Optimisation

CS454 AI-Based Software Engineering

Shin Yoo | COINSE@KAIST

Optimisation

- The problem of either maximising or minimising a real function by systematically choosing input values from within an allowed set (Wikipedia)
- The real function we optimise is sometimes called the objective function, or the fitness function.
- Perhaps even a loss function? :)
- In this course, we focus on metaheuristic optimisation and leave out the analytic, precise mathematical optimisation.
  - Not all of our problems will kindly provide gradients 🙃

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
- In general, we tend to consider a “heuristic" inferior to a precise algorithm. This is only defendable if there is an efficient precise algorithm :)

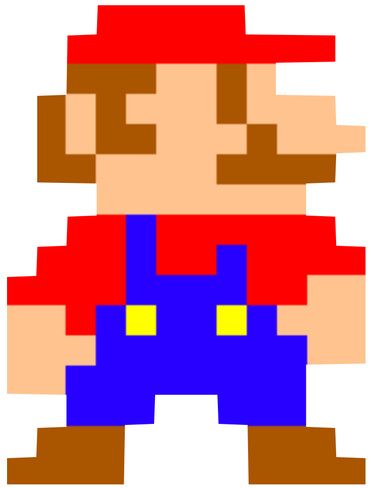
Teacher:
add numbers from 1 to 100!

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…)

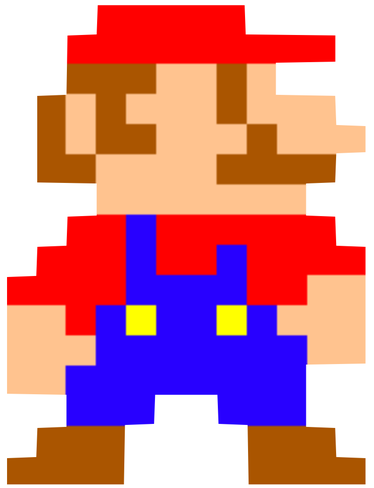
Player A

- 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

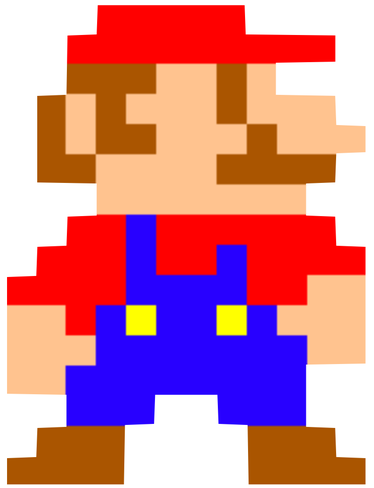
- Grab the controller.
- Play.
- Die.
- Repeat until level is cleared.



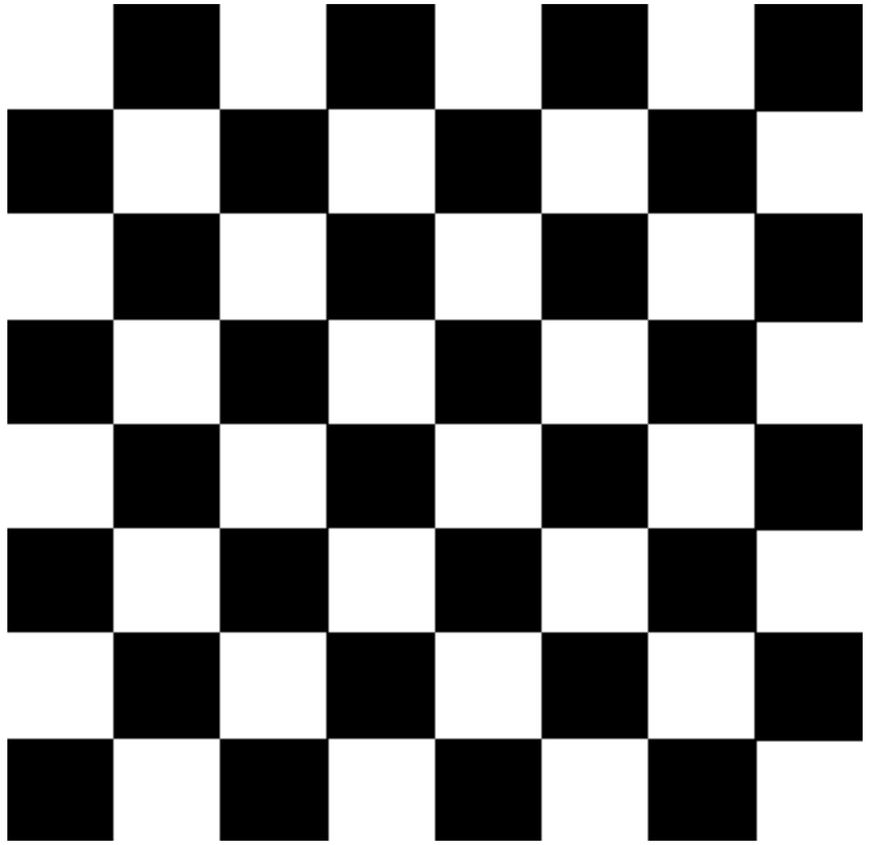
Intelligence lies in how differently you die next time.

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?”)

