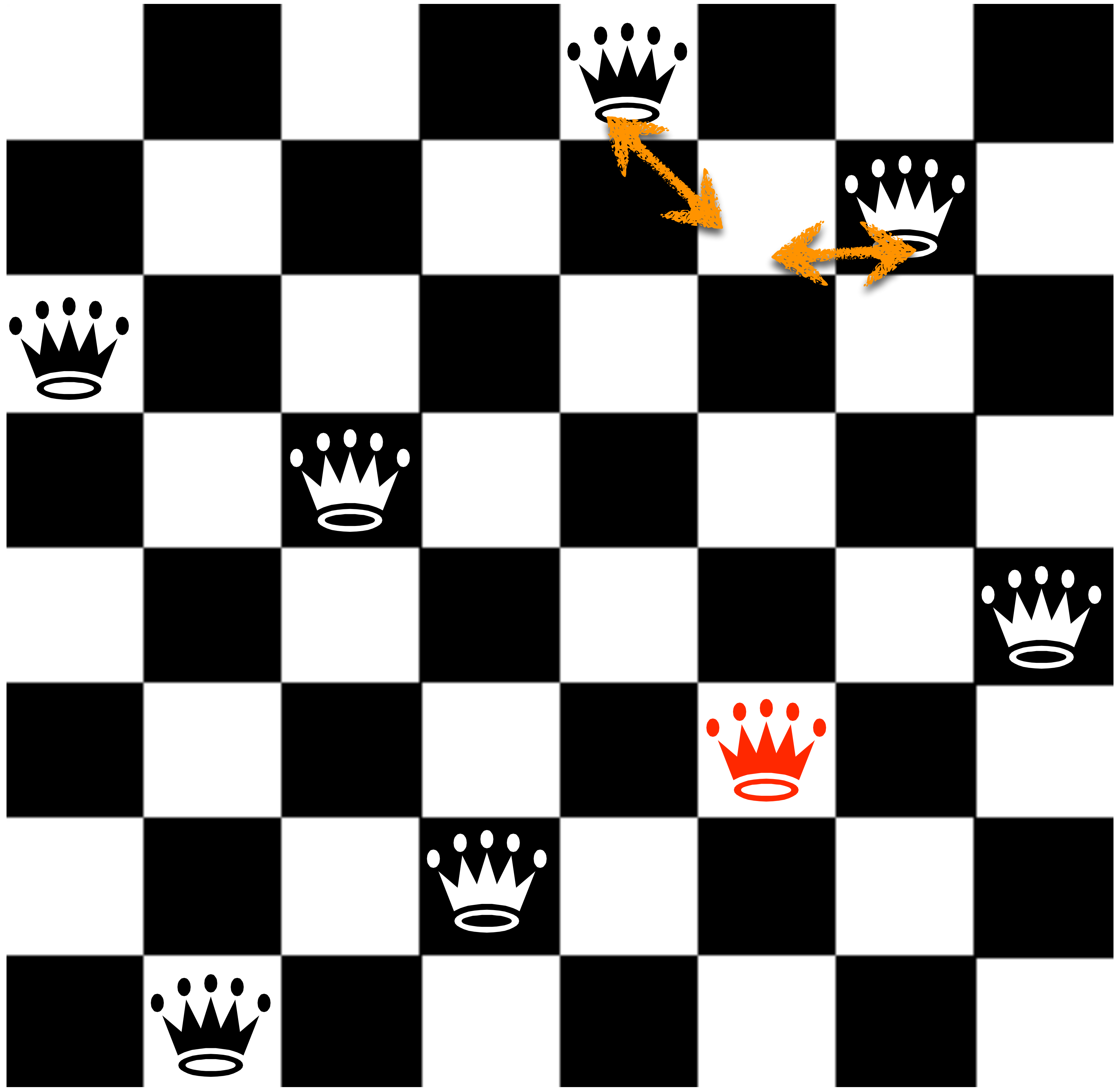
Place 8 queens on a chessboard so that no pair attacks each other.

