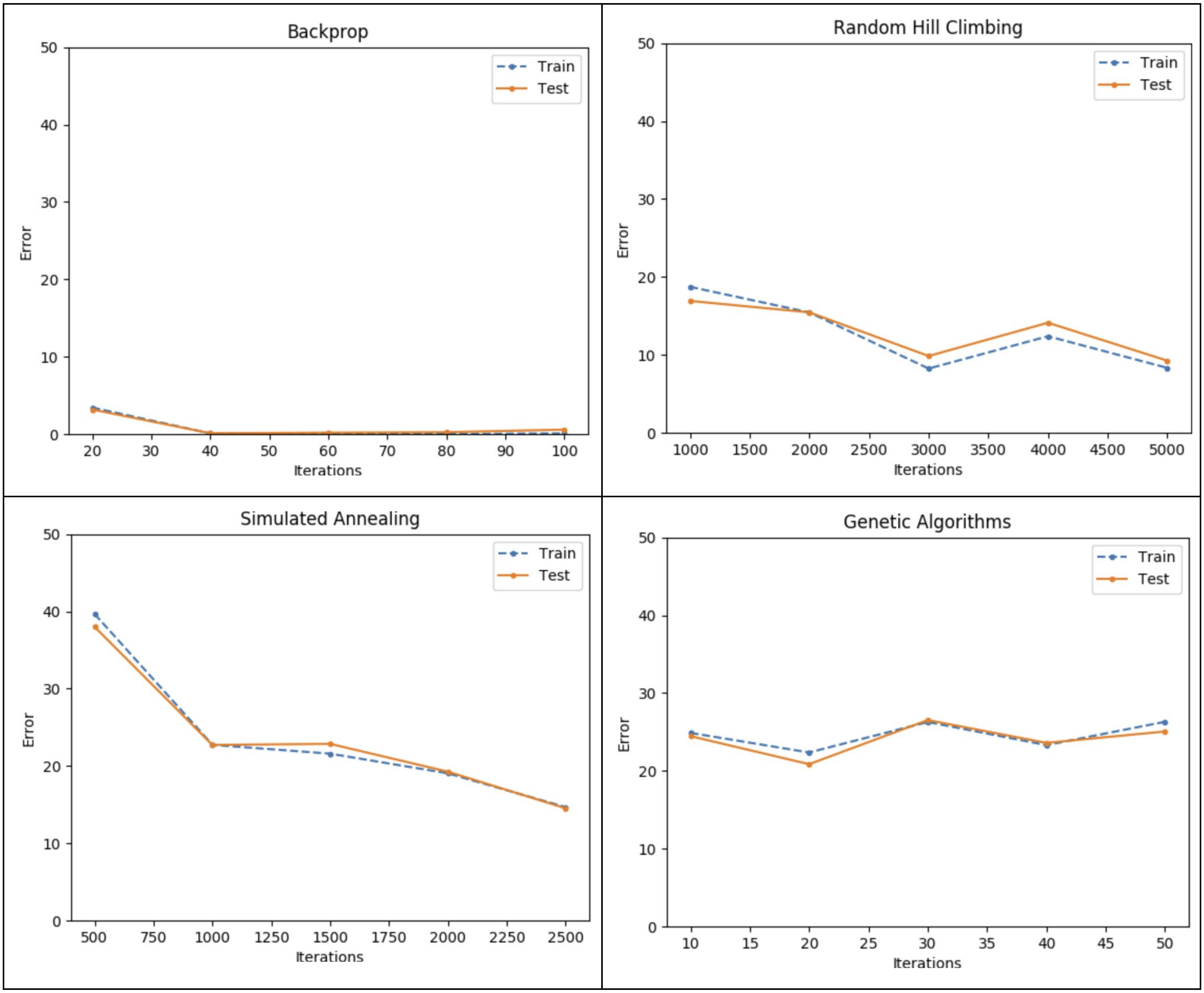
NEURAL NETWORK

After using various optimization algorithms to train a neural network, backpropagation clearly is the best method for updating the network weights. Not only does backpropagation train much faster, but the validation error was almost zero, as opposed to the next-best validation error of about 15 percent.