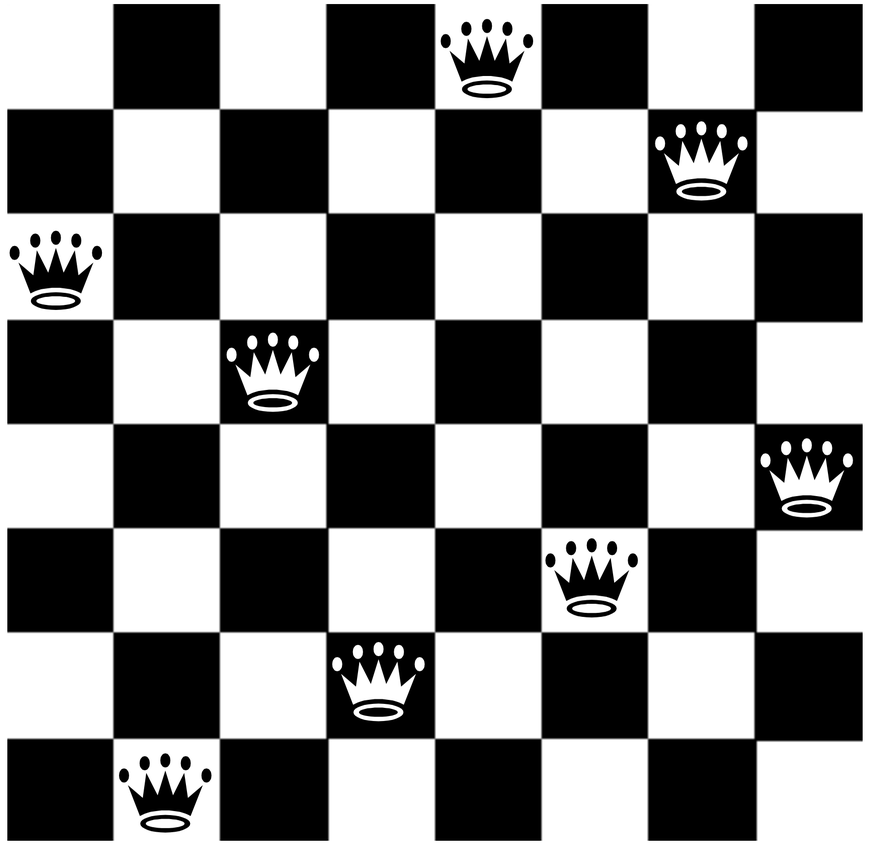


Eight Queens Problem



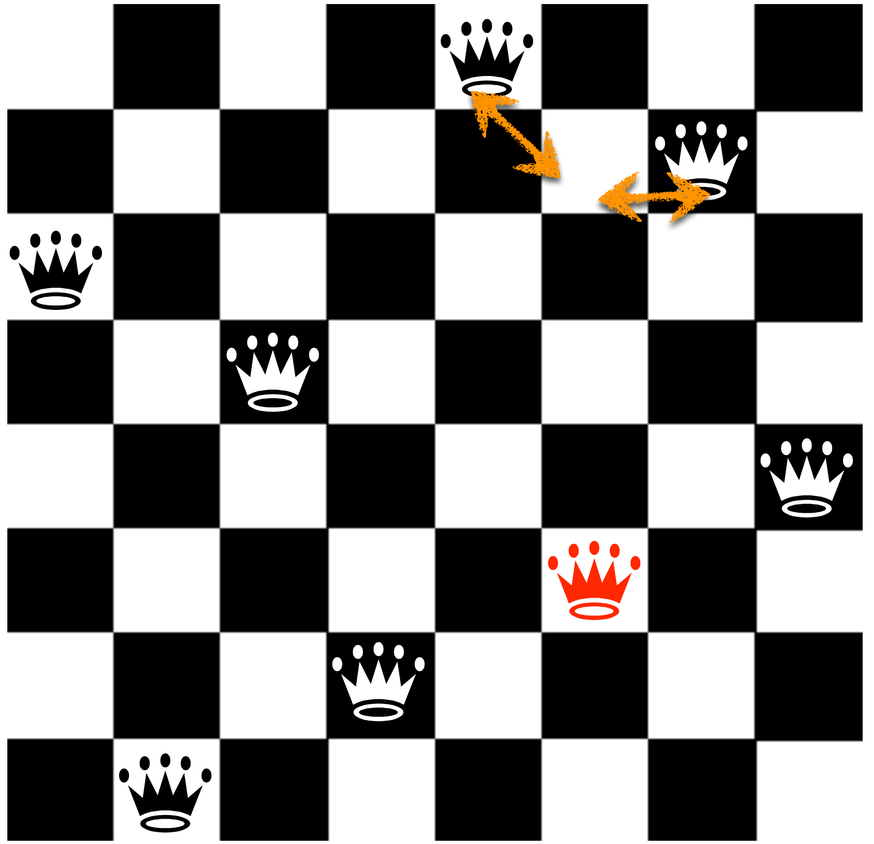
Place 8 queens on a chessboard so that no pair attacks each other.

