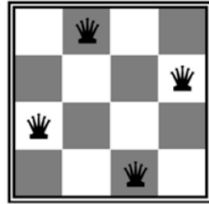


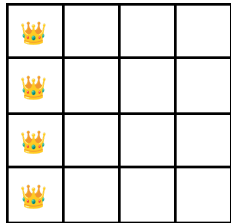
Problem 1: Local Search (16 points)

We will use local search to solve the 4-queens problem. To simplify representation, a state will be represented by a set of 4 coordinate tuples, one for each queen. Following NumPy indexing, the top left cell has coordinates (0,0) and the bottom right cell has coordinates (3,3). So the example state shown below is $\{(0, 1), (1, 3), (2, 0), (3, 2)\}$.



We will define neighboring states as those where exactly one queen is moved to a different column. We will use the “min conflicts” evaluation function h , which counts the number of different queen pairs that are in the same row, column, or diagonal.

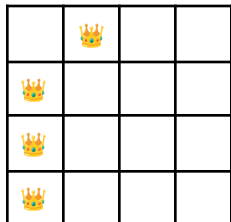
1. (3 pts) Consider the initial state with all queens in the left column, or $\{(0, 0), (1, 0), (2, 0), (3, 0)\}$. What is its h value? Remember to count all pairs of queens in conflict. Are there any neighboring states with the same or greater h value? What kind of feature would this state represent in the state space landscape?



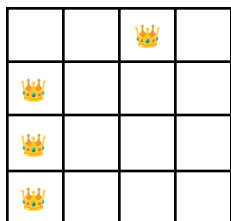
$$h = 3 + 2 + 1 = 6$$

As long as only one queen is moving to a different column in the same row, it is considered a neighboring state.

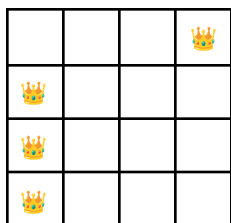
No neighboring state with the same or greater h value. (Moving any queen out of the leftmost column will resolve three conflicts and create at most two conflicts). Thus the initial state represents a local maxima, and therefore a global maxima (because h cannot be greater than 6).



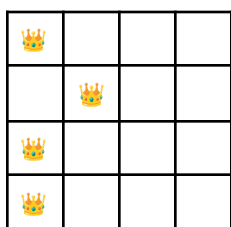
$$h = 2 + 1 + 1 = 4$$



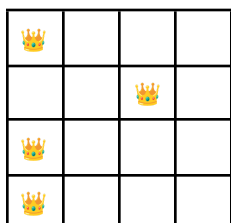
$$h = 2 + 1 + 1 = 4$$



$$h = 2 + 1 + 1 = 4$$



$$h = 2 + 1 + 2 = 5$$



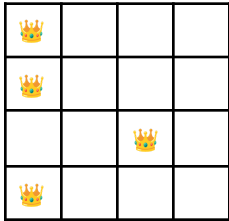
$$h = 2 + 1 + 1 = 4$$



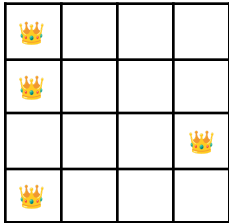
$$h = 2 + 1 = 3$$



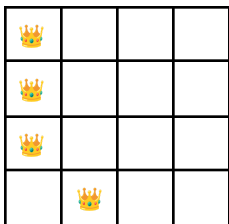
$$h = 2 + 1 + 2 = 5$$



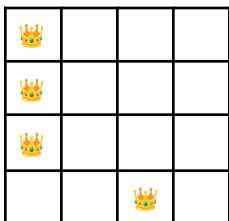
$$h = 2 + 1 + 1 = 4$$



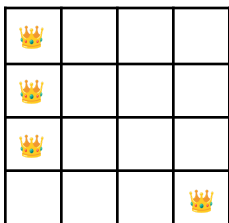
$$h = 2 + 1 = 3$$



$$h = 2 + 1 + 1 = 4$$







$$h = 2 + 1 + 1 = 4$$







$$h = 2 + 1 + 1 = 4$$

2. (6 pts) Starting from the above initial state, trace the hill-“descent” procedure in which we repeatedly move to a neighboring state with the lowest h value among all neighbors, until we can improve no further. Write out the state and h value after each iteration.





