Nikil Pancha AI Project 1

1) The code has a fairly simple design. The Solver class contains the search methods, the Board class contains all of the operations that can be done on a board, and App contains the command line interface. I decided to represent the positions of tiles using an array of bitboards, which may or may not be the best way, but I was already familiar with it and it is very clean to change the position of tiles. The list that holds the tiles holds the position of b at index 0, 1 at 1, 2 at 2 and so forth. The value of b at the goal state is 1, 1 is at 1<<1, 2 is at 1<<2, etc. The rest of the Board class contains various operations to do on a board and can store the previous positions, but when comparing equality of positions in the solver class, it was easier to just compare the list of tiles (called pieces), because although the rest of the information is important, when checking for the goal states, all that matters is what the current position is. The beam heuristic was chosen to h2 from the A* search. After experimenting with linear combinations of h1, h2, and blank tile distance, h2 seemed to perform the best.
2) The key points that must be shown for the A-star search is that if there is a solution, it finds it, and that if there is no solution, it reports so. If tiles 2 and 1 are swapped but no other ones are, there is no reachable solution, so this is adequate to demonstrate that the algorithms will report the lack of a solution. The maxNodes limit works, as is seen by the limiting of maxNodes to 10, and observing that all searches are cut off before finding a solution. Around 10k tends to be a reasonable value for maxNodes, because after that, the beam search tends to converge, and A* will nearly never take that many steps. The file of commands can be executed by running "java App commands.txt", assuming commands.txt is in the same folder as App.class. I demonstrate that the searches find solutions, and, with a shorter scramble, that the solutions A* with both heuristics and beam provide are correct.
3)
    a. (using beam width = 100, sampling about 0.1%)
    maxNodes: proportion of states solvable
    1000: 0.03
    2000: 0.65
    4000: 0.97
    8000: 0.97
    16000: 0.97
    32000: 0.97
    64000: 0.97
    Once maxNodes exceeds 8000 or so, the proportion of solvable states does not

change much, meaning that when k=100, the beam search tends to converge in about 8000 nodes.

b. H2 is better than h1, as it is dominant. If there are n misplaced tiles, h1 will give a heuristic value of n, while h2 will give a value of some amount at least n. Unless each tile is adjacent to its goal position, h2 will give a value of >n, making it dominant to h1 and therefore a better heuristic.

c. Both A* searches find solutions of an optimal length, while the beam search will find solutions that are slightly longer. The beam search solutions are not a lot longer because they tend to converge to either the solution or not finding a solution before exploring many nodes. On shorter solutions, the beam search will find a much longer solution because 15 random moves are applied to all but one board (the starting board) at the beginning to randomize the start state.

d. Both A* searches find solutions when they exist. This means that of all reachable states, 100% are solvable, or of all states, including invalid ones, 50% are solvable. For the beam search, with a beam width of 250, almost all (>99%) puzzles are solvable. With a width of 100, about 97% are solvable. With a width of 50, about 75% are solvable. With a width of 10, about 14% are solvable.

4) A. The A* search using h2 seems to be the best for solving this particular problem. It finds an optimal solution very quickly, unlike using h1, which explores many more nodes, or beam, which guarantees neither a solution nor optimality. The beam search is superior in terms of space complexity but the space is small enough that the completeness and optimality of A* with h2 make up for the extra space taken. If solving a much larger one, such as a 120-puzzle or 80-puzzle, it is likely that the beam search would be the best choice because the A* might not be able to hold enough nodes in memory, and with a wide enough beam, the beam search would very likely converge.

B. A* search was slightly simpler to implement than the beam search because there was no need to generate the initial random states. Other than that, they were very similar in difficulty. From a testing standpoint, A* is easier to test because it is guaranteed to find a solution, so if no solution is found from a valid state, it is certainly an error in the code, while the beam search could have not found a solution even if one existed. One interesting observation is that, on average, A* with h2 explored fewer nodes when finding solutions of length 32 than those of length 31.

Statistics on nodes visited in finding solution by different methods

List of average nodes visited, by length of solution (h2)
[2, 3, 4, 5, 6, 7, 9, 11, 14, 18, 25, 32, 45, 60, 85, 112, 160, 208, 297, 386, 554, 728, 1051, 1395, 2045, 2718, 4001, 5340, 7742, 11156, 10091]

h1 (only looked at 1/10 the space)
[-1, -1, 4, 5, 6, 9, 13, 16, 25, 38, 60, 93, 150, 224, 384, 533, 919, 1324, 2083, 3427, 5409, 7955, 14380, 17969, 33053, 37988, 69565, 85115, 117088, 145942]

beam (only looked at about 1/10 the possible states, k=100, only included if a solution was found)
[-1, -1, -1, -1, 139, 180, 261, 364, 457, 562, 654, 762, 856, 963, 1057, 1165, 1257, 1362, 1457, 1564, 1651, 1758, 1836, 1944, 2016, 2064, 2108, 2070, 1879, 1676, 1650, 1581, 1536, 1559, 1615, 1719, 1789, 1886, 2003, 2105, 2205, 2338, 2404, 2559, 2676, 2827, 2875, 3011, 3000, 3064, 3197, 3264, 3358, 3467, 3559, 3668, 3754, 3863, 3957, 4069, 4153, 4277, 4355, 4477, 4551, -1, 4751, 4851]