Eight Queens Problem



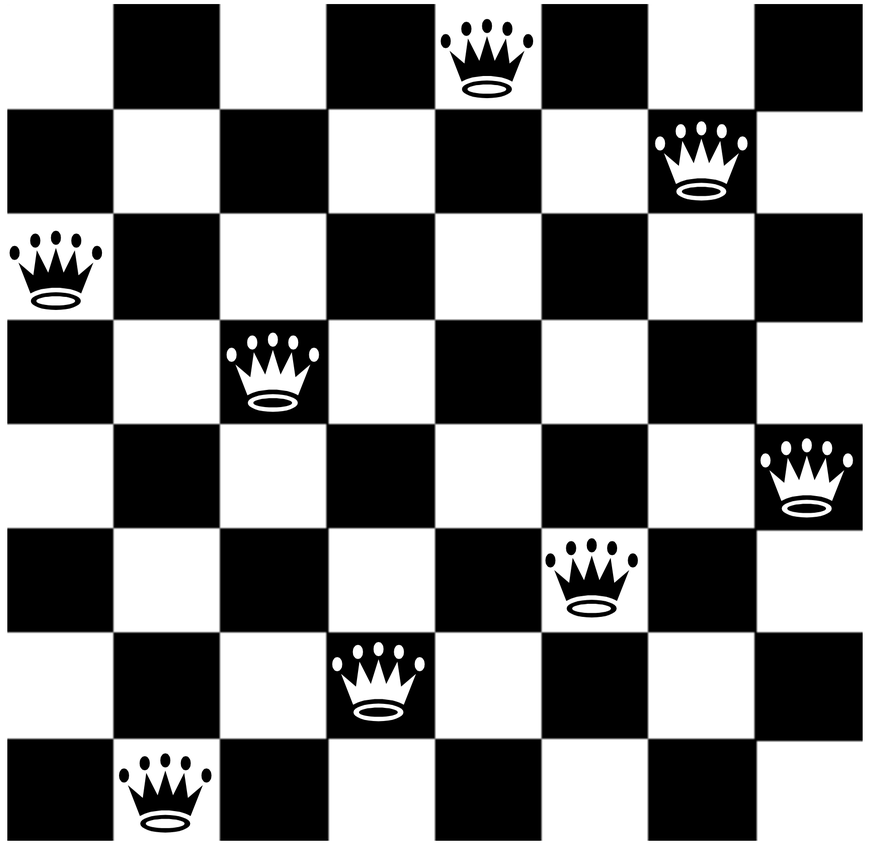
Perfect solution: score 0

Eight Queens Problem

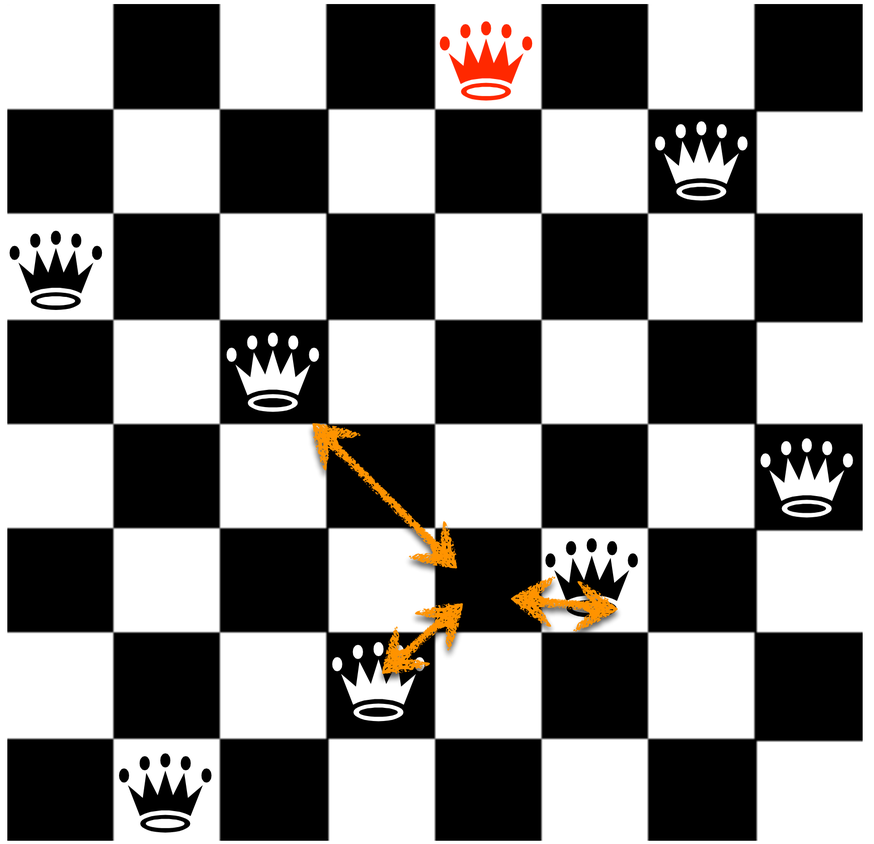


Two attacks: score -2

Eight Queens Problem



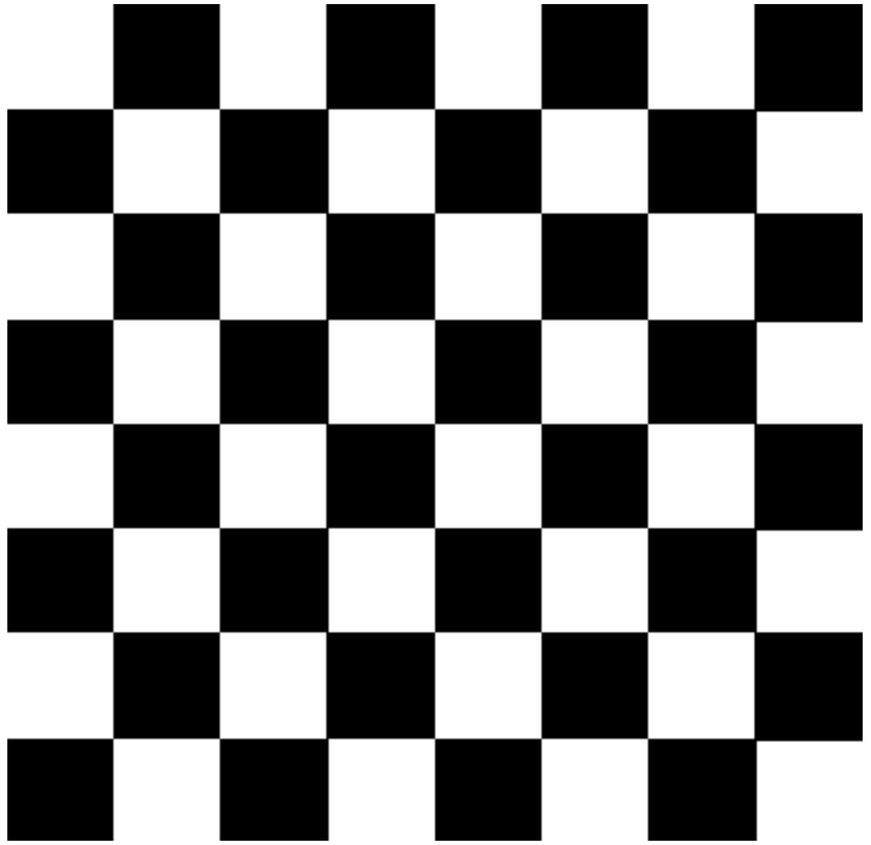
Eight Queens Problem



Three attacks: score -3

Two Approaches

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

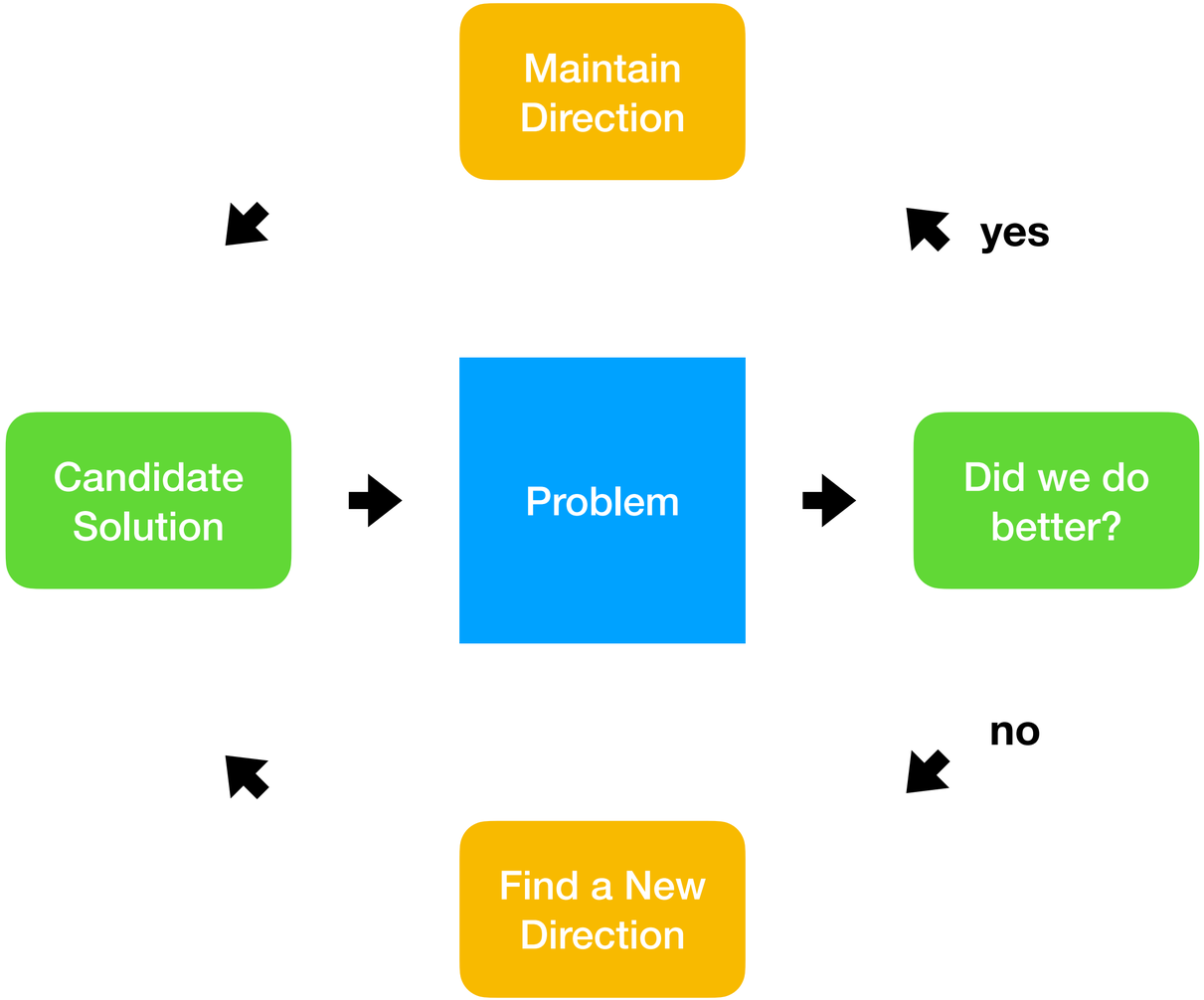


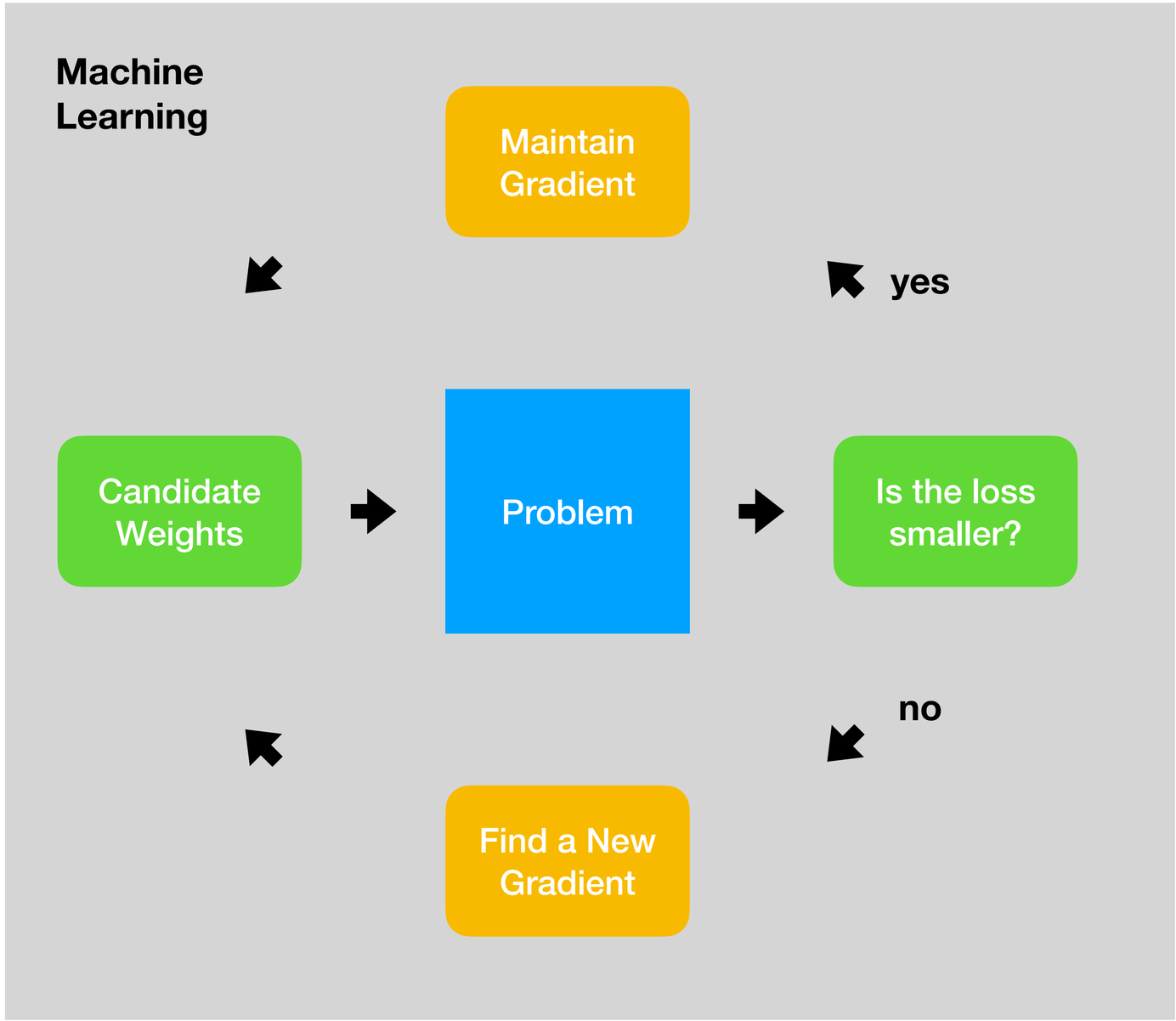
Build an algorithm that compares two solutions to the problem; try different solutions, keep the better one, until you solve the problem

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

Local vs. Global Optimisation

- Local Optimisation iteratively tries to get better **locally**.
  - We define a fitness landscape: similar solutions form neighbors, while the objective function.
  - Local optimisation maintains a current solution, and moves to one of its neighbours if it is better than the current solution
- Global Optimisation tries to sample better solution without being limited locally.
  - Some global optimisation algorithms maintain a group of solutions, thereby exploring the solution space globally
  - Others maintain a single current solution, but its movement is not bound by local moves.

Local Search Loop

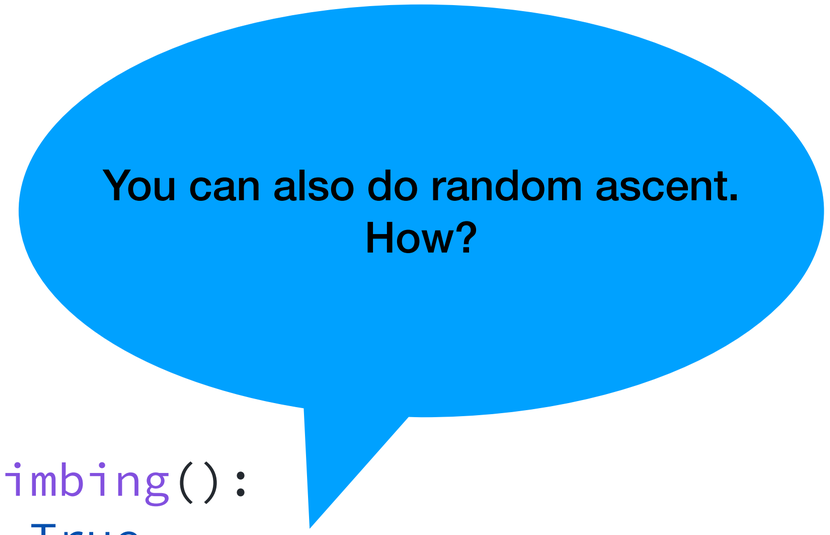
- Start with a single, random solution
- Consider the neighbouring solutions
- Move to one of the neighbours if better
- Repeat until no neighbour is better

Hill Climbing Algorithm

- This particular variation is also known as the steepest-ascent hill climbing.
  - Why? :)
  - What other versions are there?

```
def hill_climbing():
    climb = True
    s = get_random_solution()
    while climb:
        neighbours = get_neighbours(s)
        climb = False
        for n in neighbours:
            if fitness(n) > fitness(s):
                climb = True
                s = n
    return s
```

Hill Climbing Algorithm




```
def hill_climbing():
    climb = True
    s = get_random_solution()
    while climb:
        neighbours = get_neighbours(s)
        climb = False
        for n in neighbours:
            if fitness(n) > fitness(s):
                climb = True
                s = n
    return s
```

Steepest Ascent

```
def hill_climbing():
    climb = True
    s = get_random_solution()
    while climb:
        neighbours = get_neighbours(s)
        climb = False
        for n in neighbours:
            if fitness(n) > fitness(s):
                climb = True
                s = n
                break
    return s
```

First Ascent

Ascent Strategy

- Not possible to know which one is better.
- IF the current solution is near a local optimum, slowing down the ascent may (or may not) be better.
- Regardless of strategy, hill climbing monotonically climbs, until it reaches local/global optimum; it never goes down.

