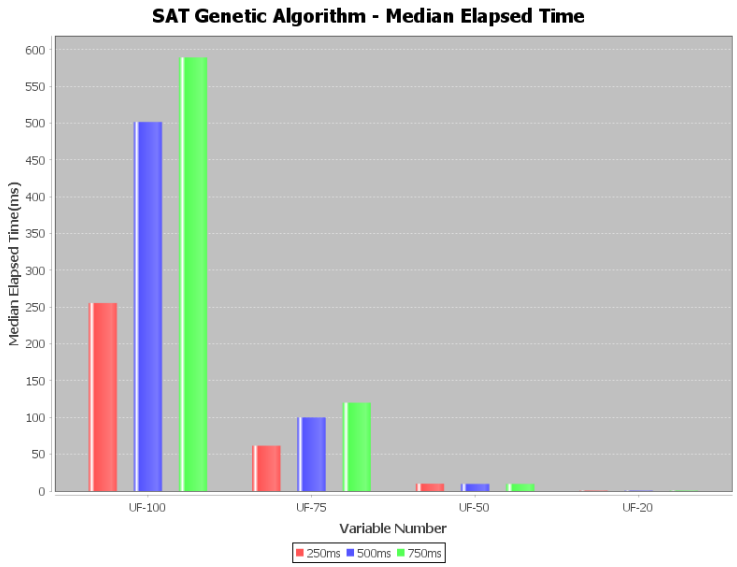
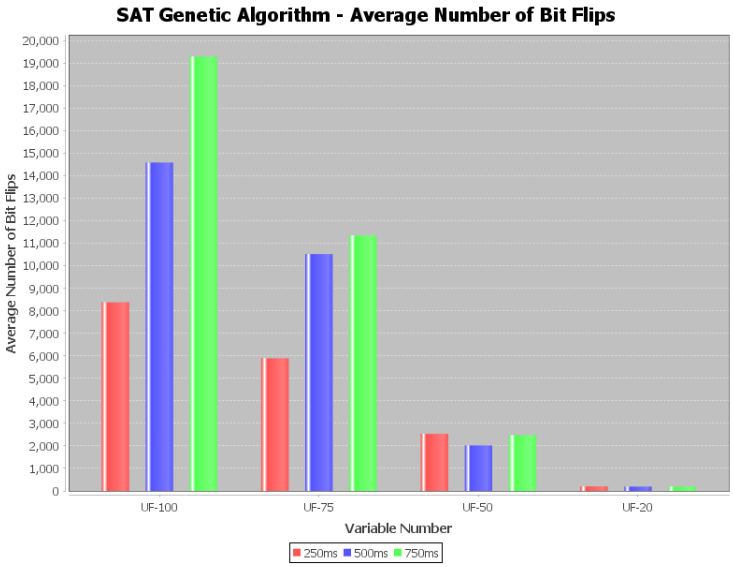
Problem 1

In this problem we were asked to use a genetic algorithm to determine a satisfiable solution for various 3-CNF statements. All of these 3-CNF statements used as benchmarks were satisfiable, our benchmarks only determine if a satisfiable solution can be found with certain time limits. Each statement was contained in its own file. We tested our algorithm on 100 different benchmarks (files) for 20, 50, 75, and 100 variable 3-CNF problems. The algorithm was allowed to run three times for each benchmark each time with a 250ms, 500ms, and 750ms time limit respectively. If an algorithm ran to its alotted time limit it was said to be a failure and was terminated.



Median Elapsed Time: Above is a graph displaying the results of our benchmark evaluations in terms of the median elapsed time. The most interesting result here occured in our UF-100 tests. For the 250ms time limit and 500ms time limit tests the average median time was 250ms and 500ms respectively. Clearly, this means that none were allowed to run to completion and failed to find a satisfiable solution in their time limits. With a 750ms time alottment however, the average runtime was about 580ms implying many, but not all, ran to completion.