Eight Queens Problem



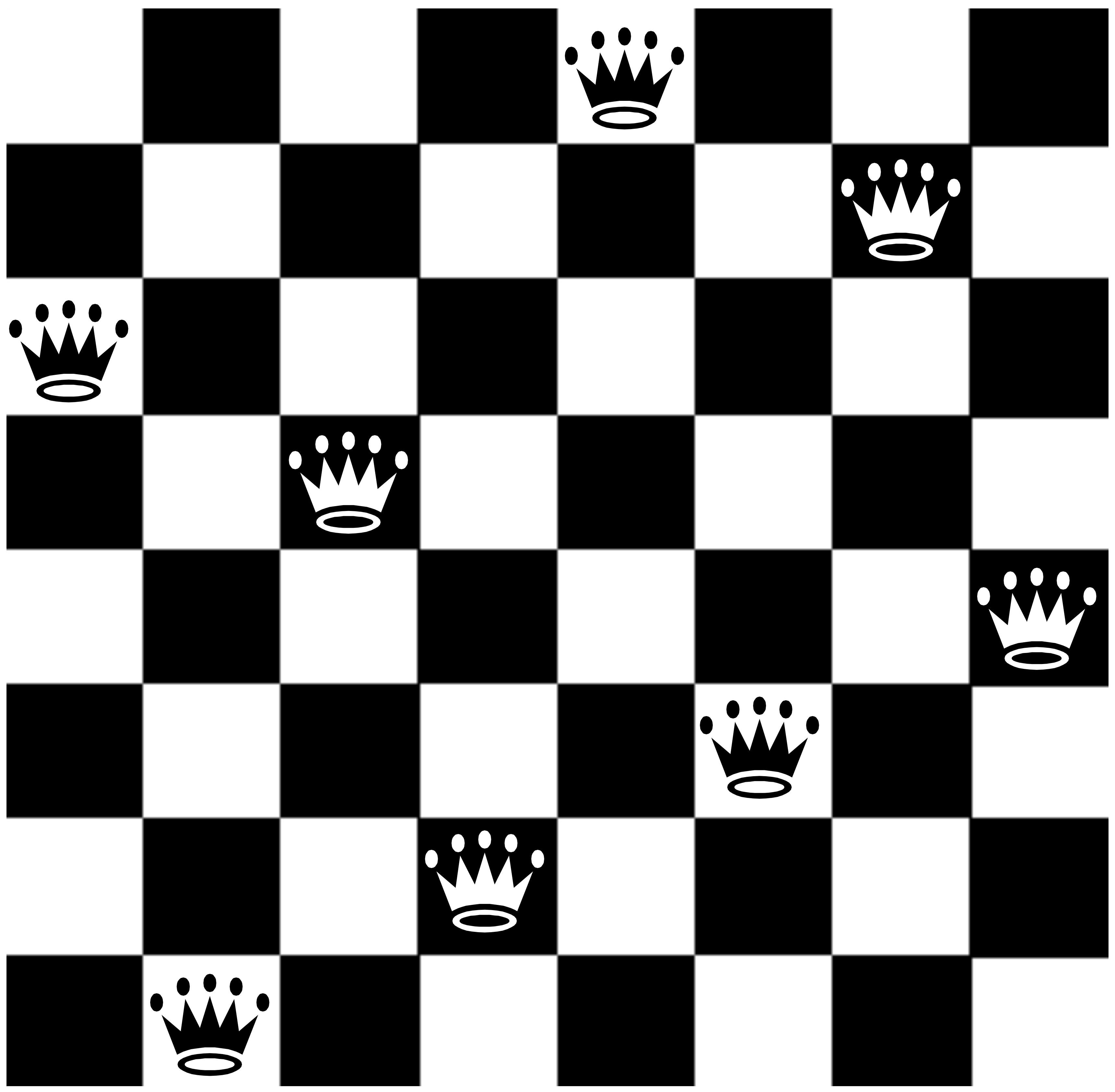
Perfect solution: score 0

Eight Queens Problem

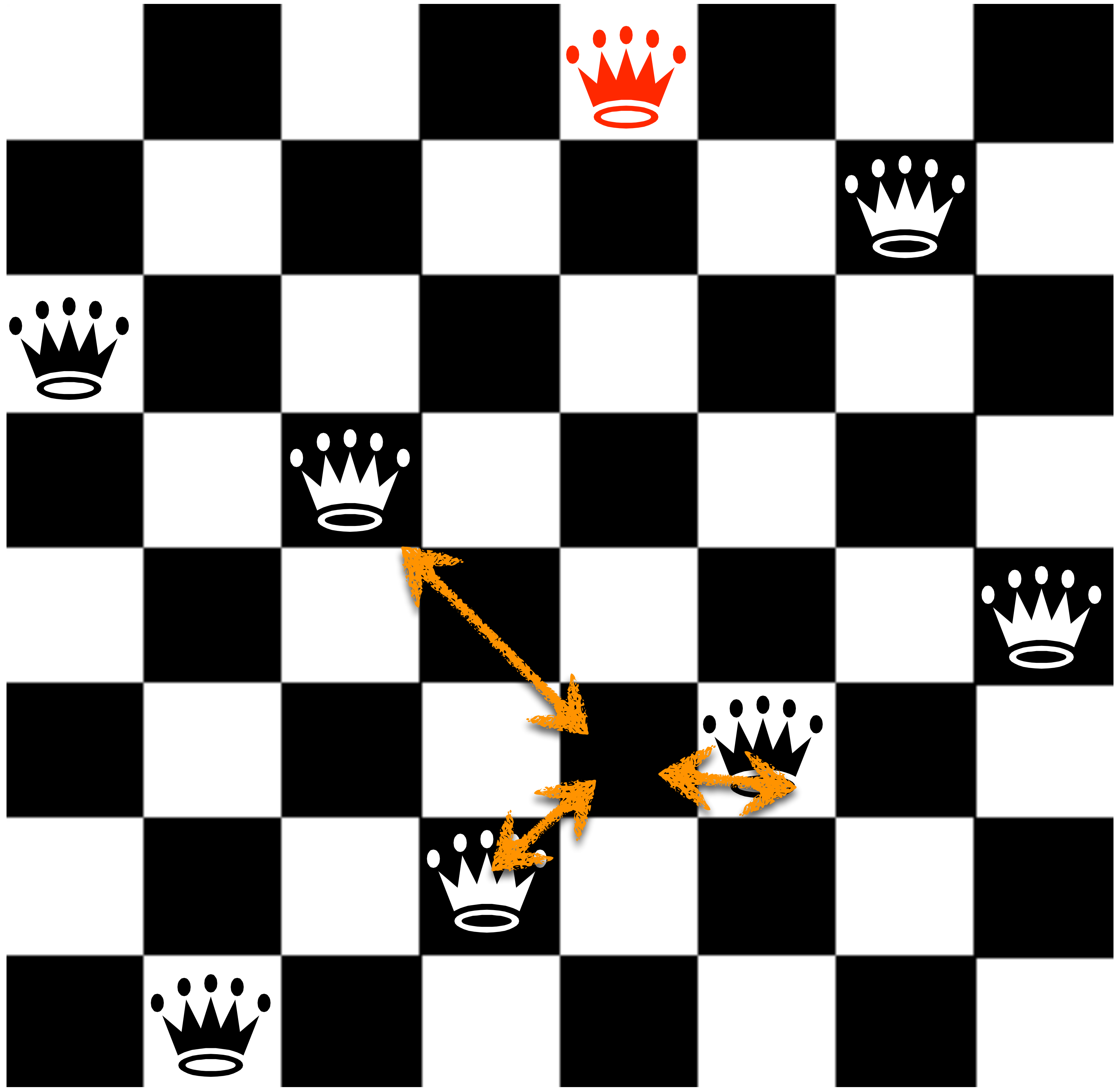


Two attacks: score -2

Eight Queens Problem



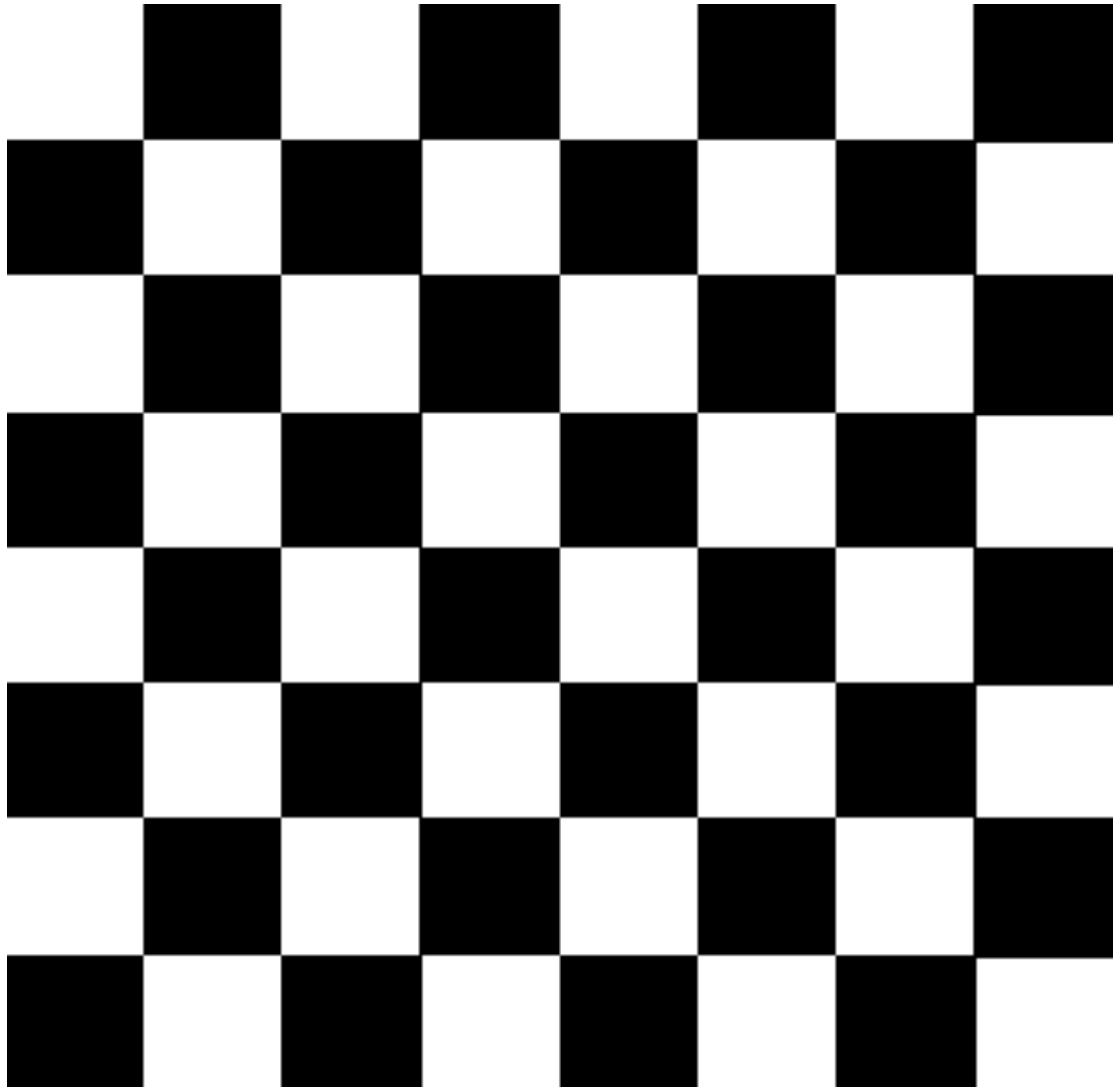
Eight Queens Problem



Three attacks: score -3