Pros/Cons

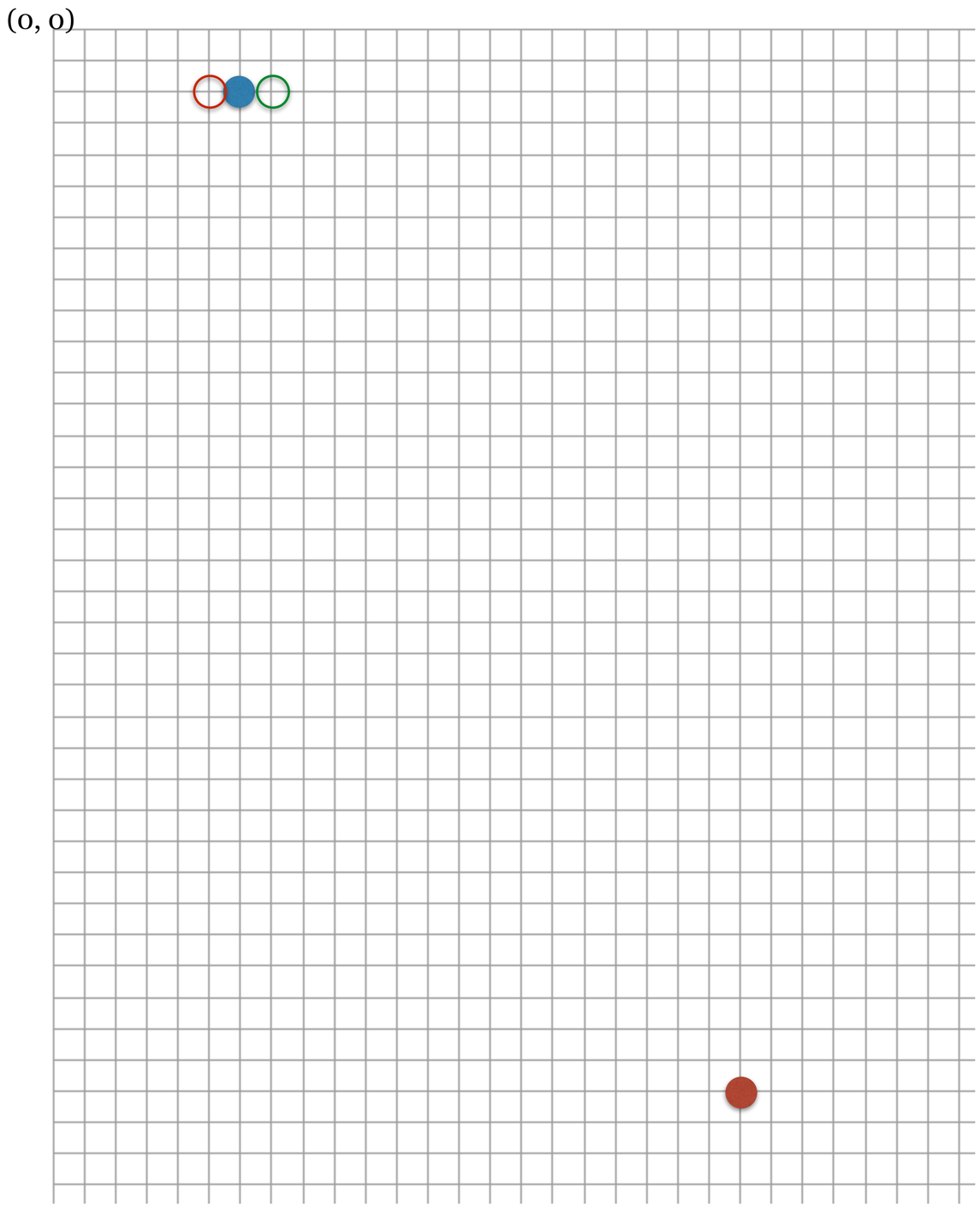
- Pros:cheap (fewer fitness evaluations compared to GAs), easy to implement, repeated applications can give insights into the landscape, suitable for solutions that need to be built through small incremental changes
- Cons: more likely to get stuck in local optima, unable to escape local optima

Alternating Variable Method (AVM)

- A type of Pattern Search: searches for an input vector that can maximise/ minimise a given objective function
- It has two operation modes: exploratory move, and pattern move.
  - For each variable:
    - Use exploratory move to decide which direction results in fitter solutions
    - Use pattern move to accelerate to that direction

Alternating Variable Method

- Based on the known empirical results, AVM is one of the most effective algorithm for achieving C/C++ structural coverage
  - M. Harman and P. McMinn. A theoretical and empirical analysis of evolutionary testing and hill climbing for structural test data generation. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2007), pages pp. 73–83. ACM Press, July 2007.
  - M. Harman and P. McMinn. A theoretical and empirical study of search based testing: Local, global and hybrid search. IEEE Transactions on Software Engineering, 36(2):226–247, 2010.



AVM: Exploratory Move

Starting from(6, 2), we want to search for the red dot at (22, 34). We can measure the distance to the goal.

First we try exploratory move for x: make the smallest change, and see which direction results in reduced distance. The initial distance is 35.77.

