# Evolutionary Computation

## 2025/26

## Master Artificial Intelligence

Arno Formella

Departamento de Informática
Escola Superior de Enxeñaría Informática
Universidade de Vigo

25/26

# course syllabus (theory)

initial planning for the theory lessons, let's see how it evolves:

| **Wednesdays** | **content** |
|---|---|
| 27/01/26 | basics |
| 03/02/26 | genetic algorithm |
| 10/02/26 | particle swarm optimization |
| 24/02/26 | other evolutionary algorithms |
| 03/03/26 | meta-heuristics and parameter tuning |
| 10/03/26 | multi-objective optimization |
| 17/03/26 | advanced topics |

# course syllabus (laboratory)

initial planning for the lab sessions, let's see how it evolves:

| **Tuesdays** | **content** |
|---|---|
| 28/01/26 | function minimization with genetic algorithm |
| 04/02/26 | TSP with genetic algorithm |
| 11/02/26 | |
| 25/02/26 | |
| 04/03/26 | |
| 11/03/26 | |
| 18/03/26 | |

There will be two major up-loads of homework exercises.

# evaluation/assessment

- active participation in class (10%)
- two major up-loads of homework (50%)
  (probably week 09-13 Feb, and week 17-20 Mar)
- final exam (40%)
  (8th April (16h) and/or 2nd June (16h))

# bibliography

- D. Simon, *Evolutionary Optimization Algorithms*, ISBN: 978-0-470-93741-9, Wiley, 2013
- A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing*, second edition, ISBN 978-3-662-44873-1, Springer, 2015
- my accompanying webpage:

  http://formella.webs.uvigo.es/doc/ec25/index.html
- more references and webpages in upcoming slides and in the lab instruction sheets
- Wikipedia:

  https://en.wikipedia.org/wiki/Evolutionary_computation

  https://en.wikipedia.org/wiki/Evolutionary_algorithm

# evolutionary computation

- Evolutionary computation is about optimization.
- Evolutionary algorithms are usually randomized or probabilistic heuristic algorithms.
- Many of them are called nature-inspired (or bio-inspired) algorithms as they exhibit similar properties as observed in nature (especially, but not only, from biology).
- Programs based on evolutionary algorithms are typically used to find approximate solutions (but still somewhat good solutions) to difficult problems.

Disclaimer: we are talking about *nature-inspired optimization algorithms*, we are not copying nature, mostly, because I've no idea what nature is doing. We are interested in mathematical models and certain types of algorithms that serve as powerful optimization tools.

scientific reports



OPEN

# The cheetah optimizer: a nature-inspired metaheuristic algorithm for large-scale optimization problems

Mohammad Amin Akbari[1], Mohsen Zare[2], Rasoul Azizipanah-abarghooee[3], Seyedali Mirjalili[4,5] & Mohamed Deriche[1]✉

Motivated by the hunting strategies of cheetahs, this paper proposes a nature-inspired algorithm called the cheetah optimizer (CO). Cheetahs generally utilize three main strategies for hunting prey, i.e., searching, sitting-and-waiting, and attacking. These strategies are adopted in this work. Additionally, the leave the pray and go back home strategy is also incorporated in the hunting process to improve the proposed framework's population diversification, convergence performance, and robustness. We perform intensive testing over 14 shifted-rotated CEC-2005 benchmark functions to evaluate the performance of the proposed CO in comparison to state-of-the-art algorithms. Moreover, to test the power of the proposed CO algorithm over large-scale optimization problems, the CEC2010 and the CEC2013 benchmarks are considered. The proposed algorithm is also tested in solving one of

scientific reports



OPEN

# A new human-inspired metaheuristic algorithm for solving optimization problems based on mimicking sewing training

Mohammad Dehghani, Eva Trojovská ✉ & Tomáš Zuščák

This paper introduces a new human-based metaheuristic algorithm called Sewing Training-Based Optimization (STBO), which has applications in handling optimization tasks. The fundamental inspiration of STBO is teaching the process of sewing to beginner tailors. The theory of the proposed STBO approach is described and then mathematically modeled in three phases: (i) training, (ii) imitation of the instructor's skills, and (iii) practice. STBO performance is evaluated on fifty-two benchmark functions consisting of unimodal, high-dimensional multimodal, fixed-dimensional multimodal, and the CEC 2017 test suite. The optimization results show that STBO, with its high

scientific reports



OPEN

# A novel hermit crab optimization algorithm

Jia Guo[1,2], Guoyuan Zhou[1], Ke Yan[3], Binghua Shi[1]✉, Yi Di[1,2] & Yuji Sato[4]

High-dimensional optimization has numerous potential applications in both academia and industry. It is a major challenge for optimization algorithms to generate very accurate solutions in high-dimensional search spaces. However, traditional search tools are prone to dimensional catastrophes and local optima, thus failing to provide high-precision results. To solve these problems, a novel hermit crab optimization algorithm (the HCOA) is introduced in this paper. Inspired by the group behaviour of hermit crabs, the HCOA combines the optimal search and historical path search to balance the depth and breadth searches. In the experimental section of the paper, the HCOA competes with 5 well-known metaheuristic algorithms in the CEC2017 benchmark functions, which contain 29 functions, with 23 of these ranking first. The state of work BPSO-CM is also chosen to compare with the HCOA, and the competition shows that the HCOA has a better performance in the 100-dimensional test of the CEC2017 benchmark functions. All the experimental results demonstrate that the HCOA presents highly accurate and robust results for high-dimensional optimization problems.

scientific reports



OPEN

# Pair barracuda swarm optimization algorithm: a natural-inspired metaheuristic method for high dimensional optimization problems

Jia Guo[1,2], Guoyuan Zhou[3], Ke Yan[4], Yuji Sato[5] & Yi Di[1,2]✉

**High-dimensional optimization presents a novel challenge within the realm of intelligent computing, necessitating innovative approaches. When tackling high-dimensional spaces, traditional evolutionary tools often encounter pitfalls, including dimensional catastrophes and a propensity to become trapped in local optima, ultimately compromising result accuracy. To address this issue, we introduce the Pair Barracuda Swarm Optimization (PBSO) algorithm in this paper. PBSO employs a unique strategy for constructing barracuda pairs, effectively mitigating the challenges posed by high dimensionality. Furthermore, we enhance global search capabilities by incorporating a support barracuda alongside the leading barracuda pair. To assess the algorithm's performance, we conduct experiments utilizing the CEC2017 standard function and compare PBSO against five state-of-the-art natural-inspired**

scientific reports



OPEN

# Mother optimization algorithm: a new human-based metaheuristic approach for solving engineering optimization

Ivana Matoušová[1]✉, Pavel Trojovský[1], Mohammad Dehghani[1], Eva Trojovská[1] & Juraj Kostra[2]