$$h = 3 + 2 + 1 = 6$$

State: $\{(0, 0), (1, 0), (2, 0), (3, 0)\}$





$$h = 2 + 1 = 3$$

State: $\{(0, 0), (1, 3), (2, 0), (3, 0)\}$

$$h = 1$$

State: $\{(0, 1), (1, 3), (2, 0), (3, 0)\}$





			
			
			
			

$$h = 0$$

State: $\{(0, 1), (1, 3), (2, 0), (3, 2)\}$

3. (3 pts) Now consider the initial state $\{(0, 1), (1, 3), (2, 2), (3, 0)\}$. What is its h value? Are there any neighboring states with the same or lower h value? What kind of feature would this state represent in the state space landscape?





Initial state: $\{(0, 1), (1, 3), (2, 2), (3, 0)\}$

$h = 1$

Neighboring state with the same h value:

$\{(0, 1), (1, 3), (2, 0), (3, 0)\}$

$h = 1$

No neighboring state with lower h value.

This is the flat local minimum where all neighboring states has the same or higher h value.

4. (4 pts) Starting from the above initial state and now allowing sideways moves, trace a hilldescent outcome (as in part 2) that results in a consistent solution in the fewest number of iterations. Explain whether it is possible to never find a solution if we make no other changes, and how a simple limit on the number of sideways moves may or may not change this outcome.

Initial state:

	👑		
			👑
		👑	
👑			

$h = 1$

A sideways move (parts 3&4) refers to moving to a neighbor state with the same h value.

Initial state:

	👑		
			👑
		👑	
👑			

$h = 1$

	👑		
			👑
👑			
👑			

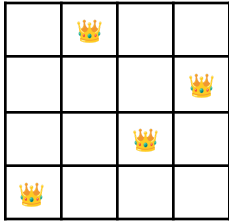
$h = 1$

	👑		
			👑
👑			
		👑	

$h = 0$

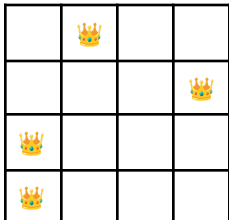
It is possible that we might never find a solution
The algorithm might enter a loop:

Initial state:



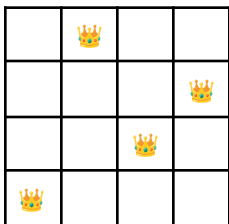
$h = 1$

->



$h = 1$

->



$h = 1$

...

By setting a limit on the number of sideways moves, there are two possible outcomes:

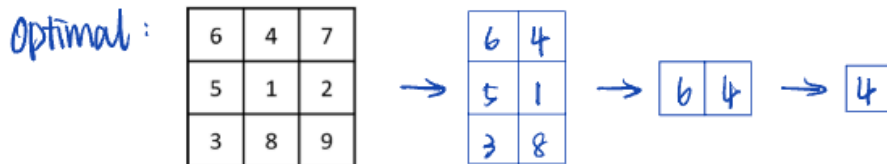
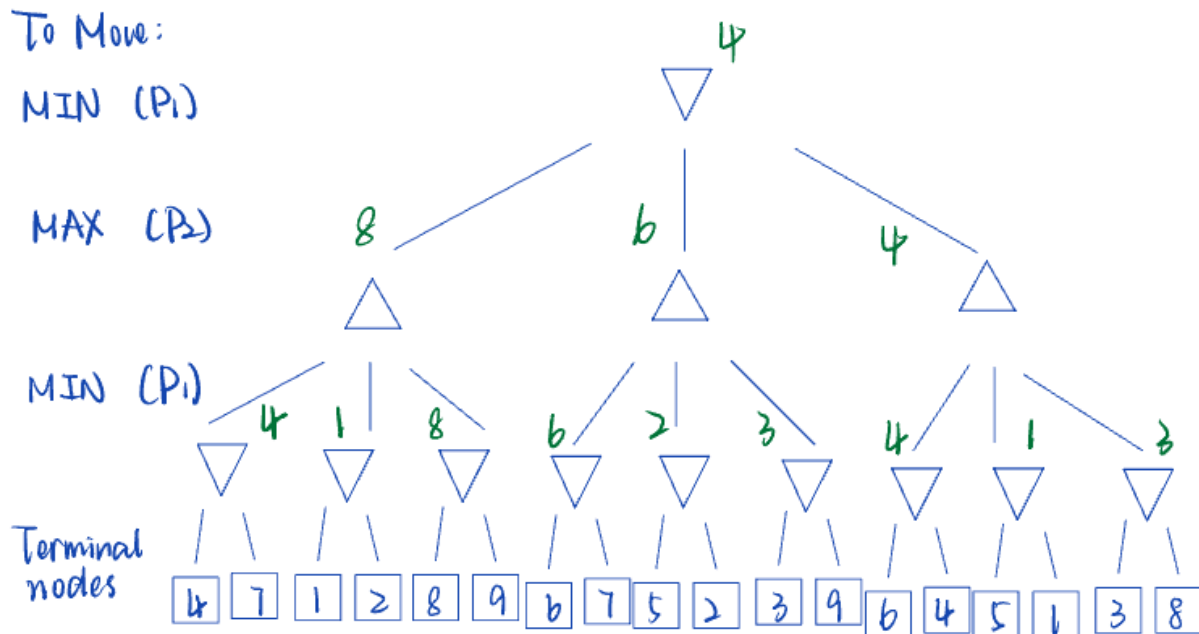
1. Prevent the algorithm from getting stuck in an infinity loop and then find a solution
2. The limit might be too low that we're not able to find an optimal solution (the optimal solution might require more sideways moves to reach).