Two Approaches

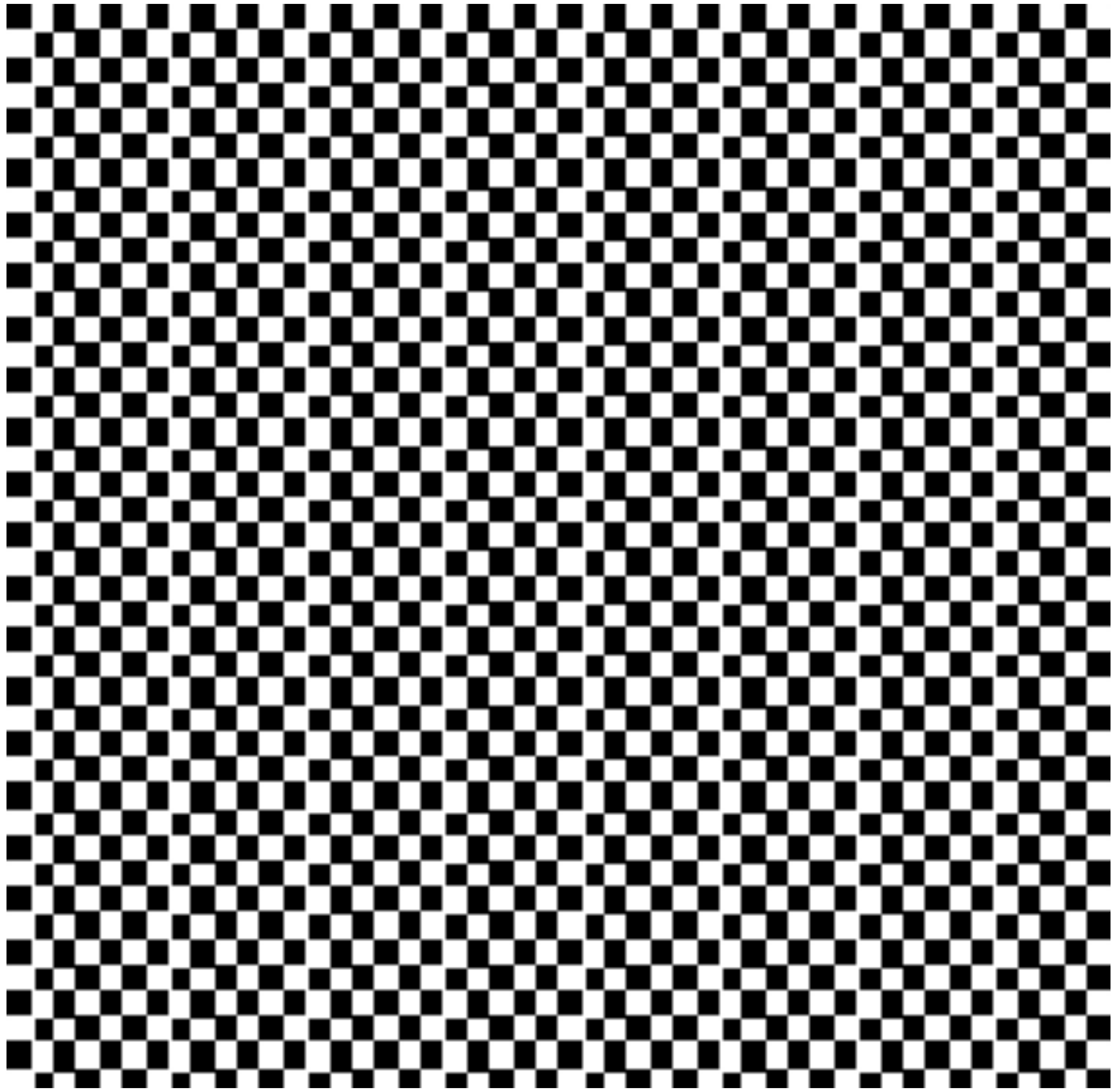
Build an algorithm that produces a solution to the problem by placing one piece at a time



Build an algorithm that compares two solutions to the problem; try different solutions, keep the better one, until you solve the problem

Does it scale?