This article's innovation and novelty are introducing a new metaheuristic method called mother optimization algorithm (MOA) that mimics the human interaction between a mother and her children. The real inspiration of MOA is to simulate the mother's care of children in three phases education, advice, and upbringing. The mathematical model of MOA used in the search process and exploration is presented. The performance of MOA is assessed on a set of 52 benchmark functions, including unimodal and high-dimensional multimodal functions, fixed-dimensional multimodal functions, and the CEC 2017 test suite. The findings of optimizing unimodal functions indicate MOA's high ability in

scientific reports



OPEN

# A new human-based metaheuristic algorithm for solving optimization problems based on preschool education

Pavel Trojovský

In this paper, with motivation from the No Free Lunch theorem, a new human-based metaheuristic algorithm named Preschool Education Optimization Algorithm (PEOA) is introduced for solving optimization problems. Human activities in the preschool education process are the fundamental inspiration in the design of PEOA. Hence, PEOA is mathematically modeled in three phases: (i) the gradual growth of the preschool teacher's educational influence, (ii) individual knowledge development guided by the teacher, and (iii) individual increase of knowledge and self-awareness. The PEOA's performance in optimization is evaluated using fifty-two standard benchmark functions encompassing unimodal, high-dimensional multimodal, and fixed-dimensional multimodal types, as well as the CEC 2017 test suite. The optimization results show that PEOA has a high ability in

scientific reports



OPEN

# A new human-based metahurestic optimization method based on mimicking cooking training

Eva Trojovská ✉ & Mohammad Dehghani

Metaheuristic algorithms have a wide range of applications in handling optimization problems. In this study, a new metaheuristic algorithm, called the chef-based optimization algorithm (CBOA), is developed. The fundamental inspiration employed in CBOA design is the process of learning cooking skills in training courses. The stages of the cooking training process in various phases are mathematically modeled with the aim of increasing the ability of global search in exploration and the ability of local search in exploitation. A collection of 52 standard objective functions is utilized to assess the CBOA's performance in addressing optimization issues. The optimization results show that the CBOA is capable of providing acceptable solutions by creating a balance between exploration

# recent publications in the field regarding TSP (2023)

ELSEVIER

journal homepage: www.elsevier.com/locate/asoc

## A review of metaheuristic algorithms for solving TSP-based scheduling optimization problems

Bladimir Toaza [a,*], Domokos Esztergár-Kiss [a]

[a] Department of Transport Technology and Economics, Budapest University of Technology and Economics, Műegyetem rkp. 3, 1111 Budapest, Hungary

### HIGHLIGHTS

- Review of 120 metaheuristics solving TSP-based scheduling optimization problems.
- Tabular summary of descriptive and assessment features of the metaheuristics.
- Automation of large amount of search terms using a programming language and an API.
- GA is the most applied algorithm in publications, but ACO is the most cited one.

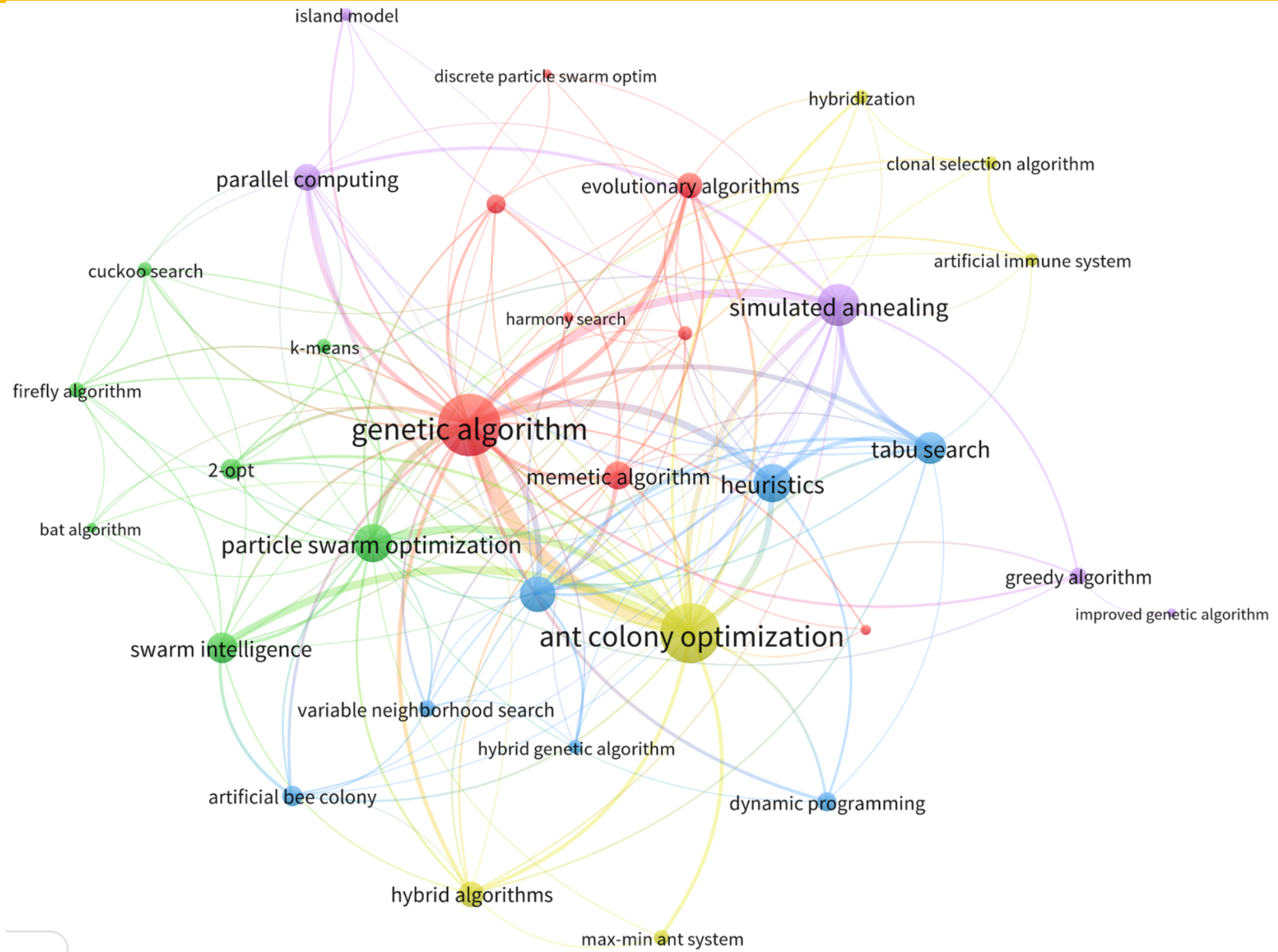
### ARTICLE INFO

*Keywords:*
Bibliometric analysis
Metaheuristic algorithms
Optimization problems
Scheduling optimization
Traveling salesman problem