Problem 2: Game Tree Search (24 points)

Two agents are playing a game using the following scoreboard. Player 1 (P1) first eliminates a column from the board. Player 2 (P2) next keeps a row from the current board. Finally, P1 selects one of the two values remaining as the game score. P1's objective is to minimize the score, while P2's objective is to maximize it.

6	4	7
5	1	2
3	8	9

- (8 pts) Draw the full game tree, clearly indicating MAX nodes, MIN nodes, and terminal nodes and their values. Please order the nodes in each ply corresponding to rows going from top to bottom or columns going left to right. Finally, label all MAX and MIN nodes with their minimax values. What is the optimal sequence of actions taken by each player?



To Move:

MIN (P1)

$$\begin{array}{|c|c|c|} \hline 6 & 4 & 7 \\ \hline 5 & 1 & 2 \\ \hline 3 & 8 & 9 \\ \hline \end{array}$$

4

MAX (P2)

8

$$\begin{array}{|c|c|} \hline 4 & 7 \\ \hline 1 & 2 \\ \hline 8 & 9 \\ \hline \end{array}$$

6

$$\begin{array}{|c|c|} \hline 6 & 7 \\ \hline 5 & 2 \\ \hline 3 & 9 \\ \hline \end{array}$$

4

$$\begin{array}{|c|c|} \hline 6 & 4 \\ \hline 5 & 1 \\ \hline 3 & 8 \\ \hline \end{array}$$

MIN (P1)

4

$$\begin{array}{|c|c|} \hline 4 & 7 \\ \hline \end{array}$$

1

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array}$$

8

$$\begin{array}{|c|c|} \hline 8 & 9 \\ \hline \end{array}$$

6

$$\begin{array}{|c|c|} \hline 6 & 7 \\ \hline \end{array}$$

2

$$\begin{array}{|c|c|} \hline 5 & 2 \\ \hline \end{array}$$

3

$$\begin{array}{|c|c|} \hline 3 & 9 \\ \hline \end{array}$$

4

$$\begin{array}{|c|c|} \hline 6 & 4 \\ \hline \end{array}$$

1

$$\begin{array}{|c|c|} \hline 5 & 1 \\ \hline \end{array}$$

3

$$\begin{array}{|c|c|} \hline 3 & 8 \\ \hline \end{array}$$

Terminal nodes

$$\begin{array}{|c|c|} \hline 4 & 7 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 8 & 9 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 6 & 7 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 5 & 2 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 3 & 9 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 6 & 4 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 5 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 3 & 8 \\ \hline \end{array}$$

Optimal:

$$\begin{array}{|c|c|c|} \hline 6 & 4 & 7 \\ \hline 5 & 1 & 2 \\ \hline 3 & 8 & 9 \\ \hline \end{array}$$

→

$$\begin{array}{|c|c|} \hline 6 & 4 \\ \hline 5 & 1 \\ \hline 3 & 8 \\ \hline \end{array}$$