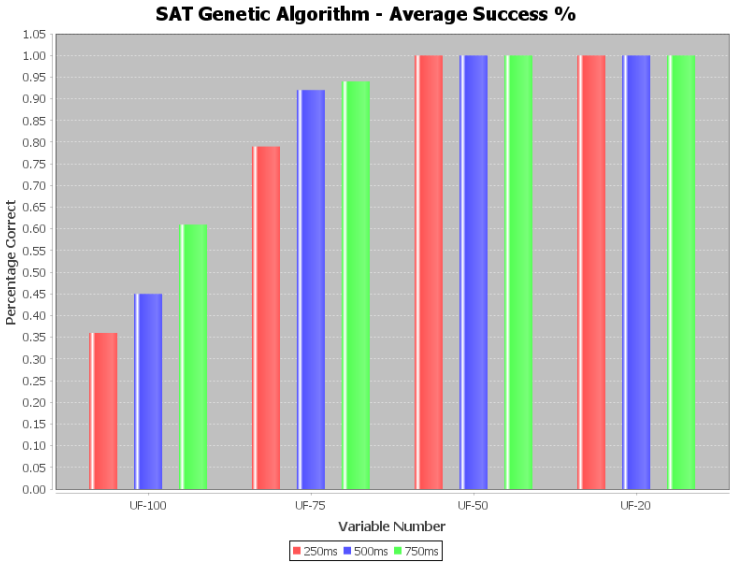
As a trend, the fewer variables a problem had, the less runtime it required. 20 variable problems required median runtimes of less than 5ms on average. 50 variable problems required median runtimes of less than 25ms on average. Based on the results of our benchmark evaluations it seems as though the trend may be that as the number of variables increases the median runtime increases exponentially. This is hard to see in the UF-100 column as the times were explicitly limited to 250 and 500ms and thus solution to completion cannot accurately be described. More benchmark evaluations would be required to prove definitively that this is the relationhip.



Average Number of Bit Flips: Above is the average number of bit flips the algorithm used during the flip heuristic segment of its code for each of our benchmarks. Clearly, the more variables a problem had, the more bitflips it required during execution of the algorithm.

For UF-100 and UF-75, tests with lower time limits yielded on average less bitflips. This is due to the fact that many of these runs did not run until they found a satisfiable state, but were instead cut off by their time limites. This effect is noticable when compared with our UF-50 and UF-20 runs which did not have a correlation between time limit and average bitflips.

In general, as the number of variables in the problem increased the number of bitflips that were required to find a satisfiable solution (or until time ran out) greatly increased.



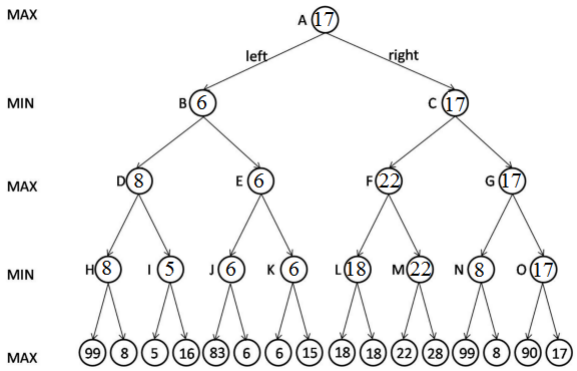
Success Percentage: Above is a chart displaying the percentage across all of our benchmarks that a satisfiable solution for the 3-CNF problem was found.For our UF-20 and UF-50 benchmark evaluations, for all given time limits (250ms, 500ms, and 750ms) every benchmarks that was ran was able to find a satisfiable solution.

As the number of variables in the problem increased however we see that more runs began to reach their time limits. For our UF-75 tests, more than 90% of benchmark evaluations were able to find a satisfiable solution with both a 750ms and 500ms time limit. However, decreasing the time limit to 250ms we began to see our first real drop in percentage successful down to 78%.

