class HillClimbingReset(LocalSearch):

"""Algoritmo de ascension de colinas con reinicio aleatorio."""

def solve(self, problem: OptProblem):

"""Resuelve un problema de optimizacion con ascension de colinas.

Argumentos:

==========

problem: OptProblem

un problema de optimizacion

"""

# Inicio del reloj

start = time()

# Crear el nodo inicial con permutacion inicial aleatoria

actual = Node(problem.init, problem.obj\_val(problem.init))

mejor = actual

while True:

# Determinar las acciones que se pueden aplicar

# y las diferencias en valor objetivo que resultan

diff = problem.val\_diff(actual.state)

# Buscar las acciones que generan el mayor incremento de valor objetivo

max\_acts = [act for act, val in diff.items() if val == max(diff.values())]

# Elegir una accion aleatoria

act = choice(max\_acts)

# Retornar si estamos en un optimo local

if diff[act] <= 0:

self.tour = actual.state

self.value = actual.value

end = time()

self.time = end - start

if actual.value < mejor.value:

mejor = actual

actual = problem.random\_reset() # Reiniciar con permutacion inicial aleatoria

continue

# Sino, moverse a un nodo con el estado sucesor

actual = Node(problem.result(actual.state, act), actual.value + diff[act])

self.niters += 1