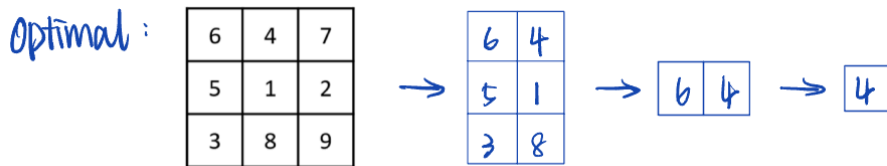
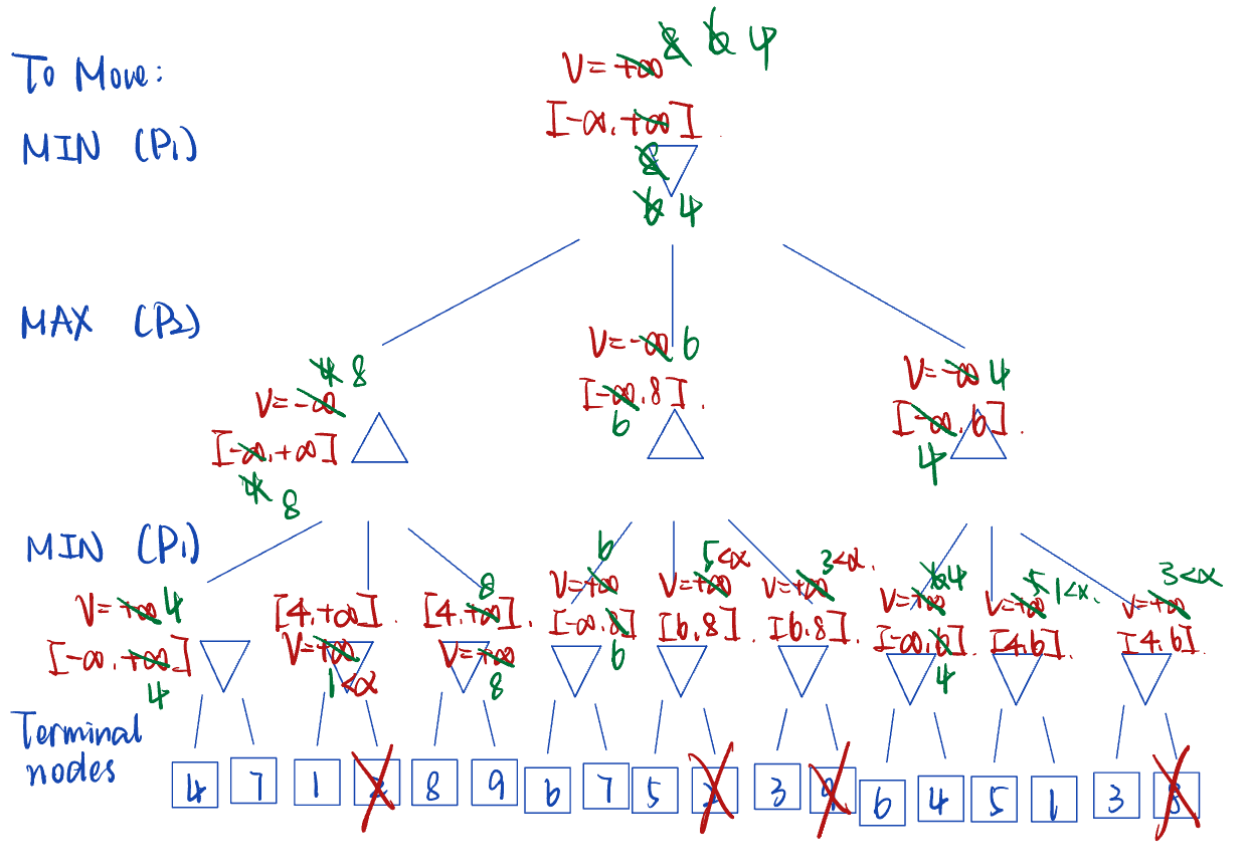
→

$$\begin{array}{|c|c|} \hline 6 & 4 \\ \hline \end{array}$$

→

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

2. (8 pts) We perform alpha-beta search on the game tree following a depth-first, left to right order. Identify all nodes that are pruned during the search procedure (you can refer to each as "node i in the ply j ", both starting from 1). For each pruned node, give the α and β values of their *parent* node, as well as the value comparison that resulted in the pruning.



