**6.2**

**a.**

**Solution A:**

There is a variable corresponding to each of the n2 positions on the board.

**Solution B:**

There is a variable corresponding to each knight.

**b.**

**Solution A:**

Each variable can take one of two values, {occupied, vacant}

**Solution B:**

Each variable’s domain is the set of squares.

**c.**

**Solution A:**

Every pair of squares separated by a knight’s move is constrained, such that both cannot be occupied.

Furthermore, the entire set of squares is constrained, such that the total number of occupied squares should be k.

**Solution B:**

Every pair of knights is constrained, such that no two knights can be on the same square or on squares separated by a knight’s move.

Solution B may be preferable because there is no global constraint, although Solution A has the smaller state space when k is large.

**d.**

Any solution must describe a complete-state formulation because we are using a local search algorithm.

* For simulated annealing, the successor function must completely connect the space
* For random-restart, the goal state must be reachable by hill climbing from some initial state

Two basic classes of solutions are:

**Solution C:**

Ensure no attacks at any time.

Actions are to

* remove any knight
* add a knight in any unattacked square
* move a knight to any unattacked square

**Solution D:**

Allow attacks but try to get rid of them.

Actions are to

* remove any knight
* add a knight in any square
* move a knight to any square