1.

a) N robots can be placed on any of the (N x N) squares. These N robots are identifiable individually based on their starting position on the grid. The state space can be calculated as a mathematical problem of choosing N squares out of the N^2 squares on the grid and then calculating all possible permutations of the placement of the N robots on the N chosen squares.

i.e. state space $= {}^{N*N}C_N \cdot N!$

$$= \frac{N^2!}{N!\,(N^2 - N)!} \cdot N!$$

$$= \mathbf{\frac{N^2!}{(N^2 - N)!}}$$

b) As the upper bound, any robot can choose to move in one of four directions or stay put. More than one robot may stay put. So the problem may be divided into that of :

- choosing some or all robots out of N which will move in order to generate the next step. This is a combinations problem computed by ${}^{N}C_{i}$.
- Thereafter the robots that are chosen to move have a maximum of 4 directions to move in, one of them possibly being jumping over an adjacent stationery robot; the 5th one, that of staying put, having been considered in the combinations calculations above.

Thus,

$$\text{Branching Factor}_{UPPER} = {}^{N}C_1 .(3.1) + {}^{N}C_2 .(3.2) + {}^{N}C_3 .(3.3) + {}^{N}C_3 .(3.3) + \ldots\ldots + {}^{N}C_N .(3.N)$$

$$= \sum_{i=1\ldots N} {}^{N}C_i \cdot 4i$$

$$= \sum_{i=1\ldots N} \frac{N!}{i!\,(N-i)!} \cdot 4i$$

$$= 4 \cdot N! \cdot \sum_{i=1\ldots N} \frac{i}{i!\,(N-i)!}$$

$$= 4 \cdot N! \cdot \frac{2^{N-1}}{(N-i)!}$$

$$= \mathbf{4 \cdot N \cdot 2^{N-1}}$$

For an average case, the branching factor would work out to be approximately the same as the upper bound specified above, since the **problem is under-constrained**.

$$\textbf{Branching Factor}_{AVG} \approx \textbf{Branching Factor}_{UPPER}$$

c) Admissible heuristic for this problem is the distance of the robot from it's goal location. The given current position of the robot is (x_i, y_i) and the goal location is (N-i+1, N). Considering that the robot has 4 degrees of freedom (north, south, east and west), it's distance from the goal position at any given time is the sum of horizontal/x-axis distance and vertical/y-axis distance. This would give the number of moves it has to make to get from current position to goal position.

$$\mathbf{H_i = | N - i + 1 - x_i | + | N - y_i |}$$

d) Admissibility of heuristics for moving all N robots to their goal positions:

- **The sum of H_i for i=1 to N is not an admissible heuristic**. This heuristic will result in an overestimation of the cost to reach the goal state. It can be used as a heuristic when

only one out of the N robots can move at one time to generate a new state. In this case, since one or more robots (including all) may move in order to generate a new state , the optimal cost to reach the goal configuration will definitely be less than the sum of all H_i, defined above. Thus except for the case when only one robot is allowed to move at a time, this heuristic will definitely be an overestimation of the distance from the goal state.

- **max{H_1, . . . , H_N} is an admissible heuristic.** The maximum value that H_i can have is when the i^{th} robot is at it's initial position. In any configuration the maximum of all H_i will be the heuristic function value for the robot which is farthest from it's goal position. Irrespective of the current positions of the other robots, it will take atleast these many (max. of $\mathbf{H_i}$'s) steps to reach the goal state. Thus the maximum will be equal to the actual number of steps required to complete the problem.
- **min{H_1, . . . , H_N} is an admissible heuristic.** The justification for this extends based on that of the above heuristic option. Only in this case the minimum of all H_i will be less than the actual number of steps required to complete the problem. Hence even this does not over-estimate the cost and is admissible.

2. Selecting **most constrained variables** helps **prune majority of the search tree/graph** initially. The part which is pruned contains nodes which contribute to paths which will not lead to valid solutions, since these are parts pruned due to constraint violation involving the most constrained variable selected. This thereby **reduces the time** and/or space requirements of the search. Selecting the **least constraining value** for the selected most constrained variable ensures that **unselected variables get more flexibility in choosing their values** and a more **comprehensive search tree/graph traversal** is performed. This improves chances of **finding a solution quicker** in cases when only one solution is required.

3.

a) See python code submitted.

b) Minimum constraints method fails when it has to **hill climb at local minimum**. Given the **limited number of iterations**, to recover from conflicts by finding cases of minimum conflicts for a randomly generated sequence of variables considered initially, the chances of the algorithm finding a solution even if it recovers from the local minimum are very bleak. Local searches are also known to fail for **heavily constrained problems**.

With **random restarts incorporated**, the chances of finding a solution improve with the number of restarts allowed, but with a **trade-off on the search time** front.

c) I could not get a solution for a magic square of size 6.

CSP is not advisable with a heavily-constrained problem. One would avoid constraints with a lot of variables when it comes to CSPs. CSP performance degrades when faced with big

N-ary constraints as opposed to binary constraints. The magic squares problem has a lot of constraints involving variables proportional to the size of the square size. Also CSP's fail to generate solutions in feasible time as the square size increases i.e as the state space increases exponentially. This may be the case why we can't generate a solution for square size 6 on.