Session 4

Name Game [5 minutes]

- **<u>The Monty Hall problem</u>** – say your name, choose a door… do you want to switch?
    - for each student:
        - the calculator chooses a random number {1, 2, 3}
        - the student gets to choose a door {1, 2, 3}, I tell them the door it is not, and they get to switch or stay with their answer.
    - Tally the results and indicate the winners.

I. Questions (Homework/LISP/Class) [20 minutes]

- How did the homework go.
- Homework discussions/Topics not well covered?

II. AIMA [20 minutes]

- The first *real* project is coming up and you need to be able to use AIMA in order to effectively use your time.
    - You will be choosing partners, either in or out of this section.
    - Make the choice early and if you're having problems finding a partner, contact me and I'll try to pair people up!
- Another introduction to AIMA

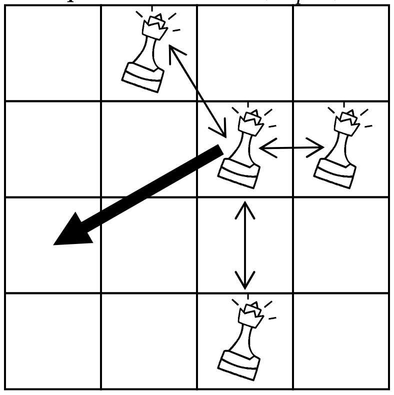
```
(defstruct (nqueens-permute-problem
              (:include nqueens-complete-problem) (:constructor create-nqueens-permute-problem))
  )

(defun make-nqueens-permute-problem (&key (n 8))
  "Returns an nqueens problem instance with all n queens placed
   randomly, one per column."
  (create-nqueens-complete-problem
   :n n :initial-state (random-nqueens-permute-state n)))

(defmethod goal-test ((problem nqueens-permute-problem) state)
  (zerop (h-cost problem state)))

(defmethod actions ((problem nqueens-permute-problem) state)
  "Generate the possible moves from a complete nqueens state.
   A move is simply the column and the new row for that queen."
  (let ((n (length state))
          (actions nil))
    ;;; For each column, generate all other rows but the current one
    (loop for col1 from 0 to (- n 2) do
      (loop for col2 from (+ col1 1) to (1- n) do
          (push (list col1 col2) actions)))
    actions))
```

```
(defmethod result ((problem nqueens-permute-problem) action state)
  "Return a new state with the specified queen moved to the specified square."
  (let ((outcome (copy-state state)))
    (swap-queens outcome (first action) (second action))
    outcome))

(defmethod h-cost ((problem nqueens-permute-problem) state)
  "Number of pairs of queens attacking each other."
  (let ((n (length state))
              (sum 0))
    (loop for i from 0 to (- n 2) do
      (loop for j from (1+ i) to (- n 1) do
              (let ((delta (- (aref state i) (aref state j))))
              (when (or (= delta 0) (= (abs delta) (- j i)))
                (incf sum)))))
    sum))

(defun random-nqueens-permute-state (n)
  "Return a random permutation state with n queens, one per column."
  (let ((state (make-sequence 'vector n)))
    (loop for i from 0 to (1- n) do
      (setf (elt state i) i))
    (loop for j from 0 to (1- n) do
       (swap-queens state j (+ j (random (- n j)))))
    state))

(defun swap-queens (list i j)
  (let ((temp (elt list i)))
    (setf (elt list i) (elt list j))
    (setf (elt list j) temp)
    list))
```

III. N-Queens continued

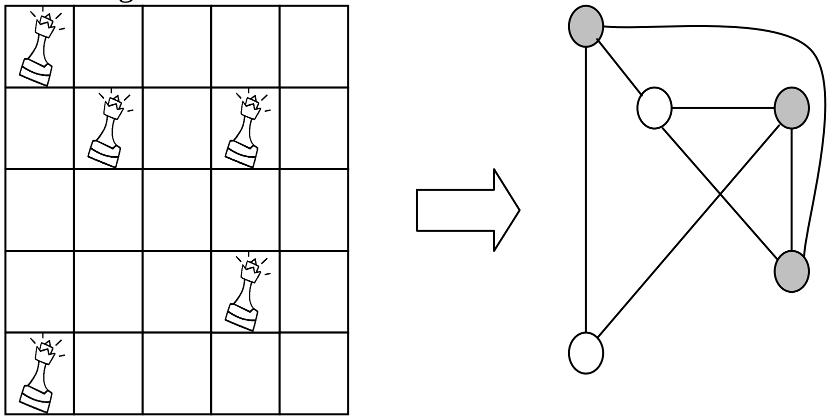
Heuristics Revisited

- Some of the heuristics we discussed for N-Queens last week were not admissible. Others were not efficiently computable.
  - o We must estimate the number of remaining moves to reach a solution.
  - o The # of pairs of queens heuristic, h_{pair}, is *NOT* admissible:

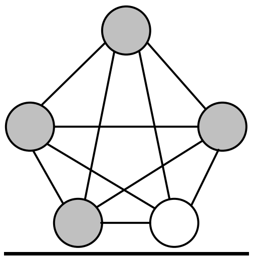


$h_{pair}=3$ but it only requires 1 move (or 2 if column and row shifts are separate) to reach a solution.