Nodes that are pruned:

node i = 4 in the ply j = 4 (node value: 2)

alpha value: 4

beta value: +infinity

value comparison that resulted in the pruning: $1 \leq \alpha = 4$

node i = 10 in the ply j = 4 (node value: 2)

alpha value: 6

beta value: 8

value comparison that resulted in the pruning: $5 \leq \alpha = 6$

node i = 12 in the ply j = 4 (node value: 9)

alpha value: 6

beta value: 8

value comparison that resulted in the pruning: $3 \leq \alpha = 6$

node i = 18 in the ply j = 4 (node value: 8)

alpha value: 4

beta value: 6

value comparison that resulted in the pruning: $3 \leq \alpha = 4$

3. (4 pts) Now suppose that P2 plays randomly rather than optimally. Regardless of what P1 does, P2 keeps the first row with probability 25%, second row with probability 50%, and third row with probability 25%. Compute the new expected score of the game (show your calculations). Briefly explain why P1 would never change their initial strategy regardless of P2's row selection probabilities.

If P1 eliminates the first column,

$$\text{Expected score} = 25\% * 4 + 50\% * 1 + 25\% * 8 = 3.5$$

If P1 eliminates the second column,

$$\text{Expected score} = 25\% * 6 + 50\% * 2 + 25\% * 3 = 3.25$$

If P1 eliminates the third column,

$$\text{Expected score} = 25\% * 4 + 50\% * 1 + 25\% * 3 = 2.25$$

P1's initial strategy: eliminate the third column.

P1 would never change their initial strategy because regardless of P2's row selection probabilities, the expected score for P1 is the lowest (optimal) when P1 chooses to eliminate the third column.

4. (4 pts) Now suppose that rather than solving the game through a full search, we use an evaluation function to estimate the game score immediately after P1 eliminates a column. We consider two such functions: a) Minimum of remaining matrix values, and b) Average of remaining matrix values. Compute the expected game score and best action for P1 for each function.

a) Minimum of the remaining matrix value:

If P1 eliminates the first column, score = 1

If P1 eliminates the second column, score = 2

If P1 eliminates the third column, score = 1

Best action for P1: either eliminate the first column or eliminate the second column

b) Average of remaining matrix values:

If P1 eliminates the first column, score = $(4+7+1+2+8+9)/6 = 5.167$

If P1 eliminates the second column, score = $(6+7+5+2+3+9)/6 = 5.333$

If P1 eliminates the third column, score = $(6+4+5+1+3+8)/6 = 4.5$

Best action for P1: eliminate the third column

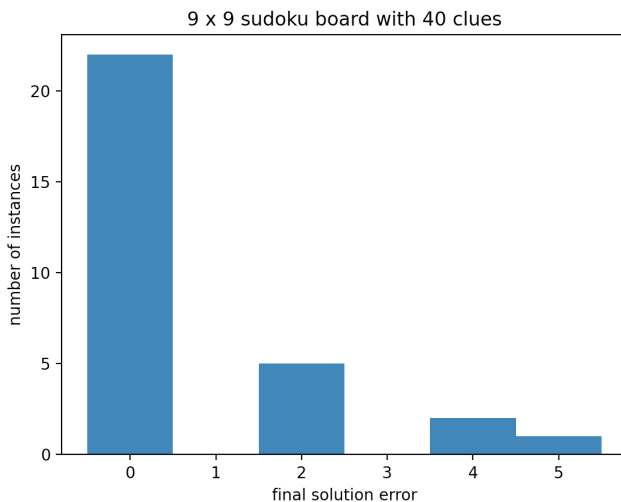
Problem 3:

Analysis (12 points) Your main goal will be to solve 9×9 sudoku puzzles with $c = 40$ clues. You can run `python sudoku.py` with the `-n` and `-c` options set to these values. Without setting any other parameters, the default values of `startT` and `decay` are 100 and 0.5, respectively. You will likely see the solver doing very poorly on most instances with these parameters.

