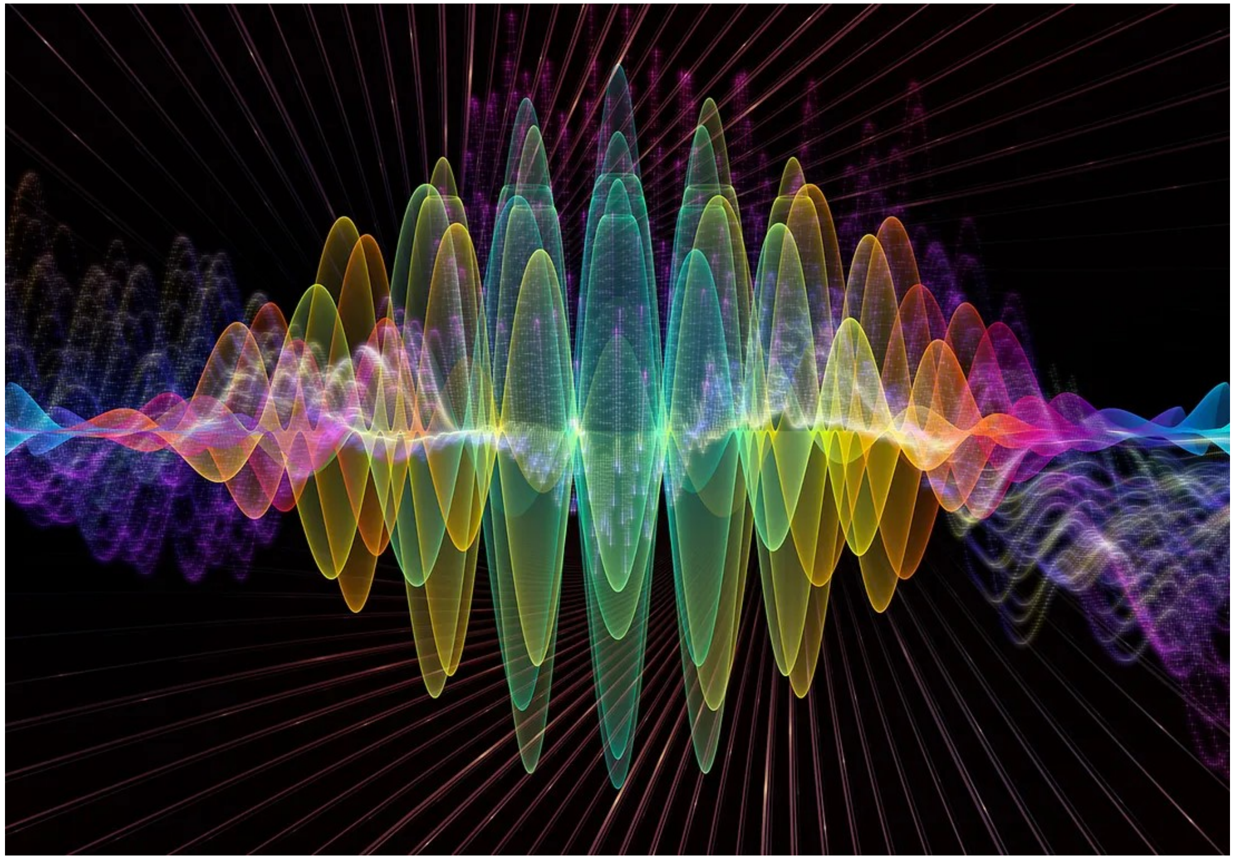
Markus Buchholz
Jan 16 · 4 min read · Listen


Google.com

The Sine Cosine Algorithm for Solving Optimization Problems with Constraints in C++

The following article presents the Sine Cosine Algorithm (SCA) for solving optimization problems with constraints. The algorithm was pioneered by **Seyedali Mirjalili.**

