Implementation and comparison of local search algorithms applied to the eight queens puzzle

Francesco Esposito

Faculty of Computer Engineering
Università degli studi di Napoli "Federico II"

June 2024

Introduction

- Goal: Solve eight queen puzzle using different local search algorithms.

Introduction

- Goal: Solve eight queen puzzle using different local search algorithms.
  - Proprieties and performance of algorithms are also discussed.

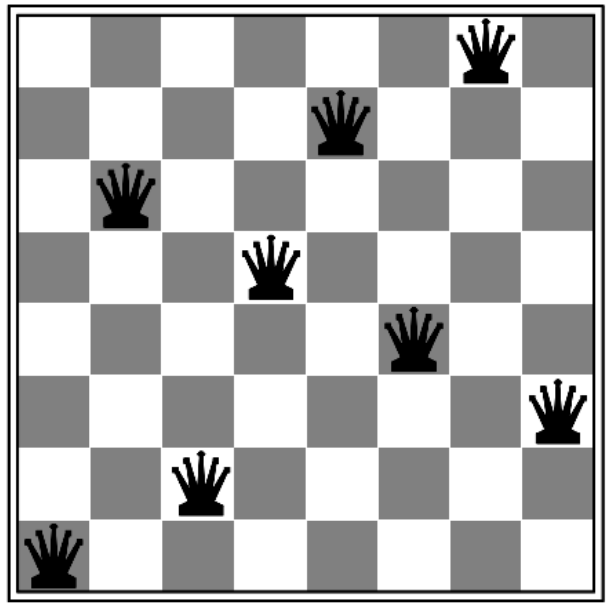
Introduction

- Goal: Solve eight queen puzzle using different local search algorithms.
  - Proprieties and performance of algorithms are also discussed.
- How? Using Python for the implementation and "Artificial Intelligent: A Modern approach (third edition)" as a reference.

Introduction

- Goal: Solve eight queen puzzle using different local search algorithms.
  - Proprieties and performance of algorithms are also discussed.
- How? Using Python for the implementation and "Artificial Intelligent: A Modern approach (third edition)" as a reference.
  - Code will be available on Github (on francespos account) after the exam.

The eight queens puzzle



The eight queens puzzle

- Problem: place eight queens on a chessboard so that they don't attack each other.

The eight queens puzzle

- Problem: place eight queens on a chessboard so that they don't attack each other.
- Rules: A queen can move horizontally, vertically and diagonally within an arbitrary range.

The eight queens puzzle

- Problem: place eight queens on a chessboard so that they don't attack each other.
- Rules: A queen can move horizontally, vertically and diagonally within an arbitrary range.
  - We will ignore states where queens are on the same column, because they can't belong to the solution.

The eight queens puzzle

- Problem: place eight queens on a chessboard so that they don't attack each other.
- Rules: A queen can move horizontally, vertically and diagonally within an arbitrary range.
  - We will ignore states where queens are on the same column, because they can't belong to the solution.
- We will use PEAS approach to describe the problem.

PEAS

- Performance measure: (Opposite of) number of pairs of queens that attack each other.

PEAS

- Performance measure: (Opposite of) number of pairs of queens that attack each other.
- Environment: 8x8 chessboard with eight queens.

PEAS

- Performance measure: (Opposite of) number of pairs of queens that attack each other.
- Environment: 8x8 chessboard with eight queens.
- Actuator: Player who moves the queens.

PEAS

- Performance measure: (Opposite of) number of pairs of queens that attack each other.
- Environment: 8x8 chessboard with eight queens.
- Actuator: Player who moves the queens.
- Sensor: Player looking at the board.

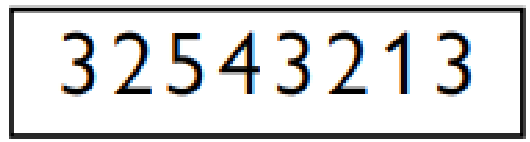
State space representation

- Since a column must be occupied by a single queen, we will represent states as a list of eight numbers between 0 and 7.

State space representation

- Since a column must be occupied by a single queen, we will represent states as a list of eight numbers between 0 and 7.
  - The element in position i is the row position of the queen on column i.

State space representation



Adjacent state: implementation

- An adjacent state corresponds to a movement of a queen in her column.