- **Conflict Graph** – identifies the queens that threaten one another. We make a graph where each queen is represented by a node and queens that can take each other have an edge between them.



  - o An admissible heuristic is τ (shaded gray) → the minimal number of queens to *totally disconnect* the conflict graph!
    - If a graph were totally disconnected there would be no conflicts – a solution. Thus, moving the τ nodes to non-conflicting locations is the minimal possible work. Hence this heuristic is admissible.
  - o
- **Maximal Clique**
  - o **Clique** – a subgraph that is fully connected (all nodes are connected to all other nodes in the subgraph).



- At least (k-1) of the nodes of a clique must be removed to disconnect the subgraph, thus, if we could find maximal cliques we could use them as a lower bound on the number of nodes required for disconnection.
- The idea of finding the "maximal clique" isn't the best approach.
  - N-Queens problems might not have a large maximal cliques.
  - Worse, finding the maximal clique is NP-Complete

- **Biconnected Components**.
  - It is possible to identify the biconnected components of the conflict graph in polynomial time.
    - biconnected components are the maximal subgraphs such that any two nodes in the subgraph lie on a common simple cycle.
  - The biconnected components must be separate.
  - For any biconnected component, the number of nodes required to disconnect it is simple to compute.
  - Still this biconnected components approach requires a good bit of effort and was non-trivial to come up with
- **Average number of edges per node**
  - Is this actually an admissible heuristic? surprisingly **YES**
    - PROOF Suppose graph *G* has *n* nodes and *m* edges. Thus it has an average of $A = m/n$ edges per node, or rather $m = A \cdot n$.
      - Suppose this heuristic is not admissible; then we must be able to *totally disconnect G* with $D < A$ nodes.
      - Thus, these *D* nodes must be the only connection the other $n - D$ nodes have, which can have at most $D(n-D)$ such connections.
      - Also, these *D* nodes can have at most $D(D-1)$ vertices between each other.
      - Thus the total edges that can be removed by removing the *D* nodes is

$$\begin{aligned} t_D &\le D(n-D) + D(D-1) \\ &= D(n-1) \\ &< A \cdot n = m \end{aligned}$$

      - But then total number of edges removable by the *D* nodes is less than the total number of edges!!!
      - Thus, we must remove at least *A* nodes to disconnect *G*.

- The following combines the average edges per node with the fact that we don't need to separate components of the graph that are already disconnected:
  - Find the *c* connected components of *G* and calculate the average number of edges per node in each component A_c.
  - Now the heuristic of our state is given by:

$$h(G) = \sum_{i=1}^{c} A_i$$

  - This function is *very* simple to calculate for each state.

IV. Constraint Statisfaction

N-Queens Again???

- As we saw in class, N-Queens fits very well into the CSP formulation.
  - What are our constraints?
    - In plain English:
      We must place N queens on a $N \times N$ chess board such that no queens can take each other. Thus, (1) no queen can share the same column (2) no queen can share the same row (3) no queen can share any diagonal.
    - A more mathematical explanation
      - The first constraint is implicit, we give each queen a variable $Q_i \in \{0,1,\dots,N\}$ representing the row of the *i*-th column in which the queen is placed.
      - The second constraint is explicitly represented by restricting ourselves to the space of permutations; that is, $(Q_0, Q_1, \dots, Q_N) \in \pi(N)$.
      - The final constraint is difficult. For the *i-th* and *j-th* queen, they must not share the same diagonal, or rather, $|Q_i - Q_j| \neq |i - j|$ since the *i*-th and *j*-th columns are of distance $|i - j|$ apart.
    - A very mathematical version

$$\begin{aligned} &\text{Find} \quad (Q_0, Q_1, \dots, Q_N) \in \pi(N) \\ &s.t. \quad \forall i < j \quad |Q_j - Q_i| \neq j - i \end{aligned}$$

  - Go through worksheet VERY thoroughly emphasizing forward checking and arc-consistency.