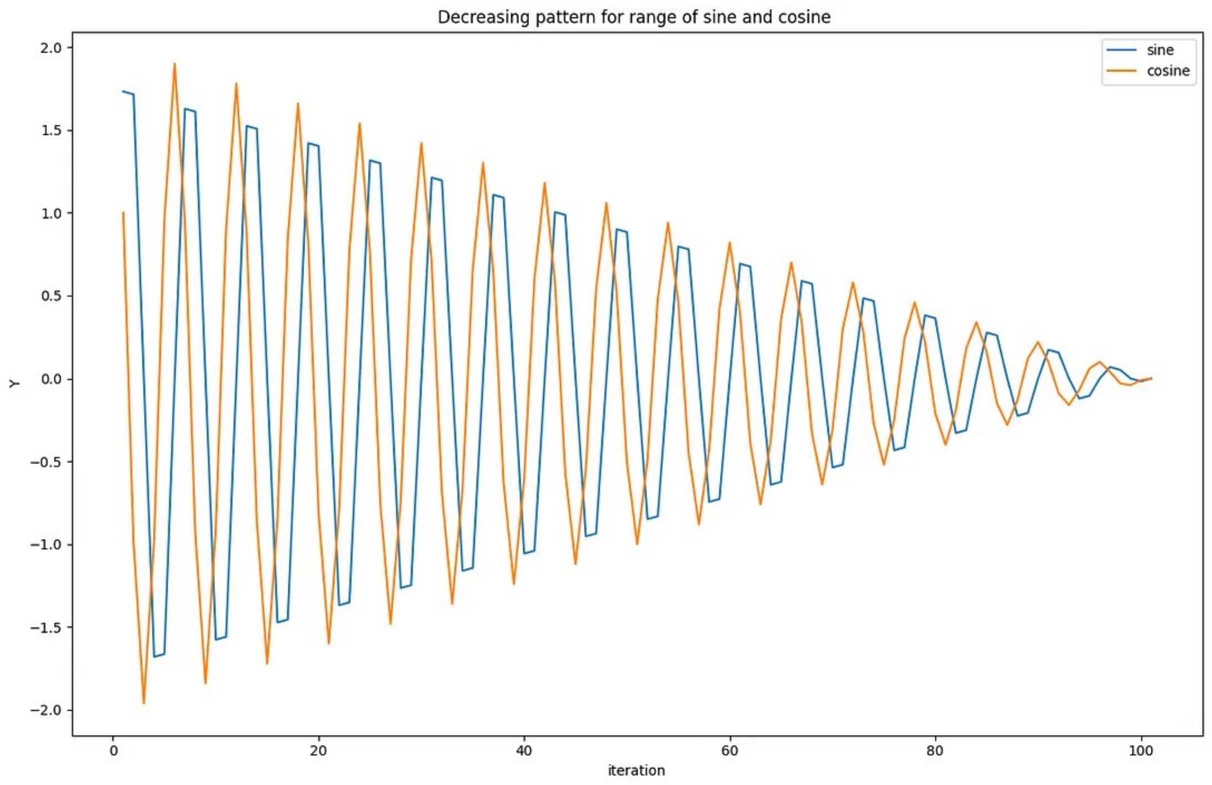
The SCA uses a mathematical model based on sine and cosine functions to generate many initial random solutions (objective function). Then, the algorithm randomly varies the solution either outwards or towards the optimal (best) solution. The applied method incorporates a number of random and adaptive variables to promote the exploitation and exploration of the search space.

The author finds the algorithm very powerful since the deployment into code (here C++) was simple. The algorithm includes only two important adjustable parameters. The number of iterations and a number of search agents to solve certain optimization problems have to be tuned. The algorithm converges to the optimal solution fast. The author compared SCA with the previously discussed Whale Optimization Algorithm. Discussed below optimization problems have been solved identically by both depicted algorithms. However, the SCA solves problems faster.

The source code for this article you will find on my **GitHub**.

Sine Cosine Algorithm (SCA)

The main principle of the algorithm is to apply the update position function only and approach the optimal solution while iterating. The iteration process reduces the influence of the sine and cosine functions for the update (the possible range of updates is reduced). The cyclic pattern of sine and cosine functions guarantees a new solution to be re-positioned around another solution. Please consider the following update functions.