Adjacent state: implementation

- An adjacent state corresponds to a movement of a queen in her column.
- For each column c (i.e. position in the list), for each row r (i.e. value of an element in that position):

Adjacent state: implementation

- An adjacent state corresponds to a movement of a queen in her column.
- For each column c (i.e. position in the list), for each row r (i.e. value of an element in that position):
  - if queen in the column c is not in position r, an adjacent state is represent by the same list with the new element r in position c.

Heuristic function: implementation

- An heuristic function is the number of pairs of queens that attack each other.

Heuristic function: implementation

- An heuristic function is the number of pairs of queens that attack each other.
- This heuristic function is admissible, because it never overestimates the cost of reaching the goal.

Heuristic function: implementation

- An heuristic function is the number of pairs of queens that attack each other.
- This heuristic function is admissible, because it never overestimates the cost of reaching the goal.
  - To resolve n conflicts, at least n moves are required.

Heuristic function: implementation

- An heuristic function is the number of pairs of queens that attack each other.
- This heuristic function is admissible, because it never overestimates the cost of reaching the goal.
  - To resolve n conflicts, at least n moves are required.
- The heuristic function is calculated as the number of conflicts on each row and on each diagonal.

Hill climbing: implementation

- Hill climbing search algorithm only stores the current state.

Hill climbing: implementation

- Hill climbing search algorithm only stores the current state.
- In each iteration, it moves to the adjacent state with the lowest value.

Hill climbing: implementation

- Hill climbing search algorithm only stores the current state.
- In each iteration, it moves to the adjacent state with the lowest value.
  - In other words, it points to the direction of the steepest descent.

Hill climbing: implementation

- Hill climbing search algorithm only stores the current state.
- In each iteration, it moves to the adjacent state with the lowest value.
  - In other words, it points to the direction of the steepest descent.
- Starting from initial state, the heuristic function is assessed for each neighbor, and the state with the lowest value is chosen.

Hill climbing: implementation

- Hill climbing search algorithm only stores the current state.
- In each iteration, it moves to the adjacent state with the lowest value.
  - In other words, it points to the direction of the steepest descent.
- Starting from initial state, the heuristic function is assessed for each neighbor, and the state with the lowest value is chosen.
  - After an iteration, we get a pair (state, value) for the best neighbor.

Hill climbing: implementation

- Hill climbing search algorithm only stores the current state.
- In each iteration, it moves to the adjacent state with the lowest value.
  - In other words, it points to the direction of the steepest descent.
- Starting from initial state, the heuristic function is assessed for each neighbor, and the state with the lowest value is chosen.
  - After an iteration, we get a pair (state, value) for the best neighbor.
  - If "value" is greater than or equal to current state value, a local minimum has been reached.

Hill climbing: implementation

- Hill climbing search algorithm only stores the current state.
- In each iteration, it moves to the adjacent state with the lowest value.
  - In other words, it points to the direction of the steepest descent.
- Starting from initial state, the heuristic function is assessed for each neighbor, and the state with the lowest value is chosen.
  - After an iteration, we get a pair (state, value) for the best neighbor.
  - If "value" is greater than or equal to current state value, a local minimum has been reached.
  - Otherwise, a new iteration starts with the pair (state, value).

Hill climbing: Properties

- Complete?

Hill climbing: Properties

- Complete? No.

Hill climbing: Properties

- Complete? No. The algorithm is “greedy”, because he chooses a nearby good state without considering the further states.

Hill climbing: Properties

- Complete? No. The algorithm is “greedy”, because he chooses a nearby good state without considering the further states.
- Time?

Hill climbing: Properties

- Complete? No. The algorithm is "greedy", because he chooses a nearby good state without considering the further states.
- Time? $O(n^2)$.

Hill climbing: Properties

- Complete? No. The algorithm is "greedy", because he chooses a nearby good state without considering the further states.
- Time? $O(n^2)$. The number of neighbors of a queen is $n-1$, so the total number of adjacent states is $n*(n-1)$.

Hill climbing: Properties

- Complete? No. The algorithm is "greedy", because he chooses a nearby good state without considering the further states.
- Time? $O(n^2)$. The number of neighbors of a queen is $n-1$, so the total number of adjacent states is $n*(n-1)$.
- Space?