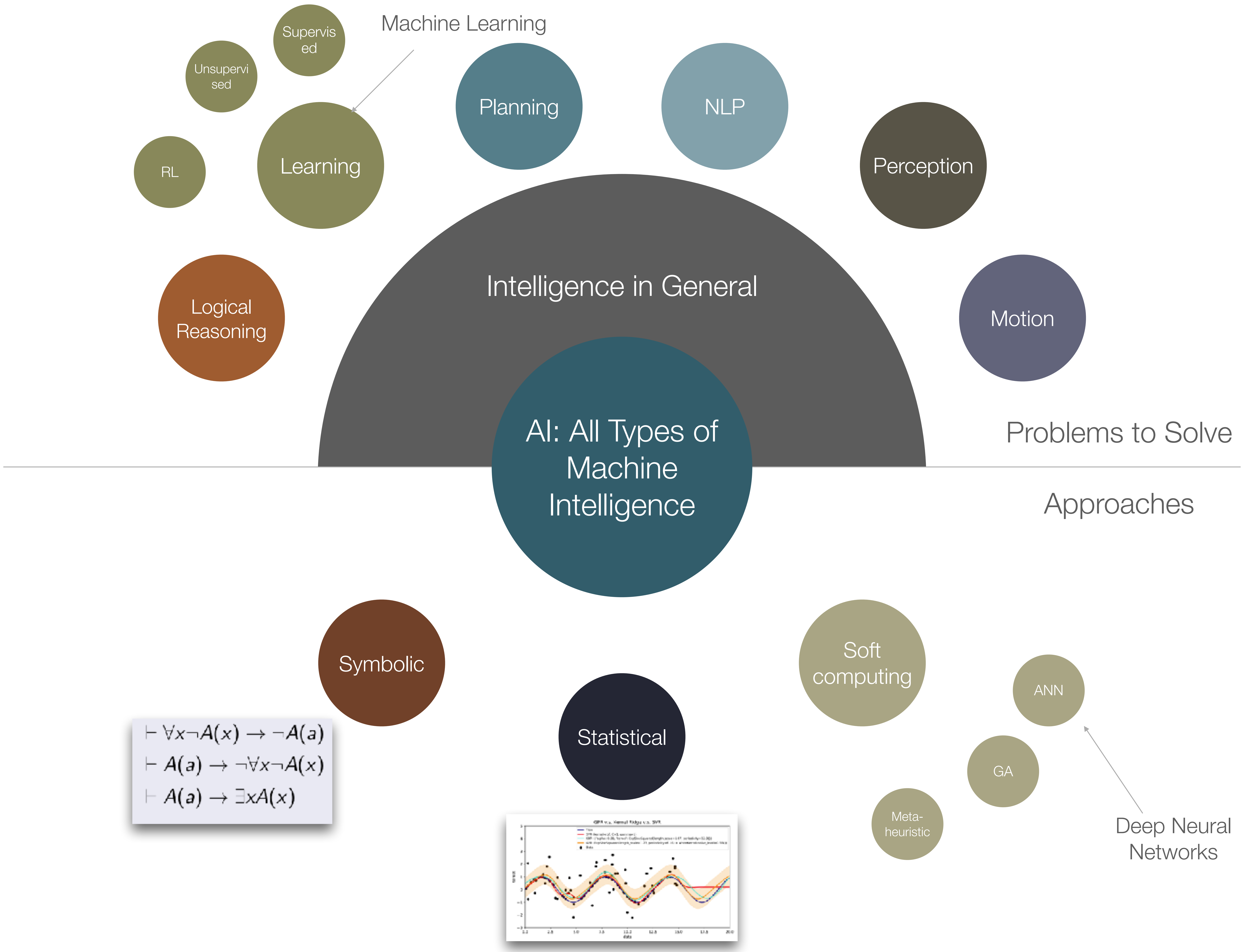
44 Queens Problem



Place 44 Queens on the board with no attack.

