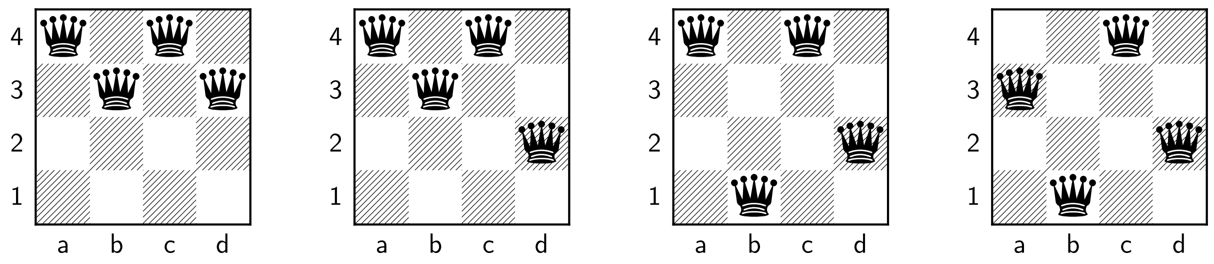
TP LSMM (Local Search and Min-Max algorithm)
Techniques of AI [INFO-H-410]
Correction

v1.0.0

Source files, code templates and corrections related to practical sessions can be found on the UV or on github (`https://github.com/iridia-ulb/INFOH410`).

Hill Climbing

Question 1. We want to solve the N-queens problem. Suppose you want to place N queens of a $N \times N$ chess board so that no queen can attack another queen.



a) What would be a good heuristic for this problem ? Calculate this heuristic for each of the above board.

b) How many board arrangments are there ?

c) What would be a good state representation for this puzzle ?

d) Given a clever state representation, how many board arrangments are there ?

Answer: (a) The number of pairs of attacking queens, h = 5, h = 3, h = 1, h = 0.
(b) There are 16 squares, $16 * 15 * 14 * 13 = 43680$.
(c) 4 digits, one digit represents the vertical position of a given queen on a row, using this state representation, already skrinks the space of solution since no 2 queens can be in the same column. With this state representation, the first board is <0,1,0,1>, the second <0,1,0,2>, the third <0,3,0,2>.
(d) If we say that we only want the permutations of 4 different digits from 0 to N (that way we already discard any position where 2 queens are in the same row or column) then there are $N! = 24$ arrangments.

Question 2. Implement exhaustive search in python (you can use the template (see page 1)) for the aforementioned puzzle.

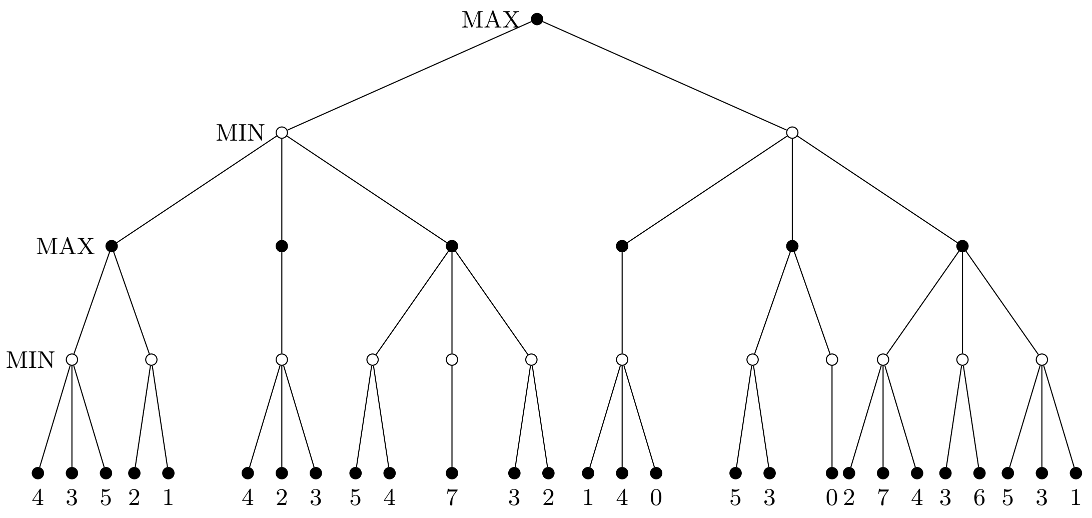
Answer: see github for implementation

Question 3. Implement (stochastic) Hill Climbing local search in python for this puzzle. Now compare your results with exhaustive search, what happens if N = 10, 15, 20 ? Why ?