Decreasing pattern for a range of sine and cosine (by author)

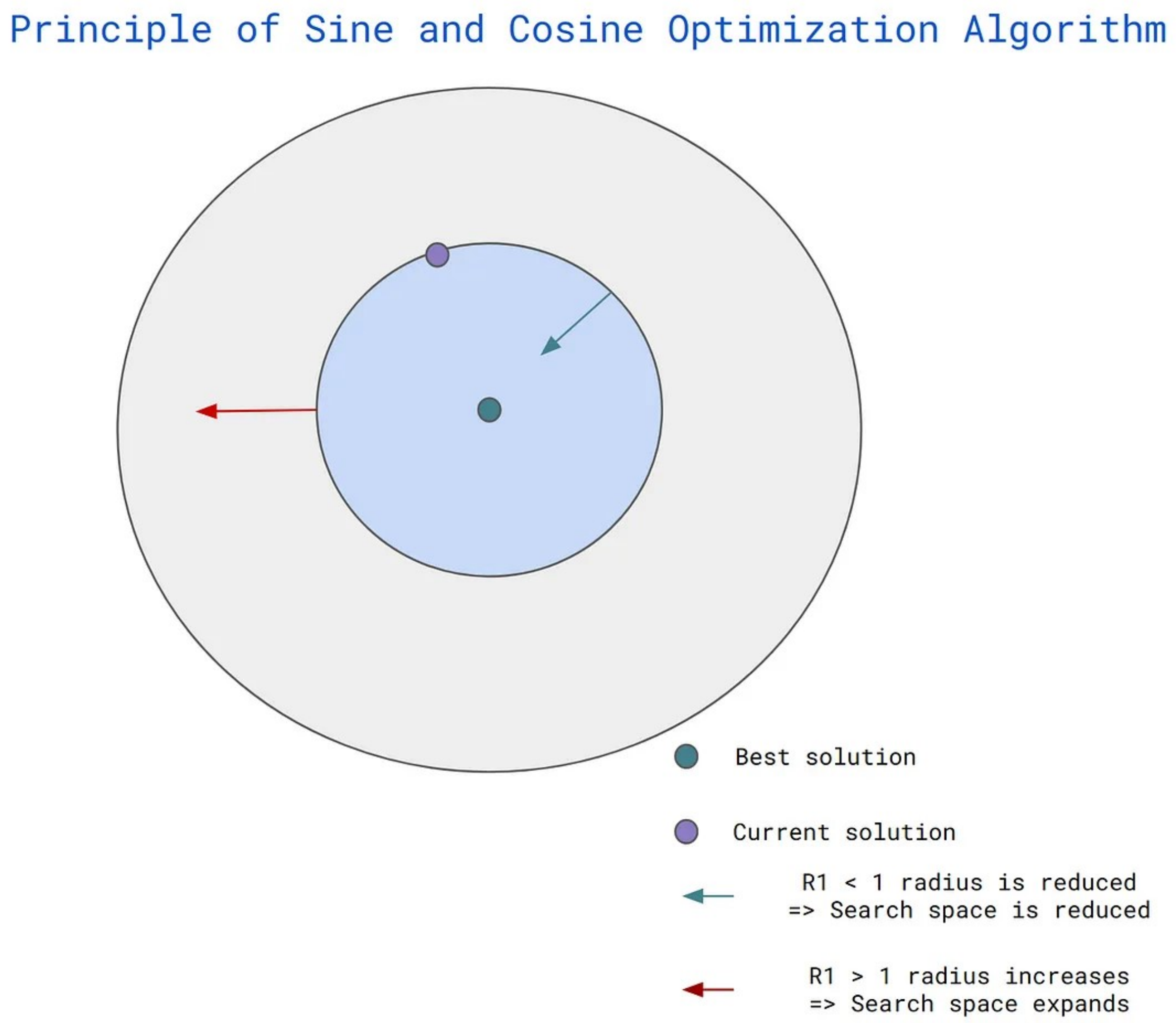
$$X(new) = \begin{cases} X(new) + rand_1 \cdot sin(rand_2) \cdot |rand_3 \cdot X(best) - X(old)| \Rightarrow rand_4 < 0.5 \\ X(new) + rand_1 \cdot cos(rand_2) \cdot |rand_3 \cdot X(best) - X(old)| \Rightarrow rand_4 \geq 0.5 \end{cases}$$

$$r_1 = A - \left(\frac{iter \cdot A}{Evolutions} \right)$$

where the X is the position of the agent, rand1 and rand4 random numbers in the range of [0,1]. rand2 — the random number in the range of [0, 2pi]. A =

