Computational Intelligence WS24/25

Exercise Sheet 2 — November 14th, 2024

Thomas Gabor, Maximilian Zorn, Claudia Linnhoff-Popien

1 Beaver goals and dreams

Consider a beaver in the final steps of building a dam. It just needs to add one more log at precisely the right angle. If it hits the right angle, the dam is now leak-proof and the water level (of the river flowing from behind the dam) rises. The beaver's behavior is thus given by a policy $\pi : \mathit{Level} \to \mathit{Angles}$ where $\mathit{Level} = [0; \infty) \subset \mathbb{R}$ is the water level in cm and $\mathit{Angles} = [0; 180] \times [0; 180] \subset \mathbb{R}^2$ are the horizontal and vertical angles in degrees at which the beaver tries to insert the final log into the dam.

(i) Assume that the environment is updated in discrete time steps and thus produces a sequence of states $\langle s_t \rangle_{1 \leq t \leq T}$ where $T \in \mathbb{N}$ is the episode length of the environment and $s_t \in \mathit{Level} \times \mathit{Angles}$ for all t. The beaver deems its work successful iff the water level keeps rising from some point in time on for at least three consecutive time steps and all future time steps afterwards. Give a goal predicate $\gamma : \langle \mathit{Level} \times \mathit{Angles} \rangle \to \mathbb{B}$ so that $\gamma(\langle s_t \rangle_{1 \leq t \leq T})$ holds iff the beaver deems its work successful.

Now, further assume that a beaver's policy for putting logs into dams is encoded by a parameter vector $\theta \in \Theta = \{A, C, G, T\}^5$ where A, C, G, T are arbitrary fixed symbols. We are given a fitness function $\phi : \Theta \to \mathbb{R}$ so that $\phi(\theta)$ encodes the time it takes a beaver with policy π_θ to put a log into a dam successfully, i.e., lower values of $\phi(\theta)$ are better.

(ii) We initialize a population $X \in \Theta$ with population size $|X| = 10$. A nice *variation* function should

- construct 5 new individuals based on random candidates from the original population X,
- not always generate completely new individuals, but have them based on the given population X, and
- be able to reach every point in the search space through iterated application.

Give a complete definition for a nice *variation* function.

(iii) Consider the following two definitions of a *selection* function:

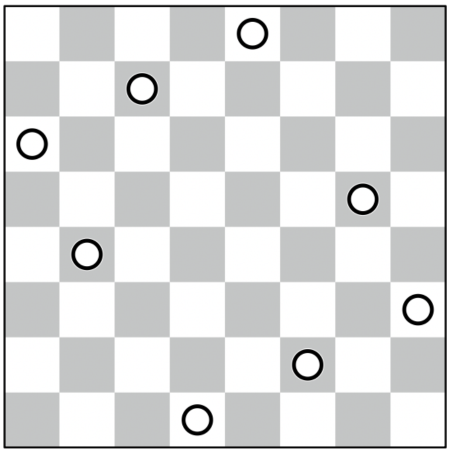
$$selection_1(X) = \begin{cases} selection_1\big(X \setminus \{\arg\max_{x \in X} \phi(x)\}\big) & \text{if } |X| > 10, \\ X & \text{otherwise,} \end{cases}$$

$$selection_2(X) = \begin{cases} selection_2\big(X \setminus \{x\}\big) & \text{for } x \sim X \text{ if } |X| > 10, \\ X & \text{otherwise.} \end{cases}$$

Briefly explain why both functions fulfill the definition for a *selection* function in the Evolutionary Algorithm as defined in the lecture? Which of these two functions is more directly useful in the context of optimization? State your argument.

2 N-Queens Optimization

In this exercise we will consider the **N-Queens**[1] problem as one prominent example domain for the optimization methods we have covered in the lecture.



(a) 8-Queens solution

```
function queens
Input n: int, k: int, board: array of array of boolean);
Output boolean
begin
    if k ≥ n then return true;
    for i = 0 up to n–1 do begin
        board[i][k] ← true;
        if not board[i][j] ∀j : 0 ≤ j < k
        and not board[i – j][k – j] ∀j : 0 < j ≤ min(k, i)
        and not board[i + j][k – j] ∀ j : 0 < j ≤ min(k, n – i – 1)
        and queens (n, k + 1, board)
        then return true;
        board[i][k] ← false;
    end
    return false;
end
```

(b) N-Queens algorithm pseudocode (backtracking approach)

(i) Define the N-Queens problem as a single-sample optimization process and give one example for each of the elements in the tuple $D = (\mathcal{X}, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$.

[1] https://en.wikipedia.org/wiki/Eight_queens_puzzle

(ii) Adjust your solution of the previous task and specify necessary parameters such that the *Simulated Annealing* (SA) Algorithm can be applied to the N-Queens Problem. Then, give a brief estimate on how SA would perform in comparison to other simple search algorithms like *Brute Force* and *Random Sampling* as the board size increases. How would the SA temperature parameter need to behave for increasingly larger boards?

(iii) How would *Simulated Annealing* compare to *Weighted Random Sampling* for the N-Queens Problem? Is the *locality* property given here?

Let's now move on from single-sample search to population-based search – in our case the *Evolutionary Algorithm* – and explore the N-Queens problem further.

(iv) Research and assign the terms **phenotype, genotype**, and **fitness** to this problem and give an example of each. How do these terms map to the definition of the optimization process D from Task *(i)*?

(v) Research and give one example each for generic **selection, mutation**, and **recombination** functions to optimize our N-Queens problem. You are free to choose your notation (e.g., code, pseudocode, formal notation etc.). Briefly explain your choices.

(vi) (Bonus 1) With the pseudocode algorithm given in Figure 1b), implement the backtracking solution algorithm to the N-Queens problem. The `queens` function is called with the number n of queens that defines the problem size, k = 0 indicating that the board should be filled starting from rank 0, and board being an $n \times n$ Boolean matrix that is initialized to false in all elements. If the function returns true, the problem can be solved. If the function returns false, the problem cannot be solved (the 3-Queens problem, for example, has no solution). (Hint: If you want to check the solutions, for smaller n the possible solutions are well documented.[2])

(vii) (Bonus 2) Instead of using the backtracking solution, try to optimize the N-Queens problem with an evolutionary algorithm approach. You could – for example – implement the evolutionary operators that you have defined above. How does the EA perform in comparison to the backtracking algorithm?

[2] `https://en.wikipedia.org/wiki/Eight_queens_puzzle`

3 Running Example: Vacuum World

With the implementation of the *Vacuum World* example from Exercise 1, extend the code to include the following elements:

(i) Introduce a target function τ of your choice, which is able to evaluate a given policy π with respect to a metric of your choosing (e.g., number of times the robot changed the room, number of times the robot vacuumed a dirty room successfully, etc.). You can freely decide whether your τ is to be minimized or maximized.

(ii) In addition to the random policy π_{rand} from Exercise 1, now also prepare a policy π_{target}, that performs better than the random baseline policy with respect to your previously implemented target function τ. Depending on your target function, you might need to adjust the observation space to include the necessary information.

(iii) Collect again a fixed number of actions and observations for each π_{rand} and π_{target}, then evaluate both with your τ function and compare.