10^{12} Queens Problem

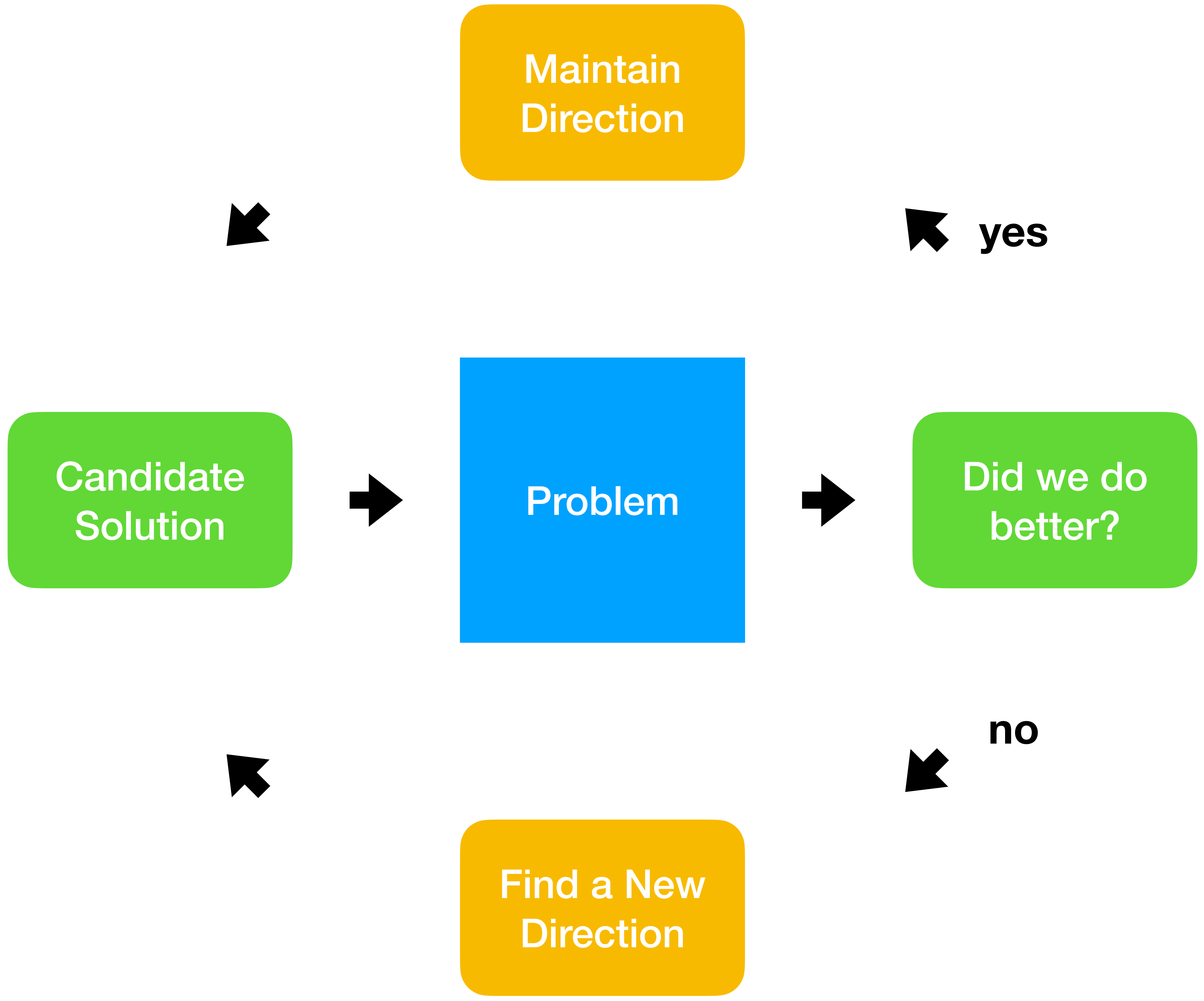
Place 10^{12} Queens on the board with no attack.


Machine Learning
Supervised
Unsupervised
RL
Learning
Planning
NLP
Perception
Logical Reasoning
Intelligence in General
Motion
Problems to Solve
AI: All Types of Machine Intelligence
Approaches
Symbolic
Statistical
Soft computing
ANN
GA
Meta-heuristic
Deep Neural Networks
⊢ ∀x¬A(x) → ¬A(a)
⊢ A(a) → ¬∀x¬A(x)
⊢ A(a) → ∃xA(x)

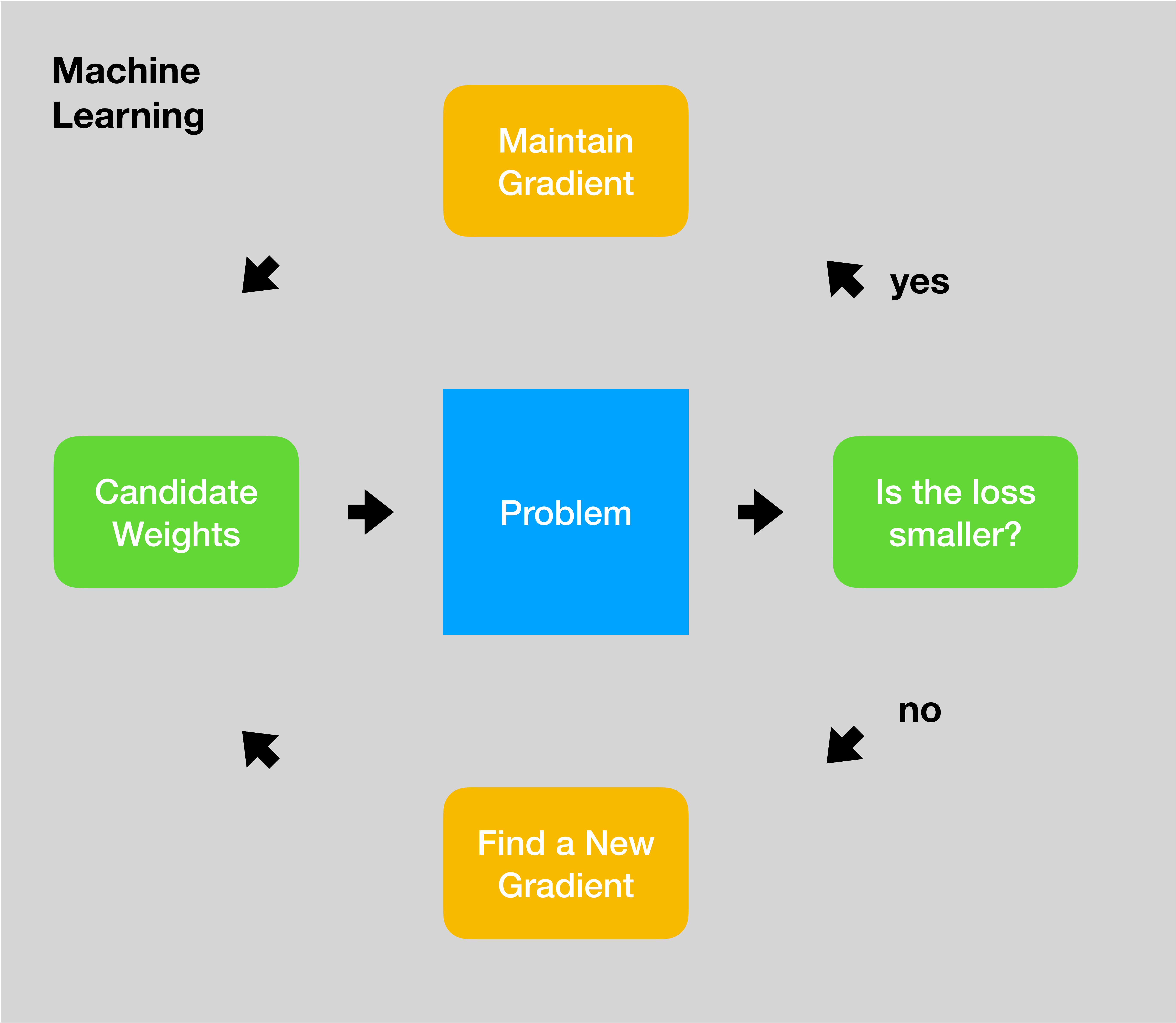
Trial and Error…

…as a general perspective on how to make machines smart

- Abundance of computational resources means many domains are adopting (knowingly or not) a similar approach.
  - Corpus-based NLP
  - Go (the only competitive AI players are based on Monte-Carlo Method)
  - Many application of machine learning