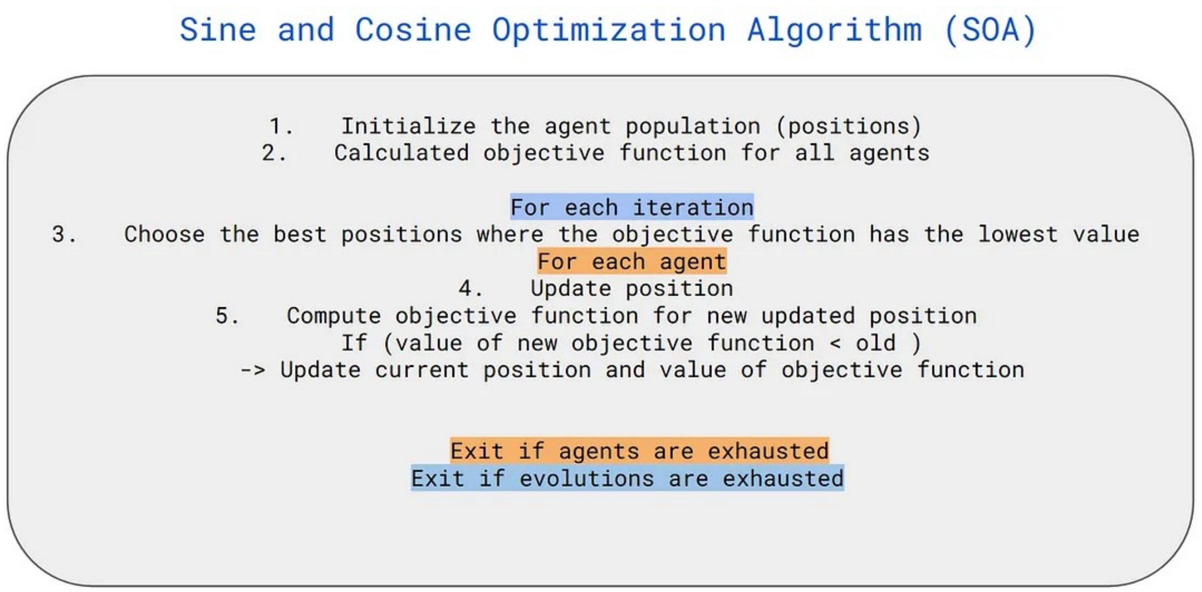
2.0.

Depending on the values of sine or cosine the different regions of space search are explored to find the optimal solution.
If the values of sine or cosine are larger than 1 or smaller than -1 the are is extended (the blue circle in the following figure is expanding and the **exploration process** takes place).
On the other hand, if the sine or cosine is within the range of -1 to 1 so the area of potential solutions (**exploitation process**) is reducing (the blue circle is shrinking). The algorithm converges.
As we can assume The SCA algorithm smoothly transits from exploration to exploitation using an adaptive range (we decrease the r1 value) in the sine and cosine functions.


Principle of Sine and Cosine Optimization Algorithm
Best solution
Current solution
R1 < 1 radius is reduced => Search space is reduced
R1 > 1 radius increases => Search space expands

SOA principle. (by author)

Pseudo-code can be depicted as follows,



Sine and Cosine Optimization Algorithm (SOA)