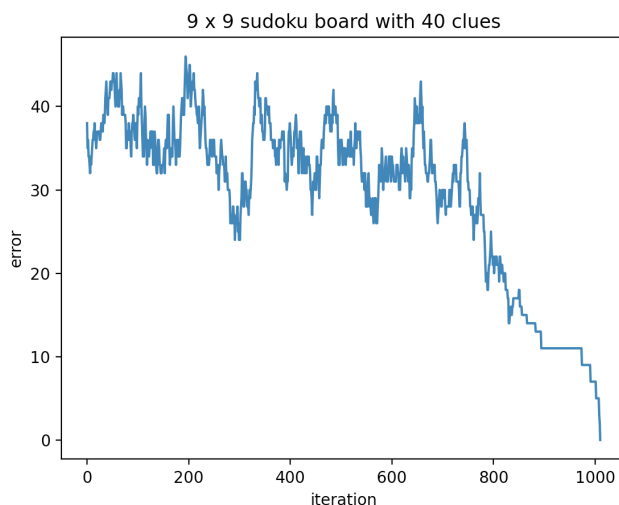
1. Use the `-d` option to experiment with the decay value. Specifically, find a value such that when you set the option `-b 30` (batch of 30 runs), at least half of the instances have a final error of 0. Explain why the new value is able to improve the performance of simulated annealing. Also include two plots, one showing the error history of an individual search that successfully finds a solution, and one showing a final error histogram of a batch of 30 searches.

Set decay value to 0.995

```
python sudoku.py -n 9 -c 40 -b 30 -d 0.995
```



```
python sudoku.py -n 9 -c 40 -d 0.995
```



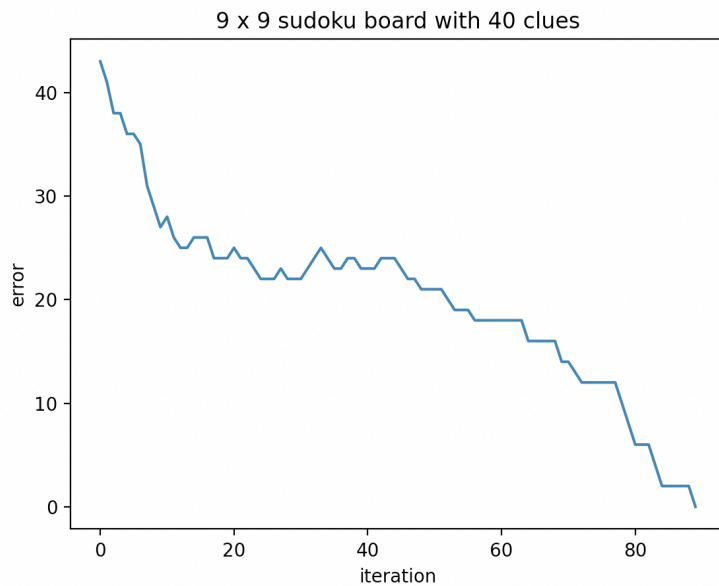
Sudoku puzzle:

```
[[9 5 7 1 0 3 4 0 0]
[0 1 2 8 4 0 0 0 5]
[6 8 0 0 0 0 0 0 0]
[0 3 8 0 0 4 1 7 9]
[0 0 1 0 8 0 0 0 0]
[4 6 5 0 0 7 8 0 3]
[0 0 3 2 0 8 0 0 4]
[0 2 0 0 0 5 3 0 0]
[5 0 0 7 3 0 6 8 2]]
[[9 5 7 1 2 3 4 6 8]
[3 1 2 8 4 6 7 9 5]
[6 8 4 5 7 9 2 3 1]
[2 3 8 6 5 4 1 7 9]
[7 9 1 3 8 2 5 4 6]
[4 6 5 9 1 7 8 2 3]
[1 7 3 2 6 8 9 5 4]
[8 2 6 4 9 5 3 1 7]
[5 4 9 7 3 1 6 8 2]]
Final errors: 0
```

Explain why the new value is able to improve the performance of simulated annealing.
Setting the decay value higher (0.995) results in a slower decrease in temperature. The algorithm is now able to explore more extensively, rather than getting stuck at a local optimum. If the decay value is too low, the algorithm might quickly settle down to a local optimal solution rather than a global optimal solution.