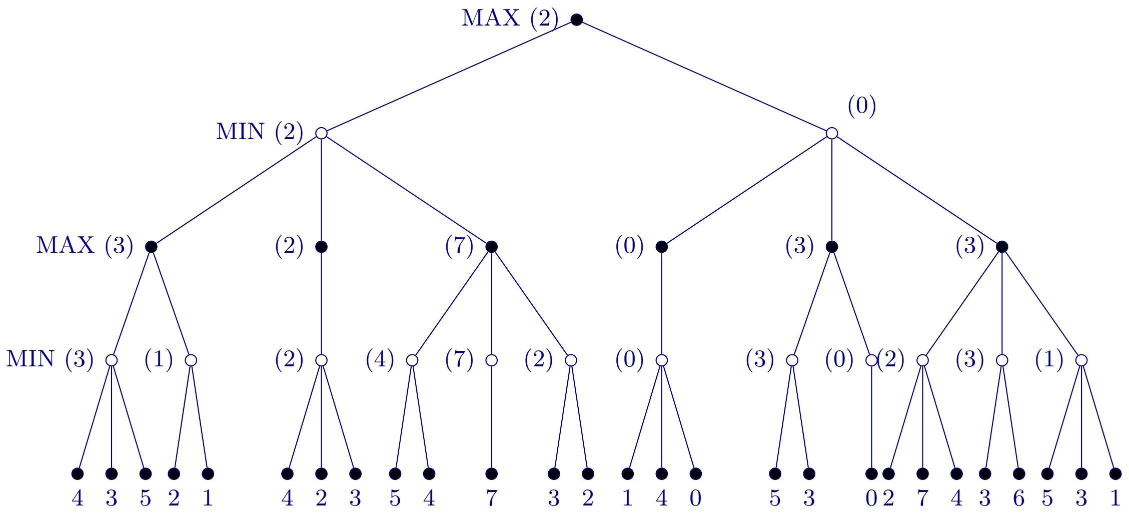
Answer: see github for implementation, exhaustive cannot work with N=12, because the search space is too big ($N!$), whereas hill climbing works because the smaller neighborhood makes the branching factor acceptable.

The Minimax algorithm

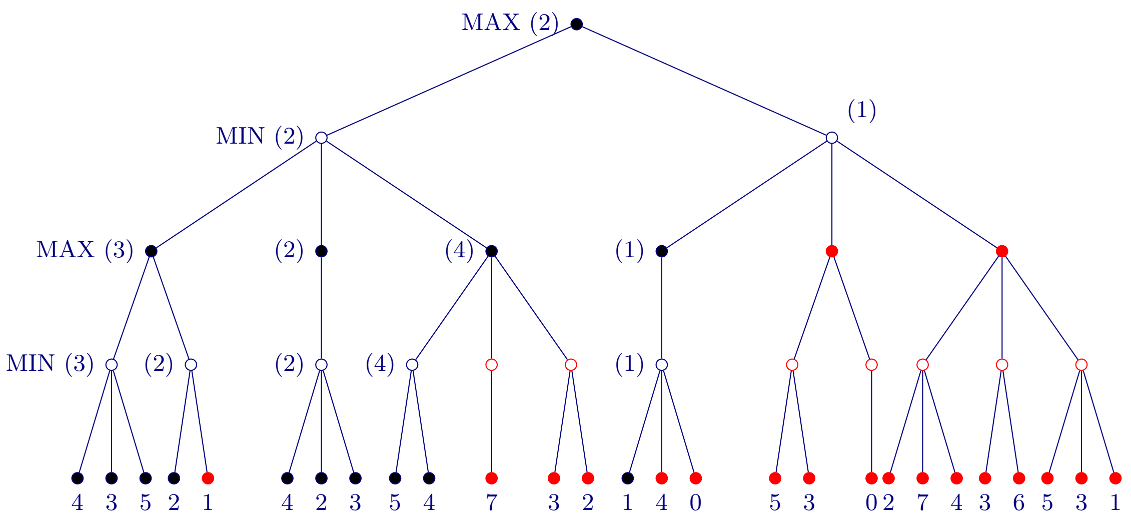
Question 4. Perform the minimax algorithm on the following tree, first without and then with $\alpha\beta$-pruning.



Answer: With regular minimax:



$\alpha\beta$-pruning saves 17 evaluations, unvisited nodes are labelled in red



Question 5. Implement the tic tac toe game using python so that you can play against an AI using the minimax algorithm (you can use the template (see page 1)).

X	X	O
O	X	X
O	X	O

Answer: see github for implementation

Local search applied to graphs: Kernighan-Lin

Kernighan-Lin is an algorithm to split a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ into two partitions P_1 and P_2 with a minimal cutisze. The cutsize c is computed as $\sum_{i \in P_1, j \in P_2} w(e_{i,j})$, where $e_{i,j}$ is an edge $\in \mathcal{E}$ connecting two vertices $i, j \in \mathcal{V}$ in different partitions and w the weight of said edge.

The algorithm will be swapping pairs of vertices across the partition boundary in order to gradually improve the cutsize, as described in aglo 1. The costs and gains it refers to are computed as follows:

- External cost of vertex $x \in P_1$: $E_x = \sum_{i \in P_2} w(e_{x,i})$
- Internal cost of vertex $x \in P_1$: $I_x = \sum_{i \in P_1} w(e_{x,i})$
- Gain of swapping x and y: $gain(x, y) = (E_x - I_x) + (E_y - I_y) - 2 \cdot w(e_{x,y})$

Question 6. Using the given template `q_bonus_kl_template.py`, implement the KL partitioning algorithm. Each line of the input file `graph.txt` describes an edge of the graph and is formatted as follows: `<edge weight> <vertex i> <vertex j>`

Answer: See file `q_bonus_kl.py`

Algorithm 1 Kernighan-Lin

```
 1: Partition randomly into $P_1$ and $P_2$                    ▷ Same size partitions
 2: Compute initial cutsize
 3: repeat
 4:     Unlock all vertices
 5:     Update costs $E_x$ and $I_x$
 6:     while Unlocked cells do
 7:         for all Unlocked vertices $x \in P_1$ do
 8:             for all Unlocked vertices $y \in P_2$ do
 9:                 Compute gain(x,y)
10:         Swap pair with highest gain                          ▷ Local search
11:         Lock those two vertices
12:     Accept $n$ first swaps leading to minimal cutsize       ▷ Hill climbing
13: until Cutsize is not improved
```