Problema OneMax e Função Armadilha

Danilo Sipoli Sanches

Departamento Acadêmico de Computação
Universidade Tecnológica Federal do Paraná
Cornélio Procópio

Problema OneMax

- Busca maximizar o número de "1s" durante a busca;
- Inicialmente as soluções são geradas aleatoriamente por uma lista de bits 0 e 1;
- O objetivo é gerar a solução com a maior quantidade de “1”;
- Exemplo:

- 10010 (soma == 2)
- 01110 (soma == 3)
- 11111 (soma == 5)

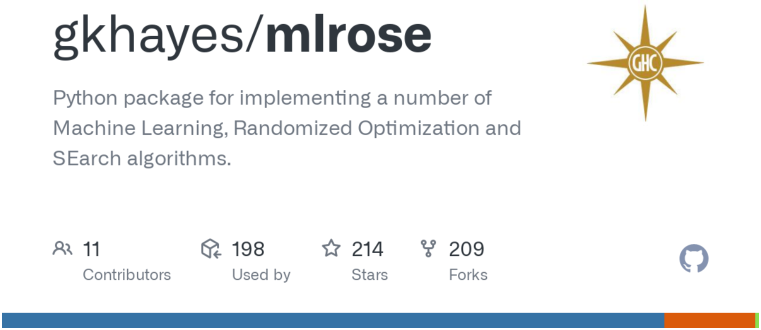
Problema OneMax Enganoso

- A aptidão de uma solução é o número de 1s que ela contém, a menos que sejam todos 0s, caso em que sua aptidão é o tamanho da solução + 1;
- Chamado de problema armadilha, uma vez que o algoritmo é recompensado gradativamente para cada 1 que adiciona ao problema, mas a melhor solução consiste em todos os 0s;
- Exemplo:

- 1001 (soma $==$ 2)
- 0111 (soma $==$ 3)
- 1111 (soma $==$ 4)
- 0000 (soma $==$ 5)

Desenvolvimento em Python - Framework Mlrose

- Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose



gkhayes/**mlrose**

Python package for implementing a number of Machine Learning, Randomized Optimization and SEarch algorithms.

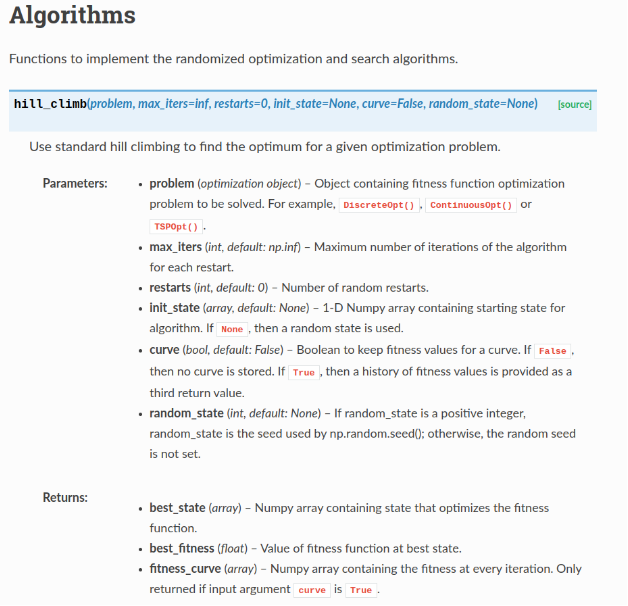
11 Contributors

198 Used by

214 Stars

209 Forks

Framework Mlrose - Hill Climbing



Algorithms

Functions to implement the randomized optimization and search algorithms.

hill_climb(*problem, max_iters=inf, restarts=0, init_state=None, curve=False, random_state=None*) [source]

Use standard hill climbing to find the optimum for a given optimization problem.