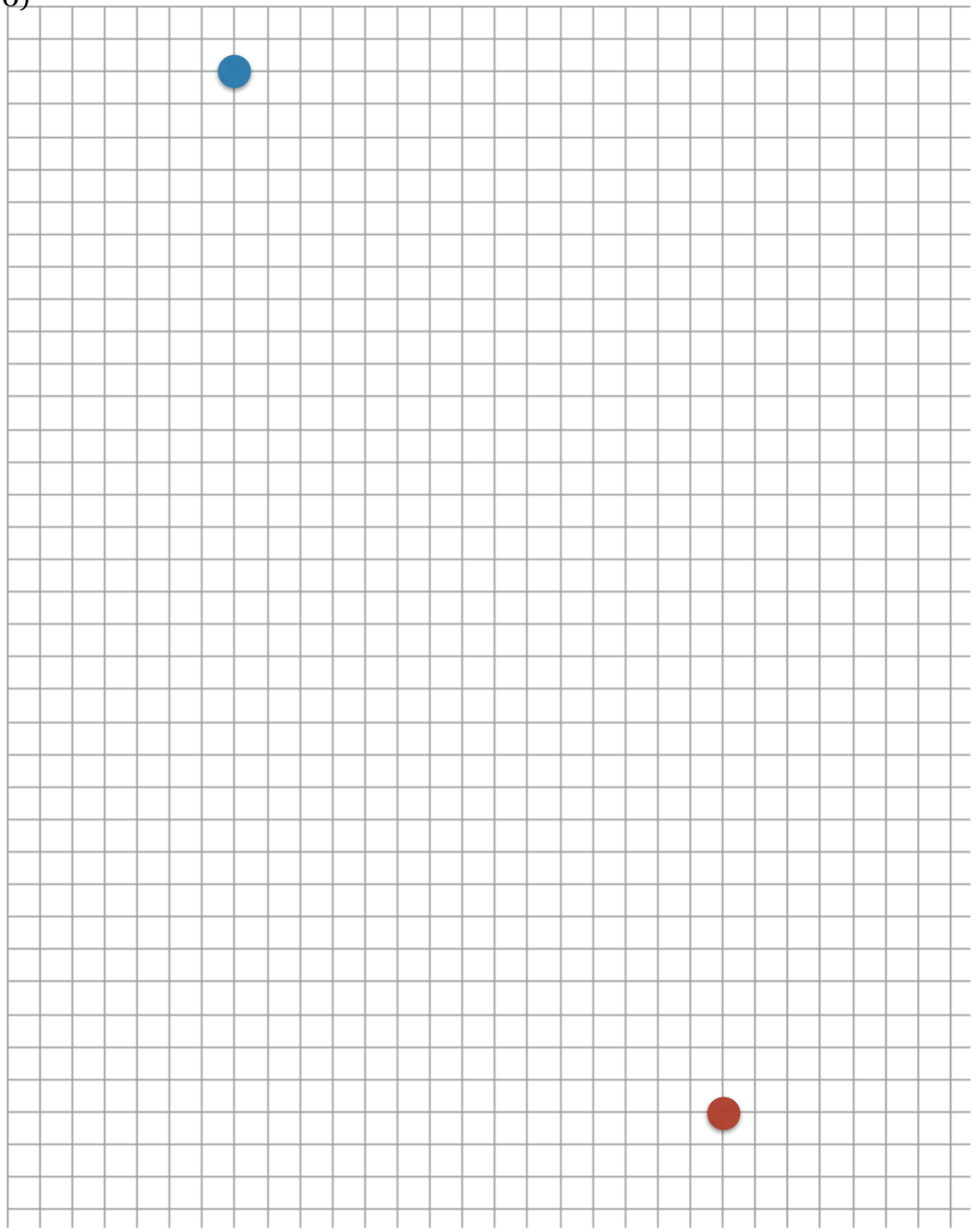
-1: (5, 2) **Increased** (36.23). X

+1: (7, 2) **Decreased** (35.34) O

Consequently, x needs to be

increased at the moment.

(0, 0)



AVM: Pattern Move

Now that we decided to increase x, try doubling the difference as long as the distance continues to decrease. At the beginning of the pattern move, x is equal to 7.

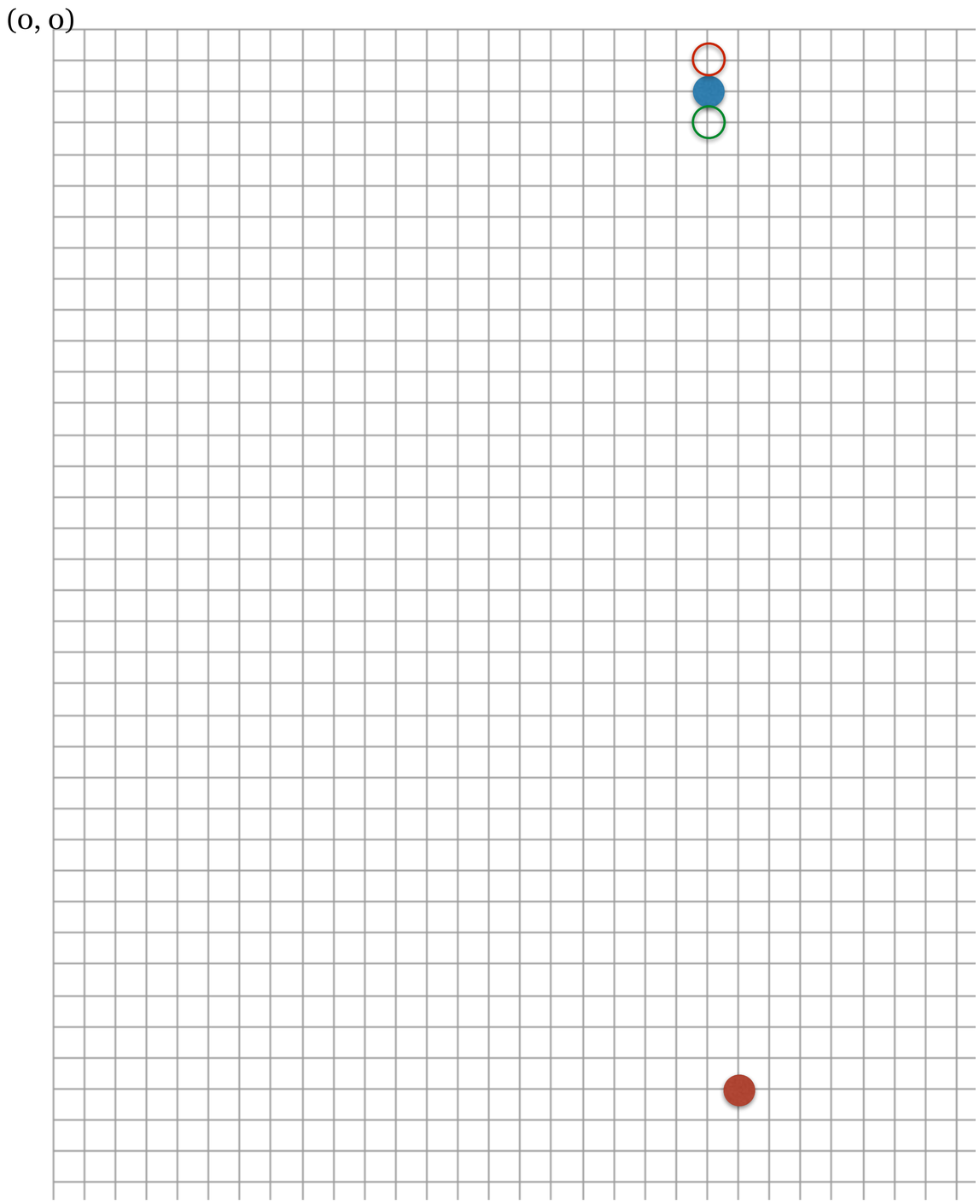
x = 9 (Δx=2): **decrease** (34.53)

x = 13 (Δx=4): **decrease** (33.24)

x = 21 (Δx=8): **decrease** (32.01)

x = 37?(Δx=16): **increase** (35.34)

With increment of 16, the distance starts to grow: this is called overshooting. In this case, we cancel the last pattern move, and start the exploratory move for the next variable, y.