### ABSTRACT

Activity-based scheduling optimization is a combinatorial problem built on the traveling salesman problem intending to optimize people schedules considering their trips and the available transportation network. Due to the difficulty of scheduling, traditional and exact methods are unable to provide appropriate solutions. Hence, new approaches have been introduced in the literature to settle these complex problems. One group of new techniques is known as metaheuristic algorithms, which provides a robust family of problem-solving methods created by mimicking natural phenomena. Although these new techniques might not find an optimal solution, they can find a near-optimal one in a moderate period. Furthermore, a myriad of novel algorithms has been

# recent publications in the field regarding TSP (2023)



Article

# Discovering faster matrix multiplication algorithms with reinforcement learning

https://doi.org/10.1038/s41586-022-05172-4

Received: 2 October 2021

Accepted: 2 August 2022

Published online: 5 October 2022

Open access

Check for updates

Alhussein Fawzi[1,2]✉, Matej Balog[1,2], Aja Huang[1,2], Thomas Hubert[1,2], Bernardino Romera-Paredes[1,2], Mohammadamin Barekatain[1], Alexander Novikov[1], Francisco J. R. Ruiz[1], Julian Schrittwieser[1], Grzegorz Swirszcz[1], David Silver[1], Demis Hassabis[1] & Pushmeet Kohli[1]

Improving the efficiency of algorithms for fundamental computations can have a widespread impact, as it can affect the overall speed of a large amount of computations. Matrix multiplication is one such primitive task, occurring in many systems—from neural networks to scientific computing routines. The automatic discovery of algorithms using machine learning offers the prospect of reaching beyond human intuition and outperforming the current best human-designed algorithms. However, automating the algorithm discovery procedure is intricate, as the space of possible algorithms is enormous. Here we report a deep reinforcement learning approach based on AlphaZero[1] for discovering efficient and provably correct algorithms for the multiplication of arbitrary matrices. Our agent, AlphaTensor, is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space. AlphaTensor discovered algorithms that outperform the state-of-the-art complexity for many matrix sizes. Particularly relevant is the case of $4 \times 4$ matrices in a finite field, where AlphaTensor's algorithm improves on Strassen's two-level algorithm for the first time, to our knowledge, since its discovery 50 years ago[2].

# *AlphaEvolve*: A coding agent for scientific and algorithmic discovery

Alexander Novikov*, Ngân Vũ*, Marvin Eisenberger*, Emilien Dupont*, Po-Sen Huang*, Adam Zsolt Wagner*, Sergey Shirobokov*, Borislav Kozlovskii*, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli and Matej Balog*
Google DeepMind[1]

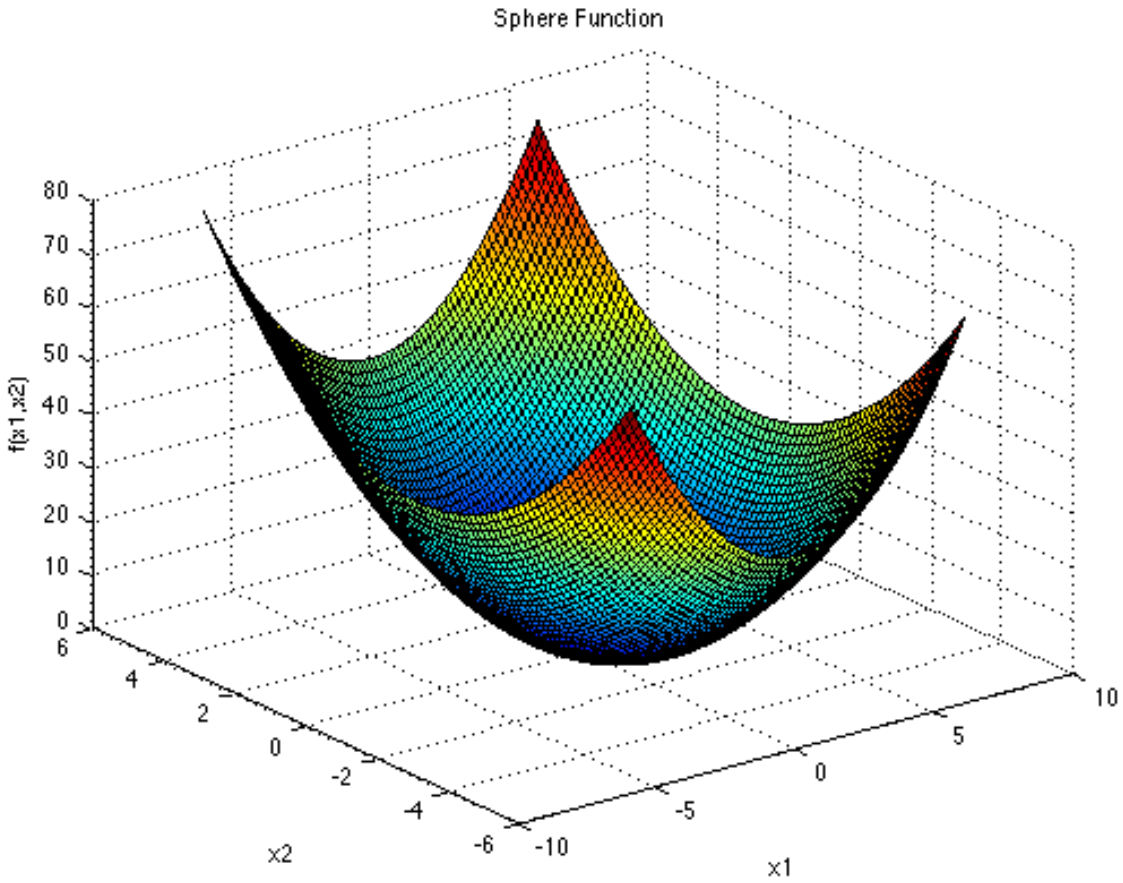
In this white paper, we present *AlphaEvolve*, an evolutionary coding agent that substantially enhances capabilities of state-of-the-art LLMs on highly challenging tasks such as tackling open scientific problems or optimizing critical pieces of computational infrastructure. *AlphaEvolve* orchestrates an autonomous pipeline of LLMs, whose task is to improve an algorithm by making direct changes to the code. Using an evolutionary approach, continuously receiving feedback from one or more evaluators, *AlphaEvolve* iteratively improves the algorithm, potentially leading to new scientific and practical discoveries. We demonstrate the broad applicability of this approach by applying it to a number of important computational problems. When applied to optimizing critical components of large-scale computational stacks at Google, *AlphaEvolve* developed a more efficient scheduling algorithm for data centers, found a functionally equivalent simplification in the circuit design of hardware accelerators, and accelerated the training of the LLM underpinning *AlphaEvolve* itself. Furthermore, *AlphaEvolve* discovered novel, provably correct algorithms that surpass state-of-the-art solutions on a spectrum of problems in mathematics and computer science, significantly expanding the scope of prior automated discovery methods (Romera-Paredes et al., 2023). Notably, *AlphaEvolve* developed a search algorithm that found a procedure to multiply two $4 \times 4$ complex-valued matrices using 48 scalar multiplications; offering the first improvement, after 56 years, over Strassen's algorithm in this setting. We believe *AlphaEvolve* and