Maintain Direction
yes
Candidate Solution
Problem
Did we do better?
no
Find a New Direction

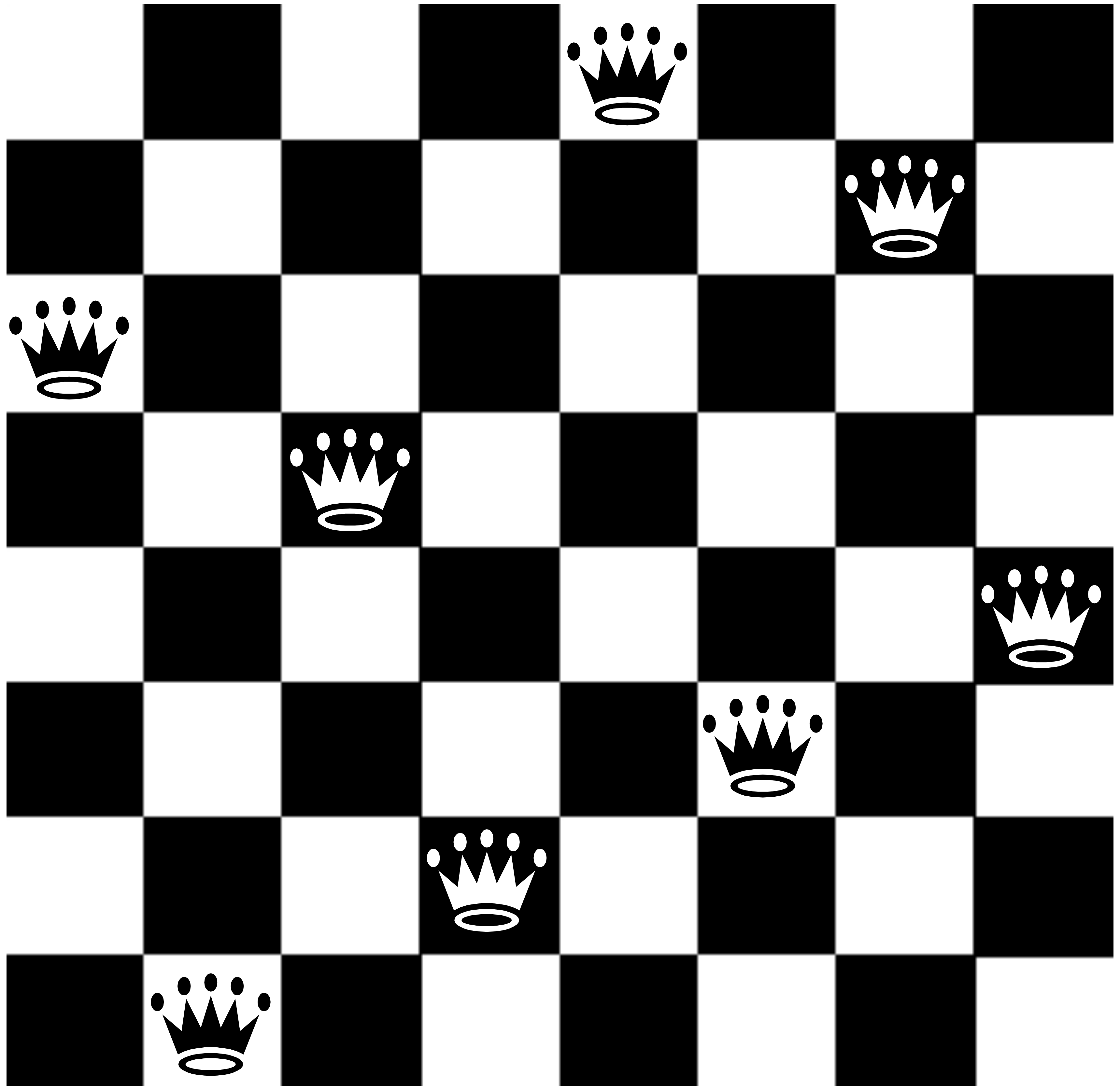

Machine Learning
Maintain Gradient
yes
Candidate Weights
Problem
Is the loss smaller?
no
Find a New Gradient

Key Ingredients of Metaheuristic Approaches

- **What** are we going to try this time? (representation)
- How is it **different** from what we tried before? (operators)
- **How well** did we do this time? (fitness/objective function)
- Minor (but critical) ingredients: constraints