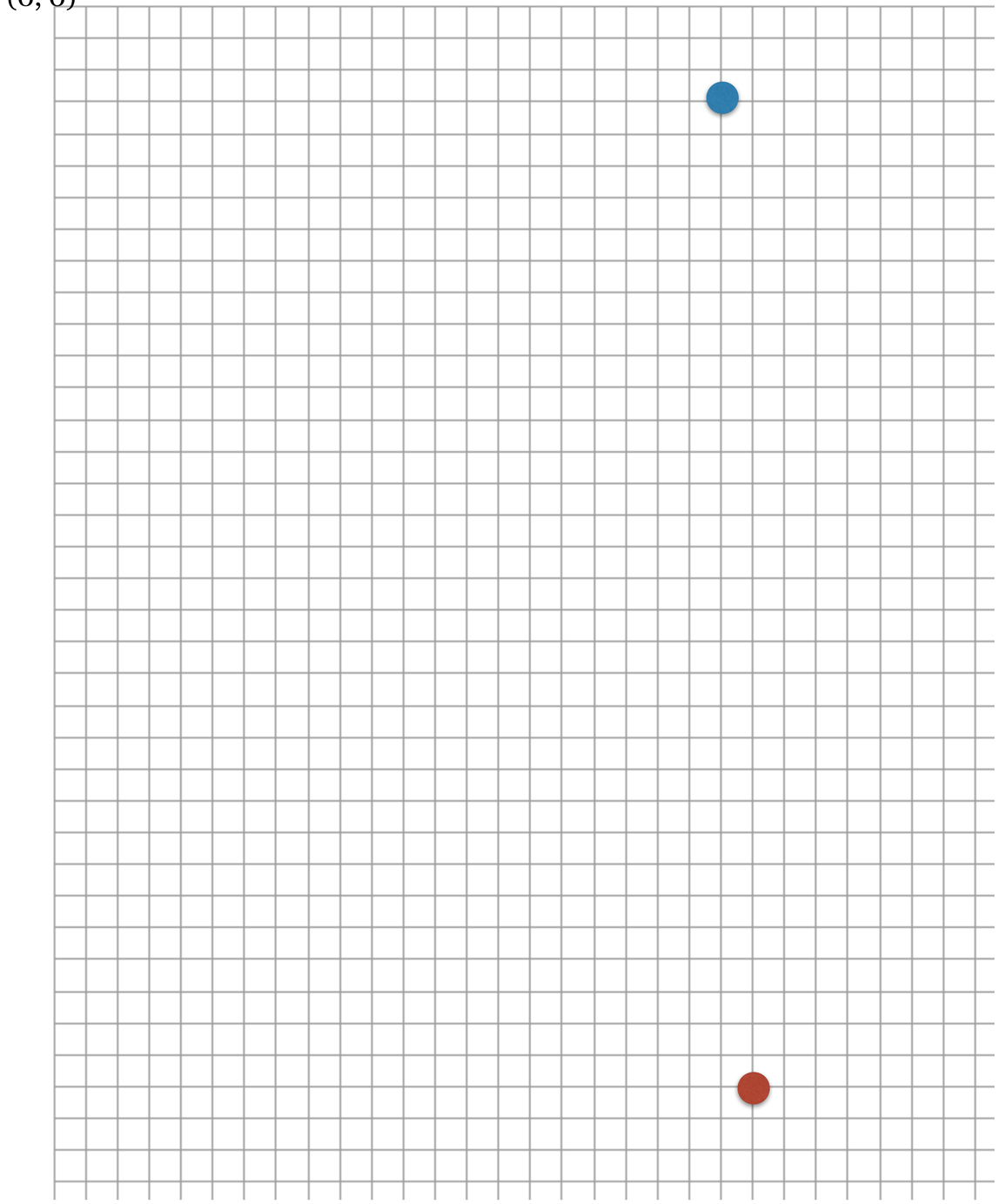
AVM: Exploratory Move

We now change y by 1 and decide the direction. The distance from the last location, (21, 2), is 32.01.

-1: (21, 1) **increase** (33.01). X

+1: (21, 3) **decrease** (31.01) O

So y needs to be increased.



AVM: Pattern Move

We increase the variable y with pattern moves now. Initially y is 3.

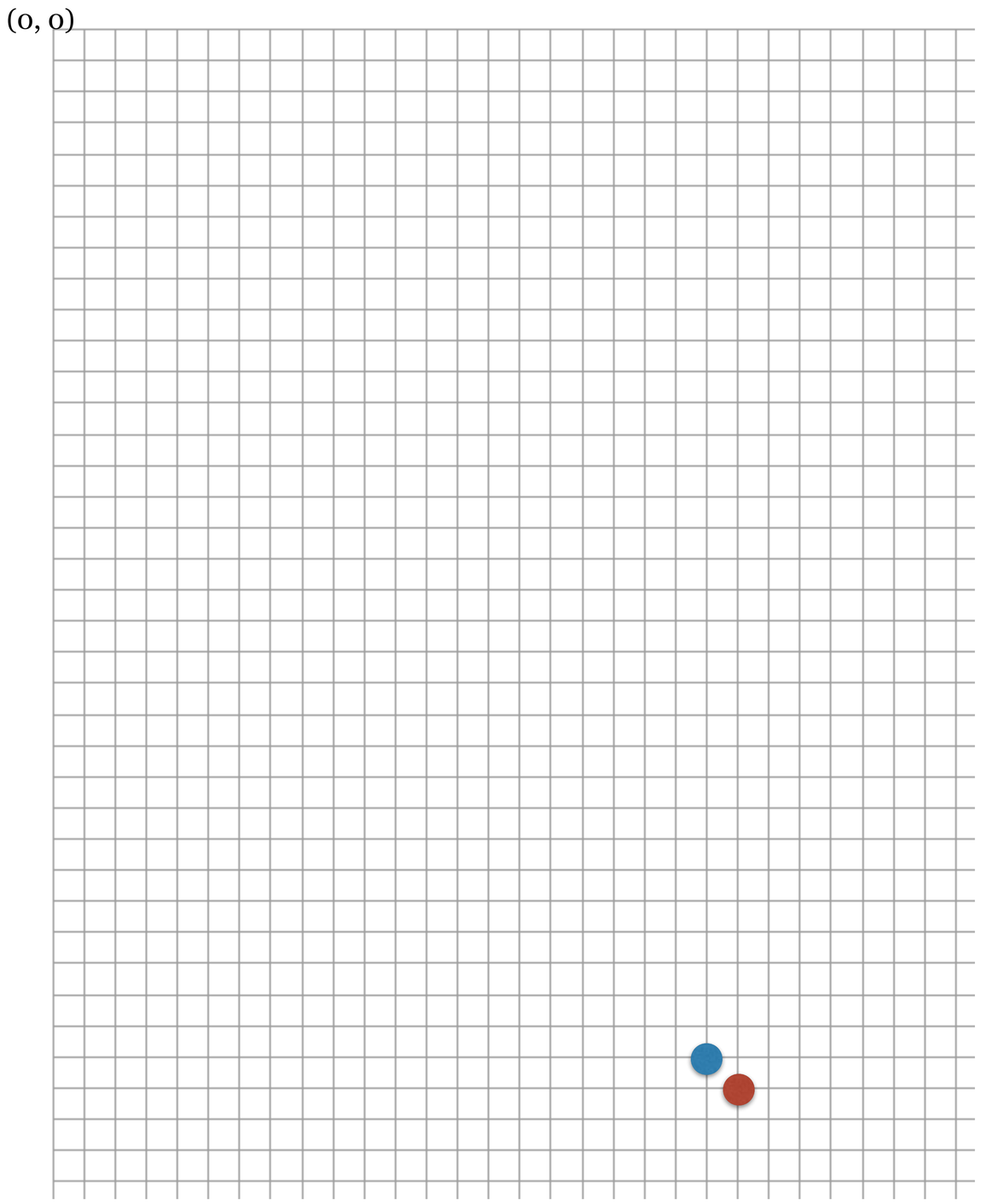
y = 5 (Δy=2): **decrease** (29.01)

y = 9 (Δy=4): **decrease** (25.01)

y = 17 (Δy=8): **decrease** (17.02)

y = 33(Δy=16): **decrease** (1.41)

y = 65(Δy=32): **Overshooting!**



AVM: Exploratory Move

After overshooting of y, we start the exploratory move for x. We decide to increase, but as soon as we try +2, it overshoots. After cancellation of this, we have the correct x.