Hill climbing: Properties

- Complete? No. The algorithm is "greedy", because he chooses a nearby good state without considering the further states.
- Time? $O(n^2)$. The number of neighbors of a queen is $n-1$, so the total number of adjacent states is $n*(n-1)$.
- Space? $O(1)$.

Hill climbing: Properties

- Complete? No. The algorithm is "greedy", because he chooses a nearby good state without considering the further states.
- Time? $O(n^2)$. The number of neighbors of a queen is $n-1$, so the total number of adjacent states is $n*(n-1)$.
- Space? $O(1)$. Only the current state is stored.

Simulated annealing: implementation

- Simulated annealing chooses a random move, rather than the best one.

Simulated annealing: implementation

- Simulated annealing chooses a random move, rather than the best one.
- If the move improves the heuristic value, it will always be choose, otherwise it is accepted with some probability $p < 1$.

Simulated annealing: implementation

- Simulated annealing chooses a random move, rather than the best one.
- If the move improves the heuristic value, it will always be choose, otherwise it is accepted with some probability $p < 1$.
- Probability exponentially decreases with the worsening of the heuristic value and the lowering of temperature.

Simulated annealing: implementation

- Simulated annealing chooses a random move, rather than the best one.
- If the move improves the heuristic value, it will always be choose, otherwise it is accepted with some probability $p < 1$.
- Probability exponentially decreases with the worsening of the heuristic value and the lowering of temperature.
- If temperature decreases slowly, for the Boltzmann distribution propriety, all probability is centered on global minimum with a limit value of 1.

Simulated annealing: Properties

- Complete?

Simulated annealing: Properties

- Complete? No.

Simulated annealing: Properties

- Complete? No. Simulated annealing uses a probabilistic model. To reach completeness, theoretically, cooling should be infinitely slow.

Simulated annealing: Properties

- Complete? No. Simulated annealing uses a probabilistic model. To reach completeness, theoretically, cooling should be infinitely slow.
- Time?

Simulated annealing: Properties

- Complete? No. Simulated annealing uses a probabilistic model. To reach completeness, theoretically, cooling should be infinitely slow.
- Time? $O(t)$, where t is the cooling speed.

Simulated annealing: Properties

- Complete? No. Simulated annealing uses a probabilistic model. To reach completeness, theoretically, cooling should be infinitely slow.
- Time? $O(t)$, where t is the cooling speed.
- Space?

Simulated annealing: Properties

- Complete? No. Simulated annealing uses a probabilistic model. To reach completeness, theoretically, cooling should be infinitely slow.
- Time? $O(t)$, where t is the cooling speed.
- Space? $O(1)$.

Genetic algorithm: implementation

- Genetic search algorithms are built on the theory of natural selection: in each generation, the best individual are selected for reproduction.

Genetic algorithm: implementation

- Genetic search algorithms are built on the theory of natural selection: in each generation, the best individual are selected for reproduction.
  - In each generation, individuals reproduce with a probability that depends on their fitness value.

Genetic algorithm: implementation

- Genetic search algorithms are built on the theory of natural selection: in each generation, the best individual are selected for reproduction.
  - In each generation, individuals reproduce with a probability that depends on their fitness value.
- The recombination procedure occurs by randomly selecting a crossover point, at which the parents are divided to form their children.

Genetic algorithm: Properties

- Complete?

Genetic algorithm: Properties

- Complete? No.

Genetic algorithm: Properties

- Complete? No. Genetic algorithms are also based on probabilistic models.

Genetic algorithm: Properties

- Complete? No. Genetic algorithms are also based on probabilistic models.
- Time?

Genetic algorithm: Properties

- Complete? No. Genetic algorithms are also based on probabilistic models.
- Time? $O(km)$, where k is the maximum number of iterations and m is the population size.

Genetic algorithm: Properties

- Complete? No. Genetic algorithms are also based on probabilistic models.
- Time? $O(km)$, where k is the maximum number of iterations and m is the population size.
- Space?

Genetic algorithm: Properties

- Complete? No. Genetic algorithms are also based on probabilistic models.
- Time? $O(km)$, where k is the maximum number of iterations and m is the population size.
- Space? $O(m)$.

Genetic algorithm: Properties

- Complete? No. Genetic algorithms are also based on probabilistic models.
- Time? $O(km)$, where k is the maximum number of iterations and m is the population size.
- Space? $O(m)$. The current population is stored.