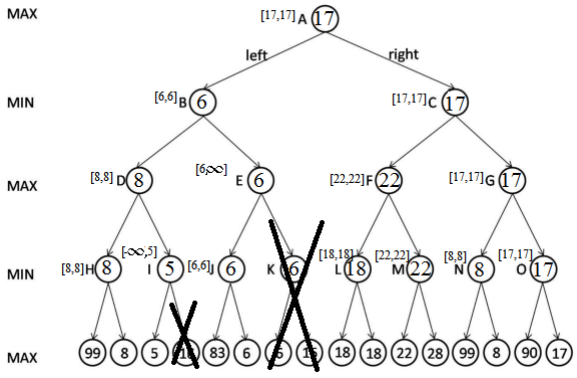
The success rate was worse for our 100 variable benchmarks tests. Naturally, as we reduced the alotted time, less of these runs were able to find a satisfiable solution. Our worst success rate was for UF-100 with an alotted time of 250ms. Here, only 36% of all runs managed to find a satisfiable solution before the time limit. Clearly the results of this section show that as the number of variables in the 3-CNF problem increases, the more time is necessary for the algorithm to find a satisfiable solution.

Problem 2

A.



B.



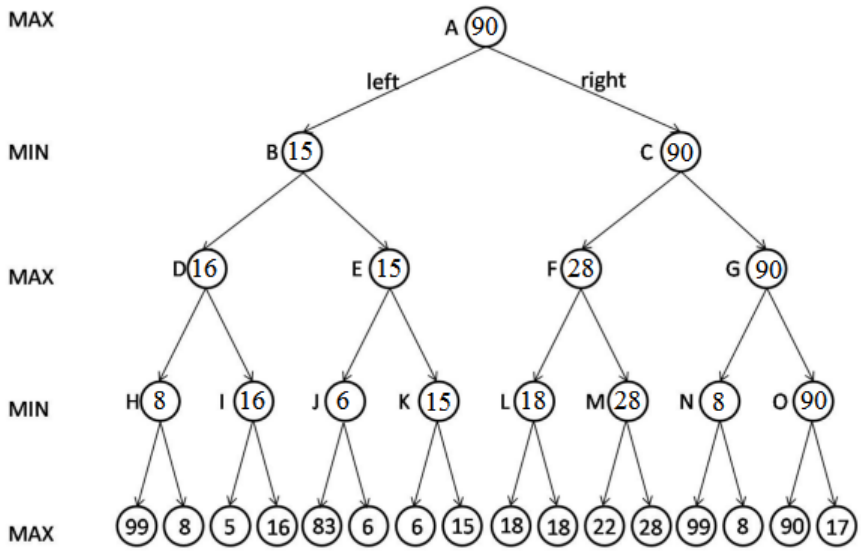
We cut off the subtree at node K since node B is a MIN and will only accept a value less than 8 we know that we wont get a value greater than 6 down said subtree. The same also occurs below node I. The terminal node 15 is pruned because node D will accept any number greater than 8 however, node I is a MIN and the left terminal node being 5 allows us to denote that the right terminal node cannot be less than 5.

C. The max player will choose 17 at the root state by using the exhaustive Minimax Algorithm. The same result will occur when using alpha-beta pruning and the max player will again choose 17. The most optimal move is guaranteed using either the exhaustive Minimax algorithm or the alpha-beta pruning algorithm. This is accomplished by alpha-beta pruning ignoring subtrees that are irrelevant while exhaustive Minimax algorithm checks every node regardless of relevance. The outcome is the same however, since alpha-beta pruning ignores irrelevant subtrees it is more efficient.

D.

E. Considering a similar variation of the game where each minimize evaluation has a 50% chance of selecting either the left action or the right action. We simulated this chance with a simple coin flipped and the end result at the root was 90.

You cannot apply alpha-beta pruning in this example because the process is dependent on being able to see which action a player will choose. With the element of randomness added to the minimization functions, it becomes impossible for the algorithm to successfully foresee which action a player will choose thus making it unable to prune any subtrees or nodes.