After one more exploratory move for y, we reach the goal.

Alternating Variable Method

- For a reference implementation and basic applications, see: http://avmframework.org

- P. McMinn and G. M. Kapfhammer. AVMf: An open-source framework and implementation of the alternating variable method. In International Symposium on Search-Based Software Engineering (SSBSE 2016), volume 9962 of Lecture Notes in Computer Science, pages 259–266. Springer, 2016.

Simulated Annealing

A local-ish global optimisation :)

- Big question: how do we escape local optima, if we are in one?
  - Thought 1: we never know whether we are climbing a local or a global optimum!
  - Thought 2: assuming that there are more local than global optima, it makes sense to escape.
  - Thought 3: but not always - when we stop, we want to stop near the top of SOME optimum.

Annealing (풀림)

- Keep metal in a very high temperature for a long time, and then slowly cool down: it then becomes more workable.

- At high temperature, atoms are released from internal stress by the energy; during the cool-down, they form new nucleates without any strain, becoming softer.

Simulated Annealing

- Introduce “temperature” into local search: start with a high temperature, and slowly cool down.
  - When the temperature is high, the solution (like atom) is unstable and can make random moves (i.e. escapes).
  - As the temperature decreases, the energy level gradually gets lower, and escapes become more infrequent.

Simulated Annealing

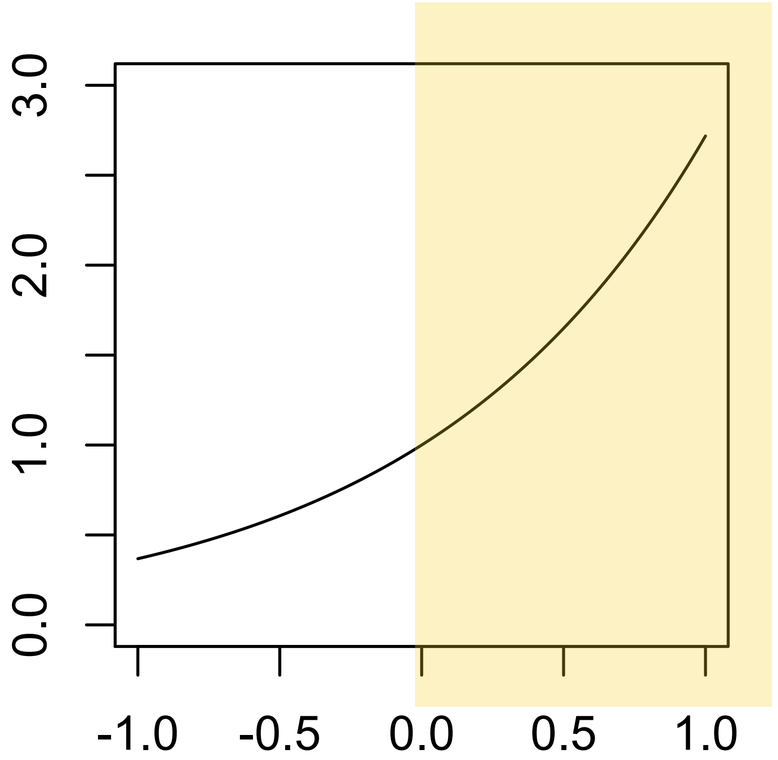
```
SimulatedAnnealing()
(1)      s ← s_0
(2)      T ← T_0
(3)      for k = 0 to n
(4)          s_new ← GetRandomNeighbour(s)
(5)          if P(F(s), F(s_new), T) ≥ random(0, 1)
(6)              s ← s_new
(7)          T ← Cool(T)
(8)      return s
```

```
P(F(s),F(s_new), T)
(1)      if F(s_new) > F(s) then return 1.0
(2)                         else return e^((F(s_new)−F(s))/T)
```

Acceptance Probability

- Borrowed from metallurgy
- When new solution is better ($F(s_{new}) > F(s)$), always accept ($P > 1$)
- When new solution is equally good, accept
- When new solution is worse:
  - more likely to accept small downhill movement
  - gets smaller as temperature drops

$$P = e^{\frac{F(s_{new}) - F(s)}{T}}$$



Cooling Schedule

Temperature at time step t as a function of t:

- Linear: $T(t) = T_0 - \alpha t$
- Exponential: $T(t) = T_0 \alpha^t (0 < \alpha < 1)$
- Logarithmic $T(t) = \frac{c}{\log(t+d)}$

- With large c, this can be very slow cooling
- There is an existence proof that says logarithmic will find the global optimum in infinite time… huh?
- It becomes essentially a random search
- Theoretically interesting, but practically not so much.

Tabu Search

Fred Glover, 1986

- Another attempt to escape local optima
- Two exceptions to local search:
  - It is possible to accept a worse move
  - Remember “visited” solutions and avoid coming back

Tabu Search

```
TabuSearch()
(1)     s ← s_0
(2)     s_best ← s
(3)     T ← [] // tabu list
(4)     while not stoppingCondition()
(5)         c_best ← null
(6)         foreach c ∈ GetNeighbours(s)
(7)             if (c ∉ T) ∧ (F(c) > F(c_best)) then c_best ← c
(8)         s ← c_best
(9)         if F(c_best) > F(s_best) then s_best ← c_best
(10)        append(T, c_best)
(11)        if |T| > maxTabuSize then removeAt(T, 0)
(12)    return sBest
```

Tabu list is a FIFO queue: with the `maxTabuSize` we can control the memory span of the search.

Minimum Viable Ingredients…

…that enables you to do optimisation?

- Some formal representation of your candidate solution (this defines your solution/search space)
- Some way to transform one or more candidate solution(s) to another, i.e., definition of neighbourhood, or genetic operations in evolutionary computation (we will see about this soon)
- Some measure of how good your current solution is, i.e., the objective function
- Essentially, trial and error combined with some state management :)
- Such a simple, yet fundamental idea - it keeps popping up all the time!

Loop Engineering 😎

https://www.ibm.com/think/topics/loop-engineering

- “Loop engineering is the practice of designing agentic workflows, or loops, that iteratively guide AI agents toward completing user-defined goals with minimal human intervention.”
  - Solution representation and transformation are fundamentally new and powerful with LLM-driven agents: they can handle long, complicated, open-ended solutions in both natural/programming language, even using other subagents
  - Objective function is what the cool kids nowadays call “evals” :)
  - And instead of repeatedly prompting, you are to write an optimisation loop.

Loop Engineering 😎

Algorithm vs. Agentic Workflow

- With agentic workflow, we gain powerful, intelligent tools that allow you flexible search over only vaguely defined space 👍
- However, we lose the tight control over what is allowed and what is not. For example, tabu search would be useful (i.e., forbidden any action that was not successful before) but also hard to strictly enforce (i.e., LLMs will forget and do it again anyway)

Summary

- Meta-heuristic optimisation aims to maximise/minimise a given objective function, even when there is no mathematical model that yields gradients.
- Trial and error combined with state management (=memory) —> basic optimisation loop!
- Local search focuses on exploitation, i.e., looking at a single specific solution and gradually improving it by analysing the nearby solutions. Its weakness is the lack of exploration, i.e., it will not go beyond the immediate neighbourhood, often getting stuck in local optima.