Abstract
We present an open-source software framework, EvoLP.jl [2], to support the research in optimisation using Evolutionary Computation (EC) in the Norwegian scientific community. EC is highly relevant in many problems in artificial intelligence, engineering and statistics when non-convex or non-differentiable functions appear. The software is a package in the Julia programming language that provides reusable computing blocks for experimenting and analysing several components for single-objective EC algorithms. By stacking these blocks, the user can quickly create modular solvers where each of the components can be easily swapped for testing. In addition, it provides a few built-in algorithms ready to use out of the box. A bunch of utilities for analysis are available as well: test functions, result reporting, and statistics logging and overview. EvoLP.jl is an effort of the Norwegian Open Artificial Intelligence Lab.

A visual example: The 8-queens problem
This example deals with a classical combinatorial problem in AI where the goal is to place 8 queens in a chess board such that no queen checks each other [1]. Fig. 1 shows three configurations where the constraints and possible clashes are highlighted.

Let’s design a solution
Using the provided blocks we can set up a solver quickly. We would need:
- A permutation generator
- A permutation mutator
- A tournament selector
- An objective: minimise conflicts
- A permutation recombinator

Let’s code the solution
What is Julia?
Julia is high-level, high-performance programming language very suitable for scientific computing. It is part of the PetaFLOPS Club (10^{15} floating point operations per second) along with C, C++ and Fortran, and its syntax is similar to Python or MATLAB. This is the Julia code for solving the 8-queen problem using EvoLP.jl:

```julia
using EvoLP
X = permutation_vector_pop(100, 8, 1:8)
S = TournamentSelectionSteady(5)
C = OrderOneCrossover()
M = SwapMutation()
f = diag_constraints(x)
result = mySteadyGA(statsbook, diag_constraints, X, 500, ➔ S, C, M, 0.8)
@show optimum(result)
@show optimizer(result)
```

And here is one possible output of the solution above:
```
 optimum(result) = 0
 optimizer(result) = Any[5, 1, 8, 6, 3, 7, 2, 4]
```

Check the full step-by-step example in the documentation.

What else can EvoLP do?
Components
- Random population generators for vectors and particles
- Parent selectors
- Several recombinators and mutators
- Test functions for benchmarking your algorithms
- Convenient result reporting and a log-book for statistics
- Built-in algorithms
- Support for custom operators

And what can I use it for?
Combinatorial challenges:
- Assignment and packing
- Scheduling and search
- Constraint satisfaction and optimisation
Numerical optimisation and tuning for machine learning:
- Hyperparameter tuning
- Neuroevolution
- Feature selection

Jump right into it
EvoLP.jl is well-tested, provides extensive documentation and is free—available for everyone to use under an open-source license at GitHub. After installing Julia, you can install EvoLP.jl by using the Julia REPL:
```
 julia> import Pkg
 julia> Pkg.add("EvoLP")
```

This should install EvoLP in your system.

Acknowledgements
EvoLP.jl is partly funded by Project no. 811284 of The Research Council of Norway. We would like to thank the Norwegian Open Artificial Intelligence Lab for the promotion and hosting of the framework in its GitHub repository, and IDI for access to its computing resources.

References