2. Keeping the decay value that you found above, experiment with the startT parameter using the -s option. Show three individual search result plots with this value set to 1, 10, and 100. Explain how the different values of startT affect the progression of the search.

```
python sudoku.py -n 9 -c 40 -d 0.995 -s 1
```



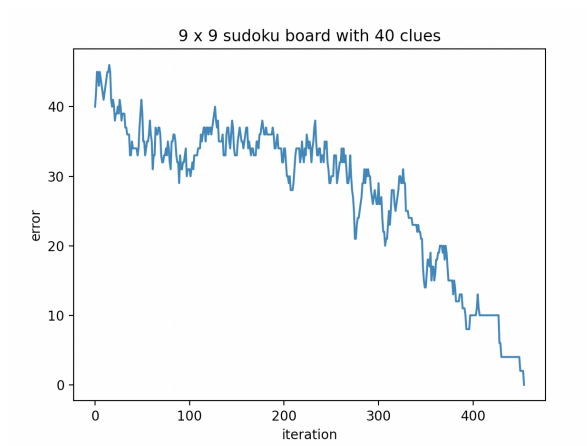
Sudoku puzzle:

```
[[0 7 0 3 5 2 0 9 0]
[0 5 0 9 0 0 7 6 1]
[4 0 9 0 0 0 0 0 2]
[0 0 0 2 6 0 3 4 0]
[5 3 4 0 0 8 6 0 0]
[0 0 2 4 0 0 0 0 0]
[3 4 0 0 0 0 2 0 0]
[6 2 5 8 4 3 1 0 9]
[9 0 7 5 0 0 4 8 0]]
```

```
[[1 7 6 3 5 2 8 9 4]
[2 5 3 9 8 4 7 6 1]
[4 8 9 6 7 1 5 3 2]
[8 9 1 2 6 7 3 4 5]
[5 3 4 1 9 8 6 2 7]
[7 6 2 4 3 5 9 1 8]
[3 4 8 7 1 9 2 5 6]
[6 2 5 8 4 3 1 7 9]
[9 1 7 5 2 6 4 8 3]]
```

Final errors: 0

python sudoku.py -n 9 -c 40 -d 0.995 -s 10



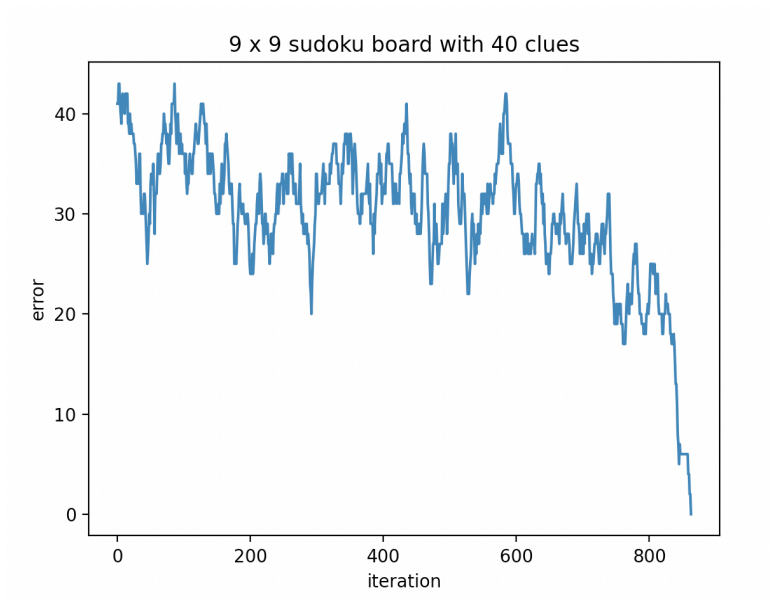
Sudoku puzzle:

```
[[0 3 2 0 7 1 0 6 4]
[4 9 6 0 8 2 0 1 7]
[7 0 0 0 4 0 3 0 8]
[0 4 0 8 6 0 0 0 2]
[2 7 0 0 0 5 8 0 0]
[0 0 0 7 0 3 0 5 0]
[0 0 0 0 3 0 0 4 5]
[5 6 0 0 9 0 0 0 3]
[3 0 7 0 0 4 2 8 0]]
```

```
[[8 3 2 5 7 1 9 6 4]
[4 9 6 3 8 2 5 1 7]
[7 5 1 9 4 6 3 2 8]
[1 4 5 8 6 9 7 3 2]
[2 7 3 4 1 5 8 9 6]
[6 8 9 7 2 3 4 5 1]
[9 2 8 1 3 7 6 4 5]
[5 6 4 2 9 8 1 7 3]
[3 1 7 6 5 4 2 8 9]]
```