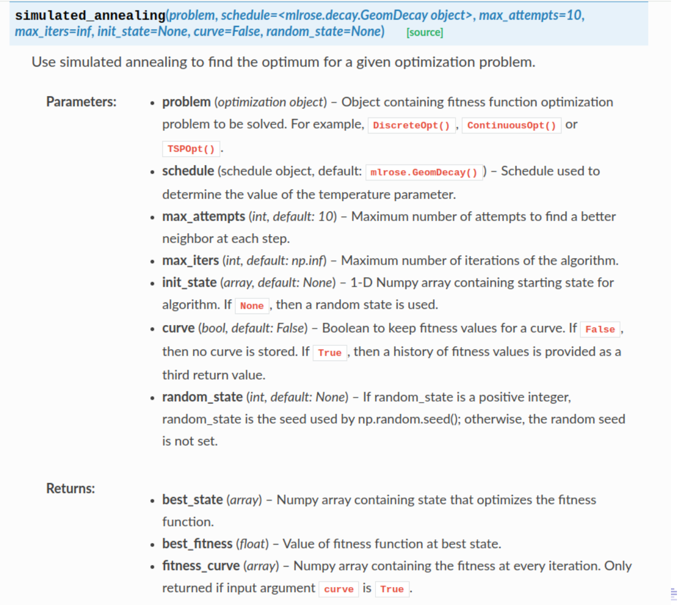
Parameters:

- **problem** (*optimization object*) – Object containing fitness function optimization problem to be solved. For example, `DiscreteOpt()`, `ContinuousOpt()` or `TSPOpt()`.
- **max_iters** (*int, default: np.inf*) – Maximum number of iterations of the algorithm for each restart.
- **restarts** (*int, default: 0*) – Number of random restarts.
- **init_state** (*array, default: None*) – 1-D Numpy array containing starting state for algorithm. If `None`, then a random state is used.
- **curve** (*bool, default: False*) – Boolean to keep fitness values for a curve. If `False`, then no curve is stored. If `True`, then a history of fitness values is provided as a third return value.
- **random_state** (*int, default: None*) – If random_state is a positive integer, random_state is the seed used by np.random.seed(); otherwise, the random seed is not set.

Returns:

- **best_state** (*array*) – Numpy array containing state that optimizes the fitness function.
- **best_fitness** (*float*) – Value of fitness function at best state.
- **fitness_curve** (*array*) – Numpy array containing the fitness at every iteration. Only returned if input argument `curve` is `True`.

Framework Mlrose - Simulated Annealing



`simulated_annealing`*(problem, schedule=<mlrose.decay.GeomDecay object>, max_attempts=10, max_iters=inf, init_state=None, curve=False, random_state=None)* [source]

Use simulated annealing to find the optimum for a given optimization problem.

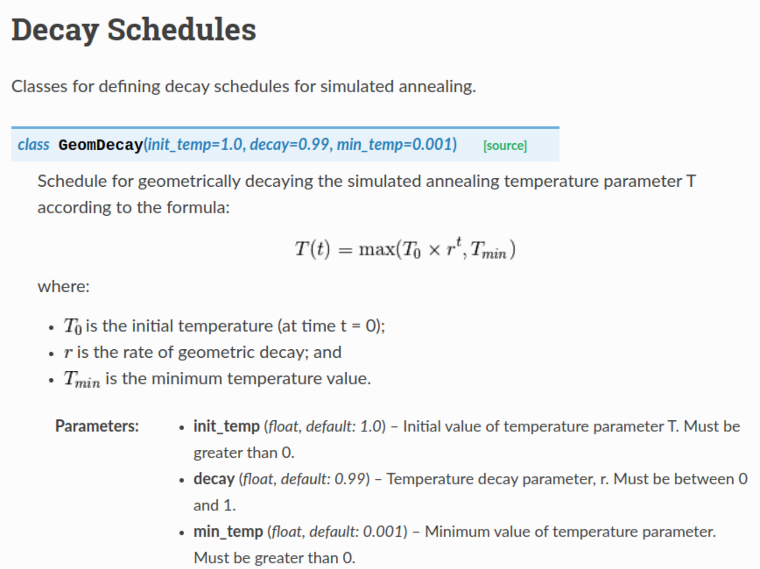
Parameters:

- **problem** (*optimization object*) – Object containing fitness function optimization problem to be solved. For example, `DiscreteOpt()`, `ContinuousOpt()` or `TSPOpt()`.
- **schedule** (schedule object, default: `mlrose.GeomDecay()`) – Schedule used to determine the value of the temperature parameter.
- **max_attempts** (*int, default: 10*) – Maximum number of attempts to find a better neighbor at each step.
- **max_iters** (*int, default: np.inf*) – Maximum number of iterations of the algorithm.
- **init_state** (*array, default: None*) – 1-D Numpy array containing starting state for algorithm. If `None`, then a random state is used.
- **curve** (*bool, default: False*) – Boolean to keep fitness values for a curve. If `False`, then no curve is stored. If `True`, then a history of fitness values is provided as a third return value.
- **random_state** (*int, default: None*) – If random_state is a positive integer, random_state is the seed used by np.random.seed(); otherwise, the random seed is not set.

Returns:

- **best_state** (*array*) – Numpy array containing state that optimizes the fitness function.
- **best_fitness** (*float*) – Value of fitness function at best state.
- **fitness_curve** (*array*) – Numpy array containing the fitness at every iteration. Only returned if input argument `curve` is `True`.

Framework Mlrose - Ajuste da Temperatura (SA)



Decay Schedules

Classes for defining decay schedules for simulated annealing.

class **GeomDecay**(*init_temp=1.0, decay=0.99, min_temp=0.001*) [source]

Schedule for geometrically decaying the simulated annealing temperature parameter T according to the formula:

$$T(t) = \max(T_0 \times r^t, T_{min})$$

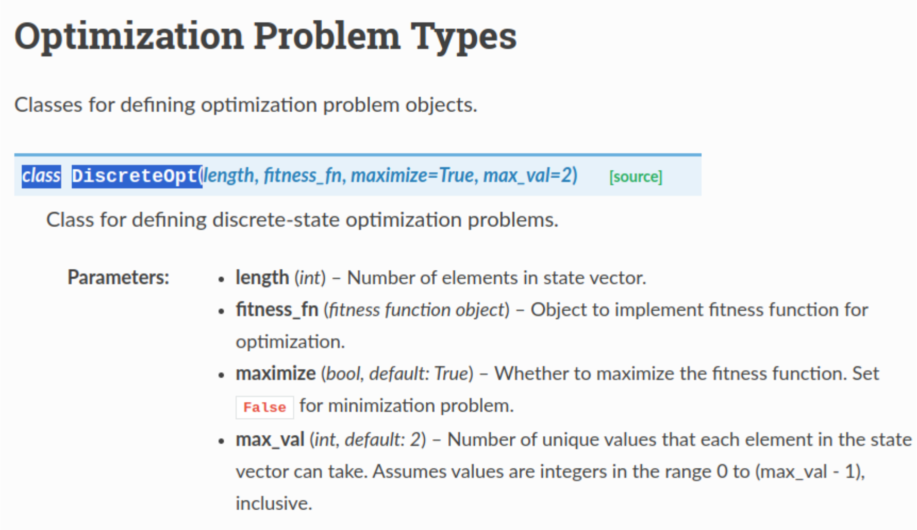
where:

- T_0 is the initial temperature (at time t = 0);
- r is the rate of geometric decay; and
- T_{min} is the minimum temperature value.

Parameters:

- **init_temp** (*float, default: 1.0*) – Initial value of temperature parameter T. Must be greater than 0.
- **decay** (*float, default: 0.99*) – Temperature decay parameter, r. Must be between 0 and 1.
- **min_temp** (*float, default: 0.001*) – Minimum value of temperature parameter. Must be greater than 0.

Framework Mlrose - class DiscreteOpt()



Optimization Problem Types

Classes for defining optimization problem objects.

class `DiscreteOpt`(*length*, *fitness_fn*, *maximize=True*, *max_val=2*) [source]

Class for defining discrete-state optimization problems.

Parameters:

- **length** (*int*) – Number of elements in state vector.
- **fitness_fn** (*fitness function object*) – Object to implement fitness function for optimization.
- **maximize** (*bool, default: True*) – Whether to maximize the fitness function. Set `False` for minimization problem.
- **max_val** (*int, default: 2*) – Number of unique values that each element in the state vector can take. Assumes values are integers in the range 0 to (max_val - 1), inclusive.