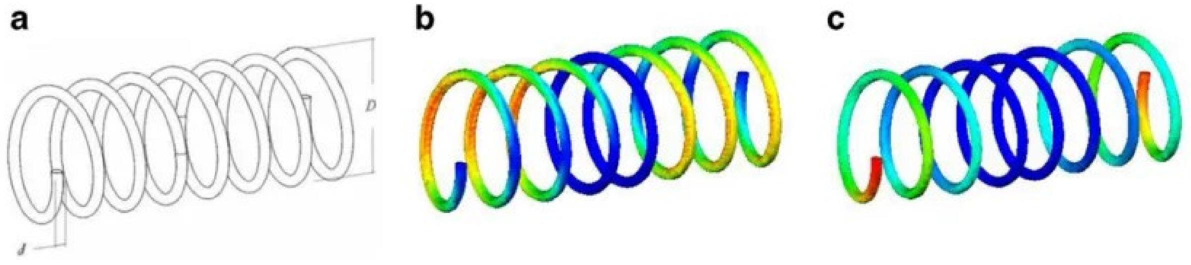
```
1.   Initialize the agent population (positions)
2.   Calculated objective function for all agents

For each iteration
3.   Choose the best positions where the objective function has the lowest value
    For each agent
        4.   Update position
        5.   Compute objective function for new updated position
             If (value of new objective function < old )
             -> Update current position and value of objective function

    Exit if agents are exhausted
Exit if evolutions are exhausted
```

Pseudo-code SOA. (by author)

Tests

In this article, I used the SCA to solve two optimizations problem. The simple one is the ordinary quadratic function I used for my personal debugging purpose.

The second problem is connected (benchmark optimization problem) with tension/compression spring where the objective of the problem is to minimize the weight. There are three design variables: wire diameter (d), mean coil diameter (D), and the number of active coils (N). This is a threedimensional problem (each variable is considered as a dimension).



(a) Schematic of the spring; (b) stress distribution evaluated at the optimum design; and c) displacement distribution evaluated at the optimum design (article I referred to)

The optimization problem can be formulated as follows :

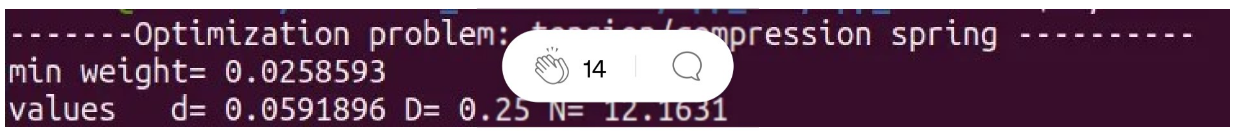
$$\begin{aligned}
&\text{Consider} && \vec{x} = [x_1\ x_2\ x_3] = [d\ D\ N],\\
&\text{Minimize} && f(\vec{x}) = (x_3 + 2)x_2x_1^2\\
&\text{Subject to} && g_1(\vec{x}) = 1 - \frac{x_2^3x_3}{71785x_1^4} \le 0,\\
& && g_2(\vec{x}) = \frac{4x_2^2 - x_1x_2}{12566\left(x_2x_1^3 - x_1^4\right)} + \frac{1}{5108x_1^2} \le 0,\\
& && g_3(\vec{x}) = 1 - \frac{140.45x_1}{x_2^2x_3} \le 0,\\
& && g_4(\vec{x}) = \frac{x_1 + x_2}{1.5} - 1 \le 0,\\
&\text{Variable range} && 0.05 \le x_1 \le 2.00,\\
& && 0.25 \le x_2 \le 1.30,\\
& && 2.00 \le x_3 \le 15.0
\end{aligned}$$

From article

The existing constraints have been included in the objective function as penalties. I modeled penalties as a **quadratic loss function** (see here and here).

The implemented method solves the problem (computes the minimum weight of the spring). The solution I received was equal (almost) to the article I was inspired by.

Optimization Algorithms Programming Mathematics Cpp



```
-------Optimization problem: tension/compression spring ----------
min weight= 0.0258593
values   d= 0.0591896 D= 0.25 N= 12.1631
```

Result of the optimization problem. Spring design (by author)

14

Plotting requires incorporating the header file which has to be in the same folder as your cpp (a file you can clone from my repository).
Your program can be compiled as follows,

```
//compile
g++ my_prog.cpp -o my_prog -I/usr/include/python3.8 -lpython3.8//

 //run
./my_prog

//folder tree

├── my_prog
├── my_prog.cpp
├── matplotlibcpp.h
```

Thank you for reading.