# the problems we will look at

- search of minimum in Real-valued Multi-dimensional Functions (RMF)
- Traveling Salesman Problem (TSP)
  (nowadays *salesperson*)
- sorting as an optimization problem (just for fun)

- maybe: (0-1)-knapsack problem (KSP)
- maybe: *p*-facility location problem (PMP and PCP)
- maybe: some more problems as examples

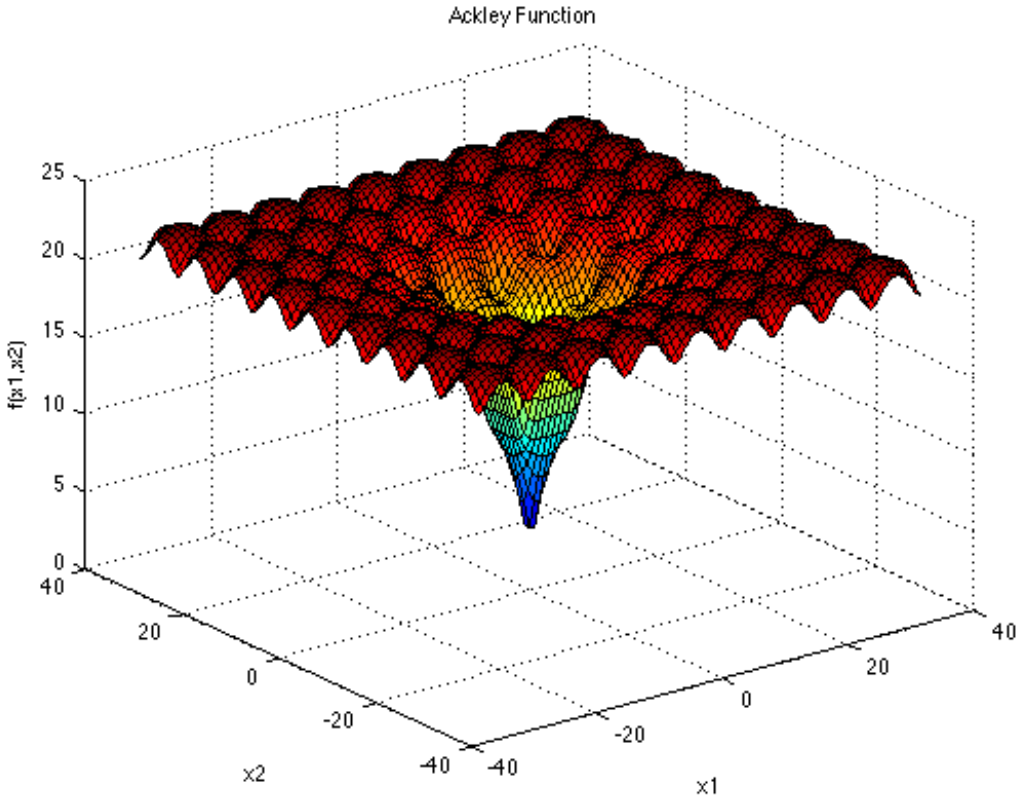
# RMF: Sphere function




$$f(x) = \sum_{i=1}^{d} x_i^2$$

[image from CEC2017 benchmark site]