Problem 3

A. **Variables:** Defined as $n_{i,j}$ where $i \in$ {set of all row numbers} and $j \in$ {set of all column numbers} for each cell n on the board. Essentially, each square on the sudoku board will be a variable, for a total of 81 variables.

Domain: Defined as the set of possible values in sudoku. Here, this set is defined as $\{1, 2, ..., 9\}$

Constraints:

1. All variables in any row must be assigned distinct values.
2. All variables in any column must be assigned distinct values.
3. All variables in any 3x3 region must be assigned distinct values.
4. The assignments given to variables in the start state cannot be changed.

B. **Start State:** The start state is a given board that has variables assigned with certain values. These values are the basis for the constraints for the rest of the problem and may not be changed. Below is an example start state.

	1		4	2				5
		2		7	1		3	9
							4	
2		7	1					6
				4				
6					7	4		3
	7							
1	2		7	3		5		
3				8	2		7	

Successor Function:

1. Select a random unassigned variable
2. Assign a value to this variable that does not violate any constraints
3. The resulting state is the successor

Goal Test: A state is considered to a valid goal state if it is both consistent and complete.

Path Cost Function: The path cost from any one state to its successor state will be 1.

C. Sudoku problems that are easy compared to ones that are hard can be determined based on the relative usefulness of applying the minimum remaining values heuristic to their search states.

Consider a Sudoku board for which there are an equal number of remaining values for each variable based on their constraints. Here, the heuristic will not be of much use to us since it will not help us narrow our search and we will have to backtrack from many more states to find a satisfiable goal state than if we were faced with an easier Sudoku board.

An easier Sudoku board can be defined as one for which there is a varied range of minimum remaining values for each variable. This will be an easier game because the minimum remaining values heuristic will direct us to expand states with the least minimum remaining values first. Since the distribution of minimum remaining values for an easy board is more varied, there will always be a good estimate of what state to expand next and this will lead us more directly to the goal state.

D. Pseudocode for Local Search

Algorithm 1 Sudoku Local Search Algorithm

```
1: procedure SudokuSolver(Board board)
2:     board ← randomizeCompleteState(board)
3:     while !completeAndSatisfied(board) do
4:         nextBoard ← updateState(board)
5:         if evaluateSatisifiability(nextBoard) > evaluateSatisifiability(board) then
6:             board ← nextBoard
7:     return board
```

Due to its random nature this problem will perform worse on easier problems and better on hard problems than its incremental formulation counterpart.

When local search using incremental formulation is used with a good heuristic it will be able to solve easier problems much faster because the heuristic will be very good at directing the search to the goal state. Compared to complete state formulation, which randomizes one variable in a complete state as a successor state, the incremental method will be much more directed and be able to find a solution much faster. Whereas because of the randomness of complete state it will most likely take longer to find a solution.

With harder Sudoku problems however, the incremental formulation will not find a solution as fast as the complete state formulation. The heuristic that the incremental formulation uses will not be very useful for hard problems because it will still have to search a lot of the search tree to make up for what cannot be informed by the heuristic. The complete state formulation however, since it changes states randomly, will not have this problem, and will probabilistically complete the algorithm in faster time than the incremental formulation.

Problem 4

For Superman to be defeated, it has to be that he is facing an opponent alone and his opponent is carrying Kryptonite. Acquiring Kryptonite, however, means that Batman has to coordinate with

Lex Luthor and acquire it from him. If, however, Batman coordinates with Lex Luthor, this upsets Wonder Woman, who will intervene and fight on the side of Superman.

