Universidad de La Sabana Felix Mohr & Sergio Amortegui
Facultad de Ingenería 2026.1

Fundamentos de Inteligencia Artificial

- Sheet 2-

Environment & Rules. Use Python with `numpy` and the provided support modules (e.g., `connect4.connect_state`) as needed by the templates and grader. Keep the function and class signatures from the provided `.py` templates unchanged, since they are tested automatically on Gradescope. In particular, follow the required API for `GeneralConstructiveSearch` (including `expand`, `goal`, `better`, and `order`) and the task-encoding function names exactly. You will only see public test outputs.

Files to implement. `general_constructive_search.py`, `task_encodings.py`.
Do not rename files or functions (only remove the '.template').

Exercise 1 (General Constructive Search - 2 Points) Implement the general constructive search algorithm (with the three arguments *expand*, $\star$, and $\succ$) in the pre-defined class. Recall that all of them are *functions*. The initial node is now the empty assignment by default. In addition, enable a parameter `order` (string) that defines how OPEN is arranged (possible values are "bfs" for a list/queue and "dfs" for a stack/inverted list).

Make sure that the class has

1. a property `active` that indicates whether more solutions could be found,
2. a function `reset(self)` to refresh the algorithm configuration, and `step` that executes one logical iteration of the algorithm (if still active); this function should return a boolean that is `True` if a new best solution was found.
3. a property `best` that returns the best known solution (or the only one if $\succ$ is not provided).

Exercise 2 (Special Case of Fixed Number of Variables - 2 Points) Implement the `encode_problem` function that will derive *expand*, $\star$, and $\succ$ from a given tuple $(D, C, \succ)$, where

- D is a dictionary where keys are variables and values are lists with the respective domains,
- C is a function that receives a dictionary with a partial assignment of variables and returns `True` iff there exists an element in $\mathcal{X}$ that is compatible with the partial assignment and `False` otherwise, and
- $\succ$ compares two full assignments (dictionaries) and assigns `True` iff the first assignment is strictly better than second (and `False` otherwise).

Exercise 3 (Sudoku - 1.5 Points)
Implement the `get_general_constructive_search_for_sudoku(sudoku)` function that returns a pair `(search, decoder)`, where `search` is a `GeneralConstructiveSearch` object that can be used to solve the given Sudoku and `decoder` is a function that takes a final node from the search space and returns a 2D numpy array of a filled board. You can choose whether you want to use BFS or DFS.

In gradescope, the `step` function will be called in a loop until your algorithm declares itself inactive or 5s have passed; then your solution will be evaluated.

Hint: Check whether this is a problem with a fixed number of decisions or not. *Hint:* Treat given Sudoku clues as fixed assignments (e.g., singleton domains) so they are preserved from the start.

Exercise 4 (Job Shop - 1.5 Points)
Implement the `get_general_constructive_search_for_jobshop(jobshop)` function that returns a pair `(search, decoder)`, where `search` is a `GeneralConstructiveSearch` object that can be used to find an optimal job assignment to machines and `decoder` is a function that takes a final node from the search space and returns an integer list that contains at position i the index of the machine (between 0 and $m-1$) to which job i is assigned. `jobshop` is a tuple (m, d), where m is the number of machines and d is a list with duration times that the jobs will take on any machine. The performance of a solution is the time at which the *last* job is finished.

In gradescope, the `step` function will be called in a loop until your algorithm declares itself inactive or 10s have passed; then your solution will be evaluated.

Hint: Check whether this is a problem with a fixed number of decisions or not.

Exercise 5 (Connect-4 - 1.5 Points)
Implement the `get_general_constructive_search_for_connect_4(opponent)` function that returns a pair `(search, decoder)`, where `search` is a `GeneralConstructiveSearch` object that can be used to find a strategy for the yellow player that wins against the deterministic red enemy encoded in the opponent and `decoder` is a function that takes a final node from the search space and returns a list of movements of the yellow player that will lead to a win. `opponent` is a function that maps a state to an action (of the red player, who will also start the game). You can choose whether you want to use BFS or DFS.

In gradescope, the `step` function will be called in a loop until your algorithm declares itself inactive or 1m has passed; then your solution will be evaluated.

Hint: Check whether this is a problem with a fixed number of decisions or not.

Hint: In gradescope, you can import `ConnectState` from `connect4.connect_state`.

Exercise 6 (Tour Planning - 1.5 Points)
Implement the `get_general_constructive_search_for_tour_planning(distances, from_index, to_index)` function that returns a pair `(search, decoder)`, where `search` is a `GeneralConstructiveSearch` object that can be used to address the tour planning problem and `decoder` is a function that takes a final node from the search space and returns an integer list (indices of visited places). The entry i, j in the matrix `distances` is the distance between places i and j, and you ought to find the tour from `from_index` to `to_index` where summed distances are minimal, bearing in mind that values of 0 in `distances` mean that there is no route between these two places. You can choose whether you want to use BFS or DFS.

In gradescope, the `step` function will be called in a loop until your algorithm declares itself inactive or 3m have passed; then your solution will be evaluated.

Hint: Check whether this is a problem with a fixed number of decisions or not.