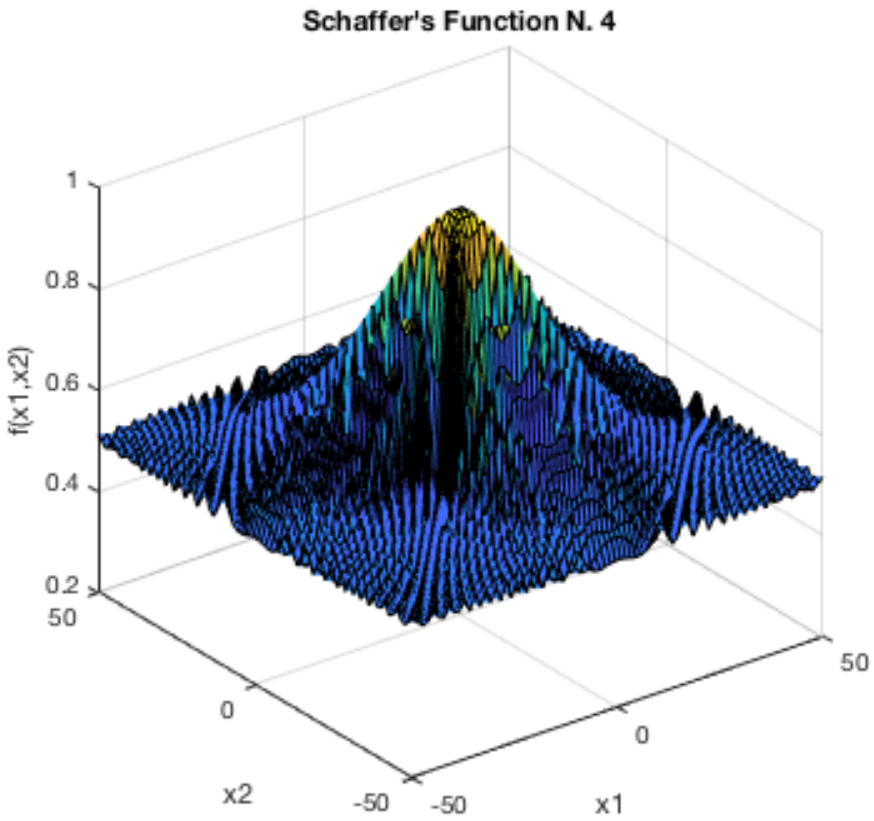
# RMF: Ackley function

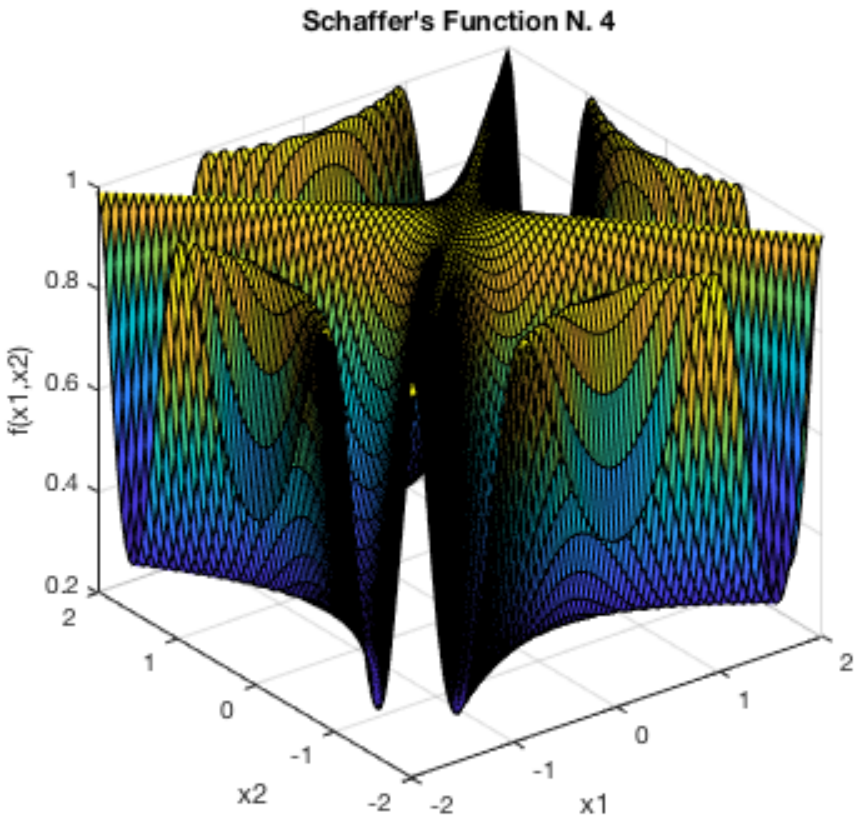



$$f(x) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + e$$

[image from CEC2017 benchmark site]

# RMF: Schaffer 4 function

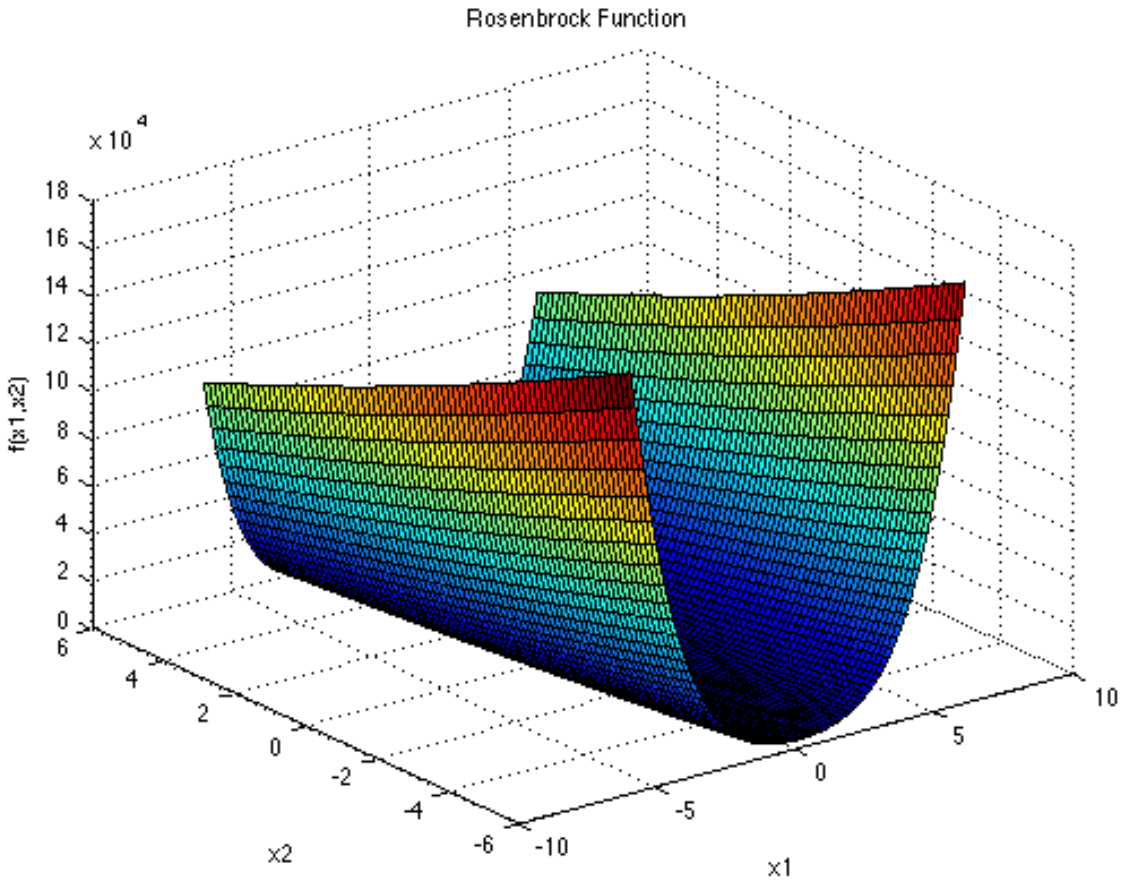



$$f(x) = 0.5 + \frac{\cos^2(\sin(|x_1^2 - x_2^2|)) - 0.5}{[1 + 0.001(X_1^2 + x_2^2)]^2}$$

[image from CEC2017 benchmark site]