S = Superman Defeated
A = Facing Opponent Alone
K = Opponent Carrying Kryptonite
B = Batman Coordinates with Lex Luthor
W = Wonder Woman Upset

A. $S \implies A \wedge K$
$K \implies B$
$B \implies W$
$W \implies \neg A$

B. $S \implies A \wedge K \equiv \neg S \vee (A \wedge K) \equiv (\neg S \wedge A) \vee (\neg S \wedge K)$
$K \implies B \equiv \neg K \vee B$
$B \implies W \equiv \neg B \vee W$
$W \implies \neg A \equiv \neg W \vee \neg A$

$$KB = (\neg S \vee K) \wedge (\neg S \vee A) \wedge (\neg K \vee B) \wedge (\neg B \vee W) \wedge (\neg W \vee \neg A)$$
$$\alpha = \neg S$$

$$(\neg S \vee K) \wedge (\neg S \vee A) \wedge (\neg K \vee B) \wedge (\neg B \vee W) \wedge (\neg W \vee \neg A)$$
$$(\neg S \vee \neg S) \wedge (\neg A \vee A) \wedge (\neg K \vee K) \wedge (\neg B \vee B) \wedge (\neg W \vee W)$$
$$\therefore \text{KB} \mid = \neg S$$

Superman being defeated implies that not only did his opponent have Kryptonite but that Superman was also fighting alone. This is equivalent to Superman not being defeated or his opponent has Kryponite and Superman is fighting alone. This all equates to:

$$(\neg S \vee K) \wedge (\neg S \vee A)$$

Superman fighting alone implies that Superman was defeated.

$$(\neg K \vee B)$$

Batman coordinating with Lex Luthor implies that Wonder Woman will be upset and will team up with Superman.

$$(\neg B \vee W)$$

Wonder Woman being upset implies that she will team up with Superman and he will not be fighting alone.

$$(\neg W \vee \neg A)$$

Based off the knowledge presented above we can prove that Batman cannot defeat Superman.

Problem 5

C. A.

Problem 6

- A. Using the minimum value of $h_1(n)$ and $h_2(n)$ for each state:

  $$\text{Given that } h_1(n) \leq h^*(n) \text{ is always true,}$$
  $$Min(h_1(n), \infty) \leq h^*(n)$$
  $$\text{Therefore } h_1(n) = minimum(h_1(n), h_2(n)) \text{ is admissible.}$$
  $$\text{consistent?}$$

- B. Using the maximum value of $h_1(n)$ and $h_2(n)$ for each state:

  $$\text{Given that } h_1(n) \leq h^*(n)$$
  $$h_2(n) \leq h^*(n)$$
  $$Max(h_1(n), h_2(n)) \leq h^*(n)$$
  $$\text{Therefore } h_4(n) = maximum(h_1(n), h_2(n)) \text{ is admissible.}$$
  $$\text{consistent?}$$

- C. The defined heuristic function

  $$h_3(n) = w \times h_1(n) + (1 - w)h_2(n), \text{ where } 0 \leq w \leq 1$$

  is admissible only when w is a value less than 0.5, any value larger than 0.5 will cause $h_1(n)$ to be multiplied by a value larger than $h_2(n)$. Since the the bounds are from 0 to 1 the heuristic is NOT admissible.

  consistent?

- D. Considering the informed, best-first search algorithm with an object function of $f(n) = (2 - w) \times g(n) + w \times h(n)$. It is guaranteed to work for $0 \leq w \leq 1$ since being multiplied by g(n) which is a constant implies no effect on which order chosen paths are arranged. However, if $w > 1$, then the goal state may have an overestimated distance which will make the heuristic not admissible and so not optimal.

  Given the information above we can assess the function for when the values of w are 0, 1, and 2.

  $w = 0$:

  When $w = 0$ it will make $f(n) = 2g(n)$ and causes the algorithm to perform like a Uniform Cost Search since there is no weight assigned it will find the most optimal path however, it will not be efficient.

  $w = 1$:

  When $w = 1$ it will make $f(n) = g(n) + h(n)$ which causes the algorithm to perform like A^* and as with $w = 0$ is guaranteed to find the optimal path however, is more efficient.

  $w = 2$:

  When $w = 2$ it will make $f(n) = 2h(n)$ which causes the algorithm to perform like Greedy Best first search.