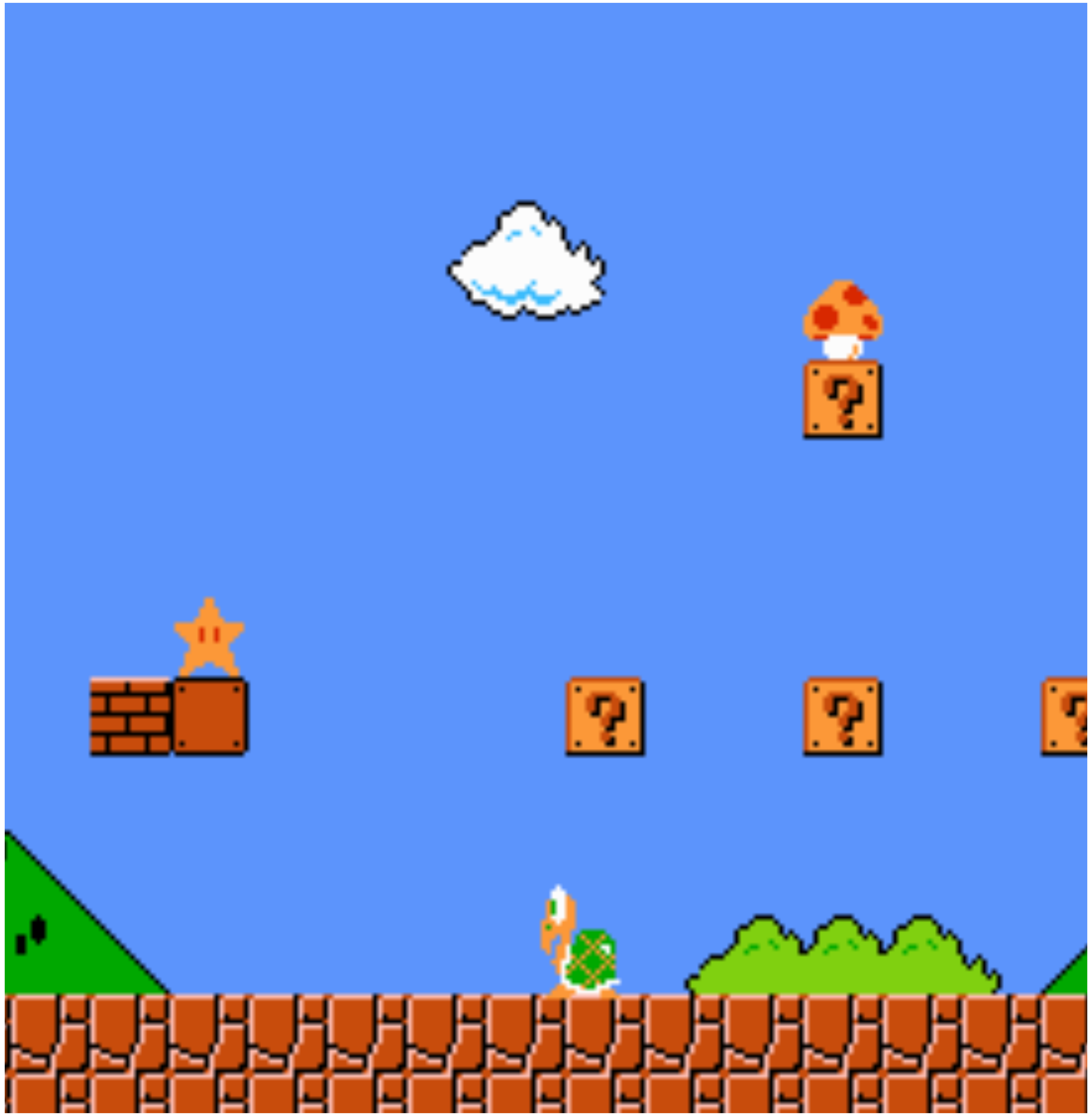
8 Queens Problem

- Representation: 8 by 8 matrix
- Operators: generate one valid board position from the current position, following the rule about Queen’s movement
- Fitness function: number of attacks (to be minimised)



Super Mario Bros.

- Representation: a list of `(button, start_time, end_time)`
- Operators: change `button` type in one tuple, increase/decrease `start_time` or `end_time`
- Fitness function: the distance you travelled without dying



Universal Viewpoint

- There are many algorithms in computational intelligence; you do need to learn individual algorithms in detail.
- However, I also want to communicate a frame of thinking, not only individual algorithms.
- The tuple of (representation, operators, fitness function) can be a universal platform to understand different classes of algorithms.
- We will revisit individual algorithms, using this tuples.

Design dictates solution

- Incorrect representation: what happens if we use `(button, pressed_time)` instead?
- Using wrong operators: what happens if we decrease/increase `start_time` and `end_time` by 5 seconds?
- Missing constraints: what happens if we swap the order of two tuples?
- Measuring the wrong fitness: what happens if we use the time elapsed until death? Or the final score?

Exploitation vs. Exploration

- Exploitation: if a candidate solution looks promising, optimisation should focus on that particular direction. However,

- Exploration: unexplored solution space may contain something *much better*.

- How to balance these two is critical to all learning/optimisation algorithms.

Machines are Dumb and Lazy

- Like human, they will do the minimum work that passes your criteria, i.e. design of the optimisation problem.
- Not because of their work ethic, but because of the fact that, usually, minimum work is the easiest to find solution.

Case Study: GenProg

- GenProg uses stochastic optimisation to modify existing faulty software code, until it passes all tests.
- We can only tell it to try until it passes all tests, not until the program is correct.

Anecdotes borrowed from Wes Weimer's SSBSE 2013 Keynote: https://www.cs.virginia.edu/~weimer/p/weimer-ssbse2013.pdf

Things GenProg Did…

- `nullhttpd`: test case for POST function failed; GenProg removed the entire functionality.
- `sort`: test required output to be sorted; GenProg's fix was to always output an empty set.
- Tests compared `output.txt` to `correct_output.txt`; GenProg deleted `correct_output.txt` and printed nothing.

Targets of our optimization

the concrete vs. the abstract

- Some objectives are concrete and tangible
  - E.g., value of input x that will take me to a specific branch in my code
- Some objectives are abstract and subjective
  - E.g., design of my system that is highly cohesive and low in coupling

What we mean and what we “really” mean

- Metric-based optimisation (where fitness equals an existing SE metric) is starting to be criticised. Compare the following two papers:
  - M. Harman and J. Clark. **Metrics are fitness functions too**. In 10th International Software Metrics Symposium (METRICS 2004), pages 58–69, Los Alamitos, California, USA, Sept. **2004**. IEEE Computer Society Press.
  - C. Simons, J. Singer, and D. R. White. **Search-based refactoring: Metrics are not enough**. In M. Barros and Y. Labiche, editors, Search-Based Software Engineering, volume 9275 of Lecture Notes in Computer Science, pages 47–61. Springer International Publishing, **2015**.

Semantic vs. Syntactic

- We are living in an interesting time, because there is a new technology in town that claims to “understand” human semantic better.
    - Well, not really “understand” - but interesting emergent behaviour nonetheless.
    - Always remember Chinese Room Experiment by John Searle