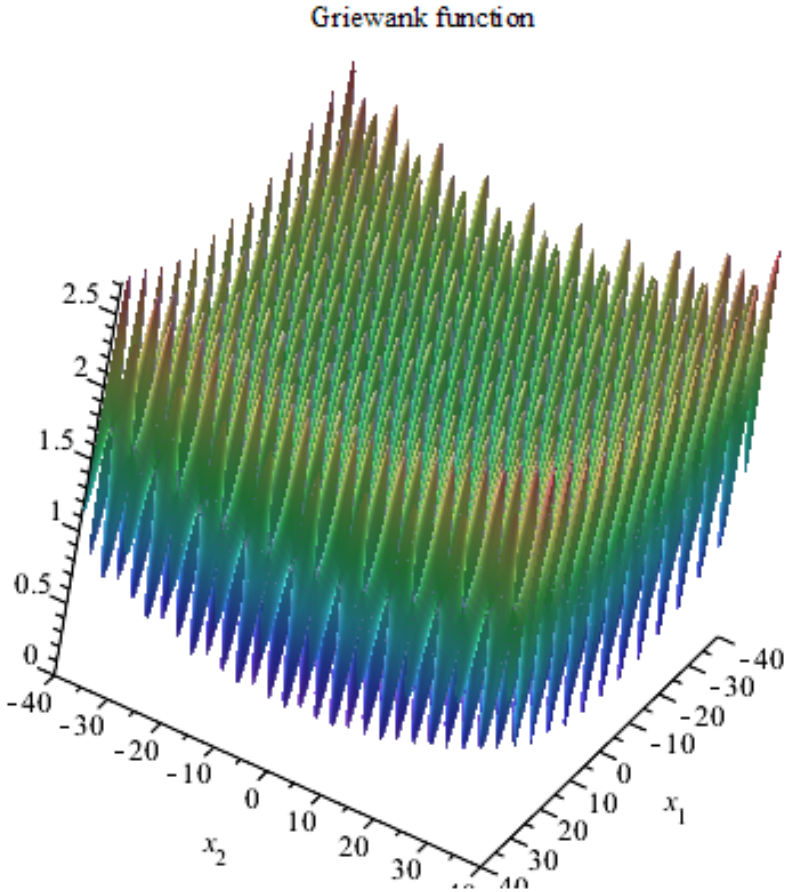
# RMF: Rosenbrock function




$$f(x) = \sum_{i=1}^{d-1} [100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

[image from CEC2017 benchmark site]

# RMF: Griewank function




$$f(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$

[By Gisling - CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=38868739]

# finding minimum of these functions

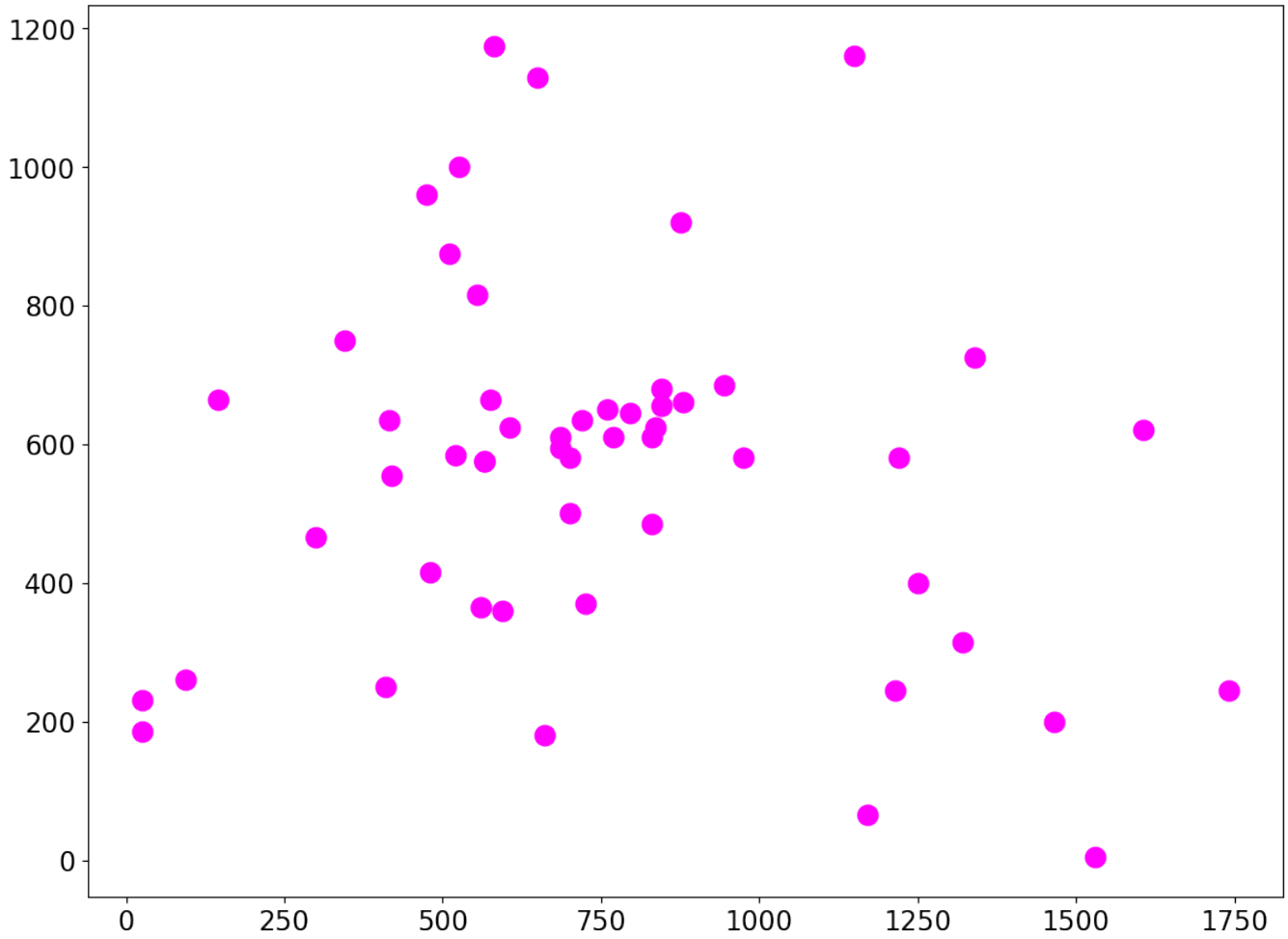
For more real-valued functions and common

- parameter settings,
- search areas,
- local/global optima,
- and code examples

take a look at https://www.sfu.ca/~ssurjano/optimization.html or look for Congress on Evolutionary Computation (https://attend.ieee.org/wcci-2026/ieee-cec-2025/) benchmarks, for instance 2017 edition