Final errors: 0

`python sudoku.py -n 9 -c 40 -d 0.995 -s 100`



Sudoku puzzle:

```
[[0 0 0 0 0 8 1 0]
[0 8 0 3 0 0 7 0 0]
[0 7 0 0 6 8 4 3 0]
[0 1 4 0 0 0 2 6 8]
[6 0 8 5 4 0 3 9 0]
[9 0 0 0 0 2 1 5 4]
[0 5 0 0 2 0 6 0 1]
[4 6 0 7 3 5 0 0 2]
[0 9 2 0 0 0 5 0 3]]
```

```
[[3 4 6 2 9 7 8 1 5]
[1 8 9 3 5 4 7 2 6]
[2 7 5 1 6 8 4 3 9]
[5 1 4 9 7 3 2 6 8]
[6 2 8 5 4 1 3 9 7]
[9 3 7 6 8 2 1 5 4]
[7 5 3 8 2 9 6 4 1]
[4 6 1 7 3 5 9 8 2]
[8 9 2 4 1 6 5 7 3]]
```

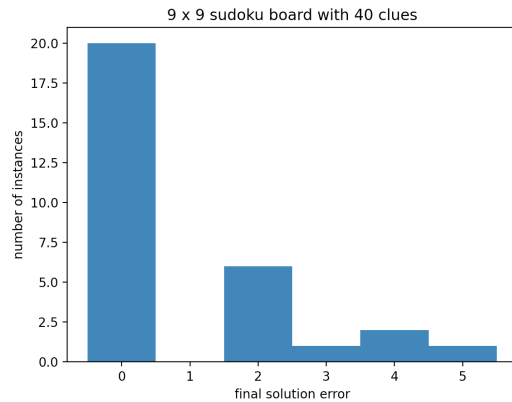
Final errors: 0

Explain how the different values of startT affect the progression of the search.

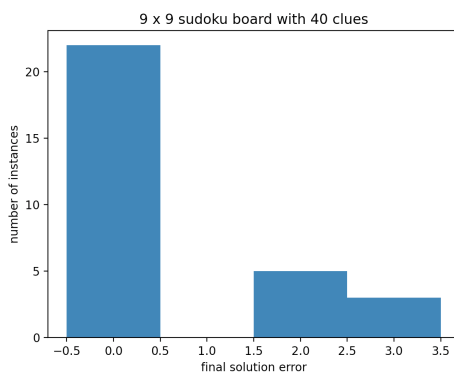
Large startT results in higher fluctuation of error and greater number of iterations. This represents an extensive search of solutions before reaching the final solution.

3. Repeat the above three experiments on 30-batch runs. Show the resultant histograms of each, and again explain how the different values of startT affect (or do not affect) the overall performance of the searches.

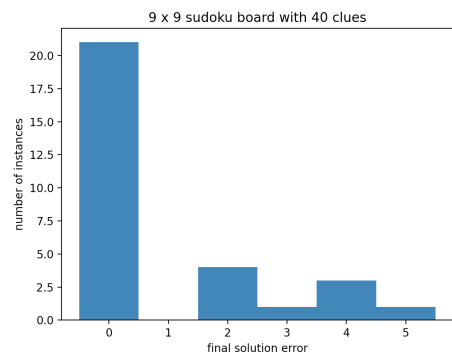
```
python sudoku.py -n 9 -c 40 -d 0.995 -b 30 -s 1
```



```
python sudoku.py -n 9 -c 40 -d 0.995 -b 30 -s 10
```



```
python sudoku.py -n 9 -c 40 -d 0.995 -b 30 -s 100
```



Explain how the different values of startT affect (or do not affect) the overall performance of the searches.

Different values of startT do not seem to affect the overall performance of the searches. This is because the decay value 0.995 plays a more important role than startT. No matter what startT we set, the algorithm will still extensively explore the solution space, therefore compensating any disadvantage in startT.