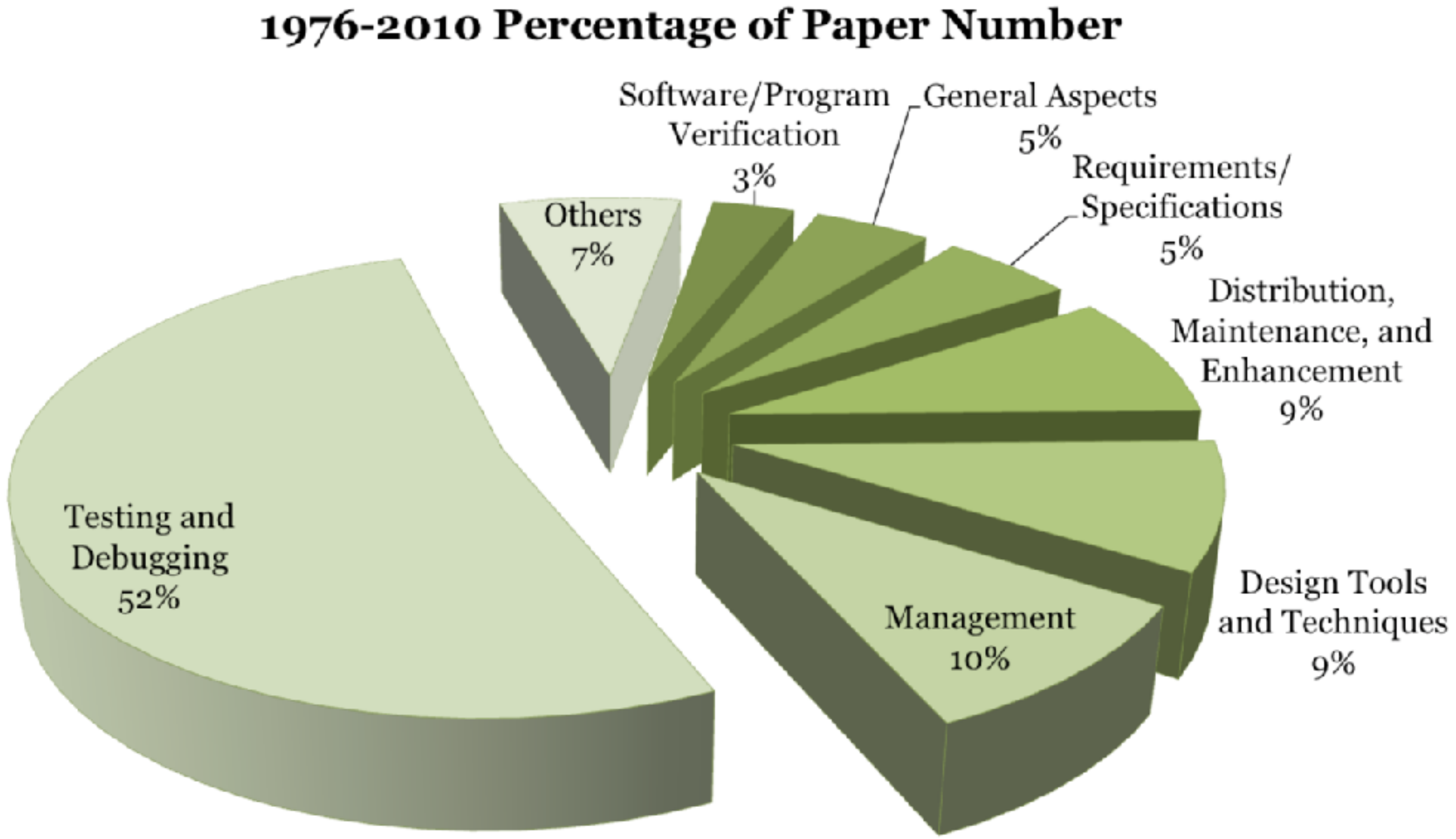
Expected Learning Outcome

- **Understand** basic metaheuristic algorithms; learn how to **implement** and **adapt** one to a given problem.
- **Embrace** metaheuristic optimisation as a valid tool for software engineers.
- Gain knowledge of the **literature**; learn **case studies** for various software development lifecycle stages.

A Sneak Peek Ahead

- What do people do with this metaheuristic in software engineering?
    - Given the definition of the task, basically anything for which you need a good-enough solution :)

Problem Domains



Structural Testing

- Take CS453 Automated Software Testing:)
- Intuitively: define the input constraints required to achieve structural coverage; solve the constraints using optimisation.
  - Symbolic execution + constraints solver
  - Dynamic analysis + metaheuristic optimisation
- Either way, huge advances in the last decade.
  - Clearly defined fitness function, industry demand (at least on achieving coverage)

Oracle Problem

- Coverage is not enough: “was the last execution **correct**?”
- Test oracle tells you whether the observed execution was correct or not
- Formal specification can serve as one; manual inspection by human can serve as one. But how do we automatically generate oracles?
- We want to test the code; we automatically generate test from the code; we want to check whether the test passed; we automatically generate test oracle from the co… wait a minute!
- This is a very hard problem; one which the state of the art does not know how to solve.

Testing non-functional properties

- Worst-Case Execution Time Analysis: strictly necessary for certain embedded systems (e.g. airbag controller), very hard to do statically; genetic algorithm has been very successful.
  - J. Wegener and M. Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. Real-Time Systems, 15(3):275 – 298, 1998.

Requirements Engineering

- Next Release Problem: given cost and benefit (expected revenue) for each features, what is the best subset of features to be released for budget B?
  - 0-1 Knapsack (NP-complete)
  - But release decisions are more political than NP-complete.
- Sensitivity Analysis: requirements data are usually estimates; which estimation will have the largest impact on the project, if it is off by X%?

Project Management & Planning

- Quantitatively simulate and measure the communication overhead (linear? logarithmic?)
- Robust planning: search for the tradeoff between overrun risk, project duration, and amount of overtime assignment

Design/architecture/refactoring

- Cluster software models to achieve certain structural properties (cohesion/coupling).
- Ironically, SBSE has also been used to analyse refactoring metrics: metric A and B both claim that they measure the same concept - optimising for A resulted in worse value of B, and vice versa :)

Genetic Improvement

- Given a source code, can we automatically improve its non-functional properties (such as speed)?
- Genetic Programming has been successfully applied to make genome-sequencing software 70 times faster. 70!
  - W. Langdon and M. Harman. Optimizing existing software with genetic programming. Transactions on Evolutionary Computation, 19(1):118–135, 2015.
- Evolve a specialised version of MiniSAT solver for problem classes.
  - J. Petke, M. Harman, W. B. Langdon, and W. Weimer. Using genetic improvement and code transplants to specialise a C++ program to a problem class. In proceedings of the 17th European Conference on Genetic Programming, volume 8599 of LNCS, pages 137–149. Springer, 2014.

Code Transplantation

- Software X has feature A, which you want to have in software Y. Can we automatically extract and transplant feature A from X to Y?
  - E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke. Automated software transplantation. In Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, pages 257–269.

Summary

- Key ingredients to metaheuristics: representation, operators, fitness function.
- Design dictates solutions - machines are dumb.
- Applications across all software development lifecycle activities, and beyond.

SBSE Repository

- http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/