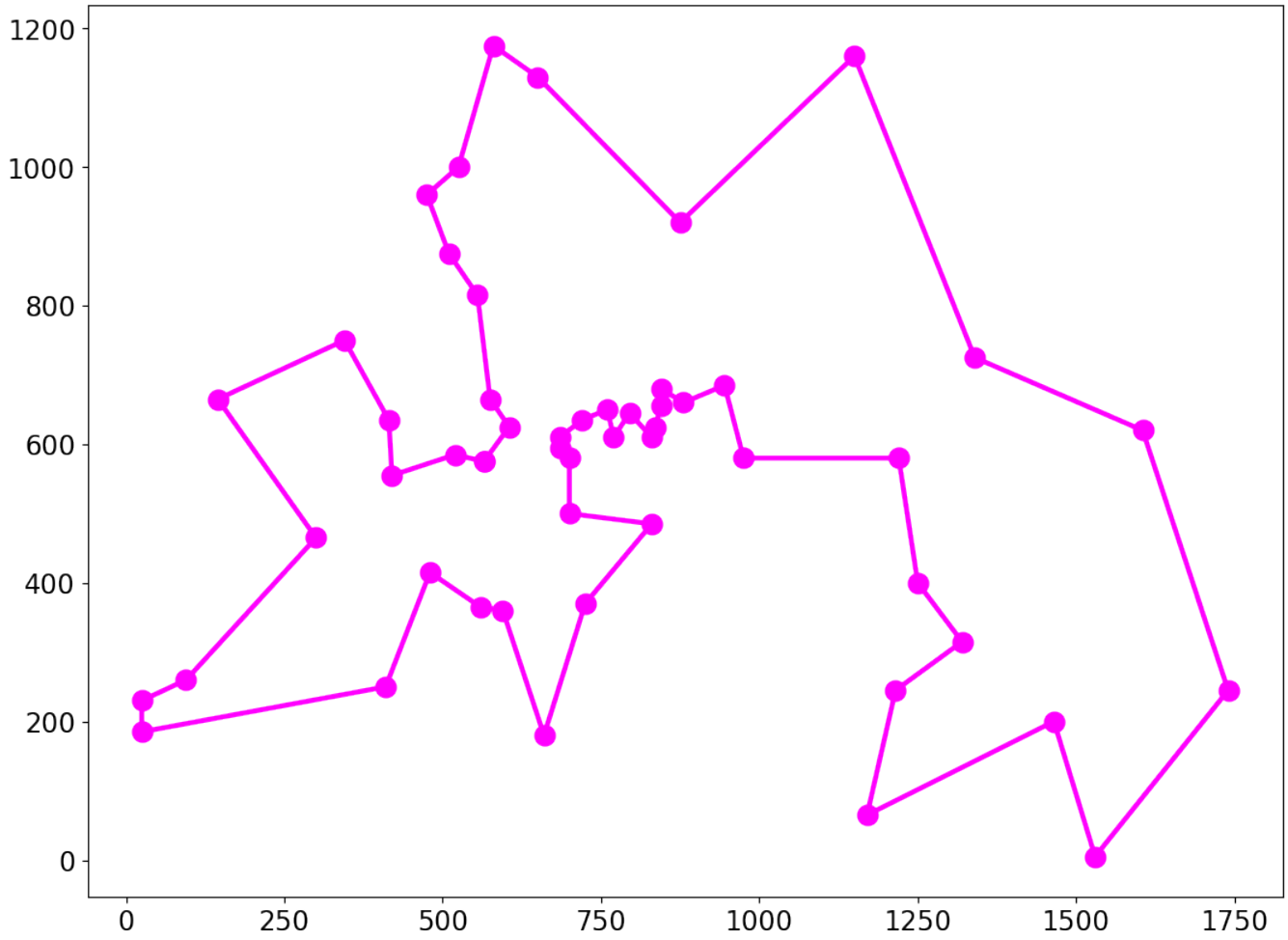
https://www.kaggle.com/code/kooaslansefat/cec-2017-benchmark

# traveling salesperson problem (Euclidean instance)




the cities distributed geographically (dataset berlin52)

# traveling salesperson problem: the solution




the best tour (known for this example): 0% relative error

# algorithms

- An algorithm is a finite sequence of well-defined steps (or instructions) to complete a task or solve a problem.
- In principal, the individual steps must be executable by a human being.
- The steps (or instructions) must perform a finite change of state (or configuration) on the system on which the algorithm is executed.
- Completing a task means that there is an other algorithm that can decide whether the final configuration has the required property.
- The time complexity of an algorithm is the number of steps the algorithm executes before it stops.

# computational complexity

- Algorithms can be grouped into classes according to their time complexity (same is true for space complexity).
- For an asymptotic upper bound according to some input of size $n$ we say:
  function $f$ is in the order of function $g$, whenever we have:

$$\exists c > 0 \;\; \exists n_0 \;\; \forall n > n_0 : |f(n)| \leq c \cdot g(n)$$

  and we write: $f(n) = \mathscr{O}(g(n))$.
- e.g.: if $f(n) = \mathscr{O}(n^3)$ then $f$ does not grow faster than cubic.
- There are more notations for other asymptotic characterizations: $o$, $\omega$, $\Omega$, $\Theta$.
- If you like, take a look at the complexity zoo:
  https://complexityzoo.net/Complexity_Zoo

# computational complexity

There are problems:

- that are arbitrary difficult
  (i.e., there is a hierarchy of problem classes).
- that don't have a solution at all (uncomputable problems)
  e.g.: decide whether an arbitrary program stops or whether two formal languages are equivalent, among others.
- where we know that they are computable
  (i.e., there exists a solution),
  but we don't know how to compute one. Look into *forbidden graph minors in graph theory*.
- for which there are known algorithms, but we don't know the *smallest* class they belong to,
  e.g., unknotting an unknot (take a look into knot theory) or factorization of numbers.

# runtime examples according to complexity

Assume we can deal with 1 million items in 1 second using a linear time algorithm (i.e., megahertz item processing):

| **size** $n$ | **function** | $\mathcal{O}$**-notation** | **time** |
|---|---|---|---|
| 1000000 | linear time | $\mathcal{O}(n)$ | 1 second |
| | quasi-linear time | $\mathcal{O}(n \log n)$ | 20 seconds |
| | quadratic time | $\mathcal{O}(n^2)$ | 11.6 days |
| | cubic time | $\mathcal{O}(n^3)$ | 31710 years |
| | quartic time | $\mathcal{O}(n^4)$ | 32 billion years |
| | exponential time | $\mathcal{O}(2^n)$ | eternal |
| | factorial time | $\mathcal{O}(n!)$ | no words any more |

Note that 32 billion years is roughly 2.3 times the age of the universe.

# runtime examples according to complexity

Assume we can deal with 1 million items in 1 millisecond using a linear time algorithm (i.e., gigahertz item processing):

| **size** $n$ | **function** | $\mathcal{O}$**-notation** | **time** |
|---|---|---|---|
| 1000000 | linear time | $\mathcal{O}(n)$ | 1 millisecond |
| | quasi-linear time | $\mathcal{O}(n \log n)$ | 20 milliseconds |
| | quadratic time | $\mathcal{O}(n^2)$ | 16 minutes |
| | cubic time | $\mathcal{O}(n^3)$ | 31.7 years |
| | quartic time | $\mathcal{O}(n^4)$ | 32 million years |
| | exponential time | $\mathcal{O}(2^n)$ | eternal |
| | factorial time | $\mathcal{O}(n!)$ | no words any more |

1000 times faster, but considering a human life span, we don't win much on large problems even for just cubic algorithms.

# other non-algorithmic methods to speed-up computation

- Parallelization on a $p$-processor machine gives you at most a linear speedup with a factor of $p$
  (and most of the time not even that).
- Quantum computation offers sometimes (Grover algorithm) a quadratic speedup regarding input size $n$.
- Quantum supremacy reduces runtime from exponential to polynomial but no application known for interesting problems.
- Google claimed in 2025 a quantum advantage of a factor of 13 000 on its quantum computer for a complex physical simulation.

# handable problem sizes according to complexity

Assume 1 nanosecond processing time per item (i.e., 1 GHz operating frequency), problem sizes handable in one hour:

| **function** | $\mathcal{O}$**-notation** | **problem size** |
|---|---|---|
| linear time | $\mathcal{O}(n)$ | 3.6 trillion |
| quasi-linear time | $\mathcal{O}(n \log n)$ | 96.6 billion |
| quadratic time | $\mathcal{O}(n^2)$ | 1.9 million |
| cubic time | $\mathcal{O}(n^3)$ | 15.3 thousand |
| quartic time | $\mathcal{O}(n^4)$ | 1377 |
| exponential time | $\mathcal{O}(2^n)$ | 41 |
| factorial time | $\mathcal{O}(n!)$ | 15 |

# algorithm types

- deterministic algorithms
  (i.e., you compute a solution step by step)
- non-deterministic algorithms
  (i.e., you guess a solution and check step by step)
- randomized or probabilistic algorithms
  (i.e., you use a die or a random generator sometimes)
- quantum algorithms (i.e., you use superposition from quantum theory)

Note, all types compute the same set of computable functions, they differ only in time and/or space complexity (see below).

# When do we consider a problem to be difficult?

A problem is difficult whenever we only know deterministic algorithms solving the problem that have at least exponential runtime (or polynomial runtime with a large exponent).

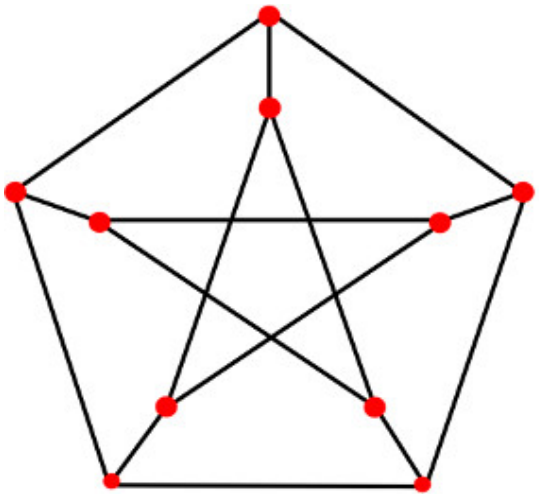
# side note: NP-complete and NP-hard problems

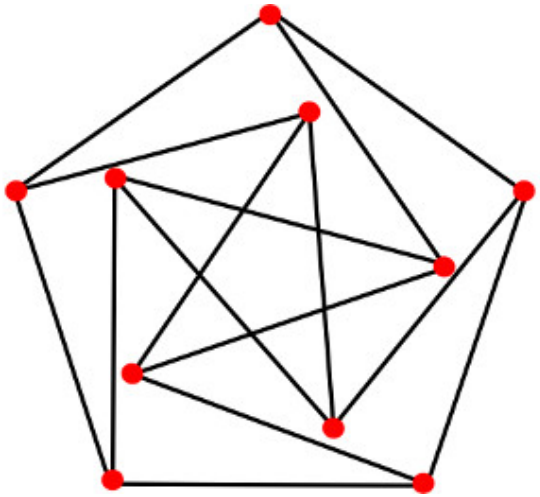
Evolutionary algorithms are often mentioned as a way to tackle NP-complete or NP-hard problems: What does that mean?

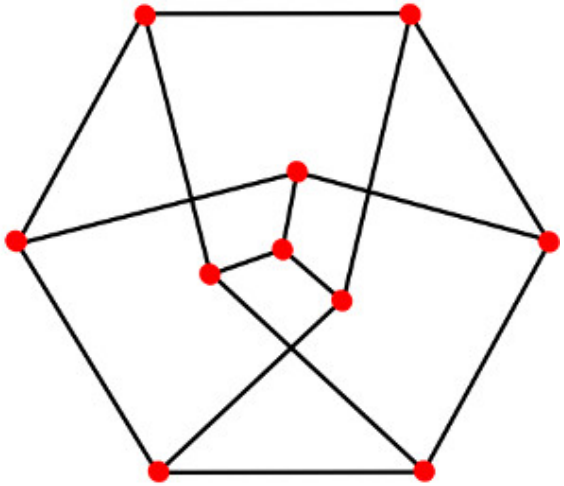
- A problem is NP-complete when we know a polynomial time deterministic algorithm that checks a solution, but we know only an exponential time deterministic algorithm to find a solution.
- A problem is, at least, NP-hard when we even don't know a polynomial time deterministic algorithm for the check.
- There are problems of which we know that they can be solved in exponential time, but we don't know whether they are NP-complete (or even simpler), e.g., the graph isomorphism problem, or the unknot problem.
- Essentially, we don't know whether the NP-complete problems are the same class as the polynomial time solvable problems. i.e., the famous 1 000 000$ question: $P = NP$ or $P \neq NP$?

# graph isomorphism problem

Are these graph isomorph (i.e., have the same structure)?







# unknot problem

Are these knots equivalent (i.e., have the same structure)?