Performance



	Iterations	**Time (Seconds)**
Backprop	100	3.425
Random Hill Climbing	5000	54.730
Simulated Annealing	2500	30.483
Genetic Algorithms	50	91.031

Parameter Optimization

Neual Network		Simulated Annealing			Genetic Algorithm			
Nodes	Error	Initial T	Cooling	Error	Population	Mates	Mutations	Error
5	0.2	100000	0.9	52.378	315	63	63	22.748
10	0.286	100000	0.95	45.634	315	63	126	22.93
17	0.371	100000	0.99	48.532	315	126	63	25.173
20	0.343	100000	0.999	46.647	315	126	126	25.395
50	0.344	100000000	0.9	53.888	472	63	63	22.013
100	0.458	100000000	0.95	48.795	472	63	126	20.982
200	0.4	100000000	0.99	50.034	472	126	63	22.584
		100000000	0.999	53.54	472	126	126	21.991
		10000000000	0.9	49.387	630	63	63	24.446
		10000000000	0.95	54.292	630	63	126	24.475
		10000000000	0.99	47.641	630	126	63	23.082
		10000000000	0.999	50.212	630	126	126	23.179
		1000000000000	0.9	48.582	787	63	63	21.372
		1000000000000	0.95	50.739	787	63	126	25.135
		1000000000000	0.99	47.442	787	126	63	23.672
		1000000000000	0.999	51.138	787	126	126	21.157
		1000000000000000	0.9	48.494				
		1000000000000000	0.95	51.408				
		1000000000000000	0.99	47.92				
		1000000000000000	0.999	53.058				

Analysis

Optimization algorithms that climbs towards optima don’t seem to perform any better than backpropagation, which is presumably because backpropagation itself uses gradient descent.

The best hyperparameters for each randomized optimization algorithm were obtained by training a neural net using many combinations of hyperparameters over a small number of iterations. Once the best hyperparameters were chosen based on the performance of each model, the best set of hyperparameters to train a neural network over an increasing number of iterations. Slightly better results could potentially be achieved by testing different hyperparameters combinations over a larger number of iterations, but doing so would likely not be worth the increased duration of execution required to train over more iterations.

Randomized optimization could be most useful if used to find the neural net hyperparameters instead of simply trying all combinations of sequential values. Doing so would definitely take more time, but would likely cover a much larger domain of potential parameters. The validation error could simply be used as a cost function.

OPTIMIZATION PROBLEMS

Many problems were tried, but the three represented in this report are “Flip Flop”, “Knapsack”, and “Cosine of Sine of Ones”. All problems are maximization problems. For each problem, randomized hill climbing, simulated annealing, a genetic algorithm, and MIMIC were used to find the set of optima of the fitness function.

Flip Flop
The flip flop fitness function takes as input a bitstring returns the number of alternating bits. “010101” and “101010” are examples of ideal bit strings because every bit alternates, resulting in a fitness score of 5 alternations.

The flip flop problem was intended to show off MIMIC but ended up showing off simulated annealing, presumably because of the ability of the hill climbing algorithm to track how individual bit changes affect the fitness function and follow and accordingly ascend to a summit. Interestingly, randomized hill climbing performed far worse than simulated annealing, so there are presumably many local optima that may lead a hill climber down the wrong path.

Knapsack
The knapsack problem is, given a set of items each with weight and value and a knapsack with a weight limit, find the highest total value that can be held by the knapsack. The fitness function takes as input a set of items and returns the total value of the items, provided the weight limit is not exceeded. If the weight limit is exceeded, the fitness function returns the excess weight multiplied by a number slightly greater than 0.

The knapsack problem is interesting because finding an optimum may require trying many different combinations of items, and there isn’t a clear path to the optima like a hill climbing algorithm might find for a bitstring problem.

Cosine of Sine of Ones
The cosine of sine of ones fitness function is a custom fitness function that is a simple modification of the “Count Ones” fitness function. The fitness function takes as input a bitstring and returns the cosine of the sine of the number of 1’s in the bitstring. The curve of the fitness function looks like a sine or cosine curve but with only positive values.

The goal of this function was to show that genetic algorithms can do well with a strange evaluation function that has many different optima.

Summary
Simulated annealing performed incredibly well for any problems that had any kind of continuous ascent towards optima, which is generally true for bitstring problems where chang one bit at a time causes the fitness function to go up towards an optimum or down away from one. Simulated annealing executed quickly, even with many iterations.

MIMIC also performed very well on a number of problems, particularly problems that were too complex for simulated annealing to get just right. Since MIMIC retains a distribution for the entire sample space, it is not easily confused by problems that have misleading local optima or unexpected global optima. MIMIC generally executed slower in terms of time, but used fewer iterations than other algorithms.

Genetic algorithms did not perform particularly well nor badly. The adage, “a genetic algorithm is the second best for any problem” makes more sense now, as the consensus on genetic algorithms is that they provide a decent heuristic for pretty much any problem, but are unlikely to converge on an absolute optimum.[1]

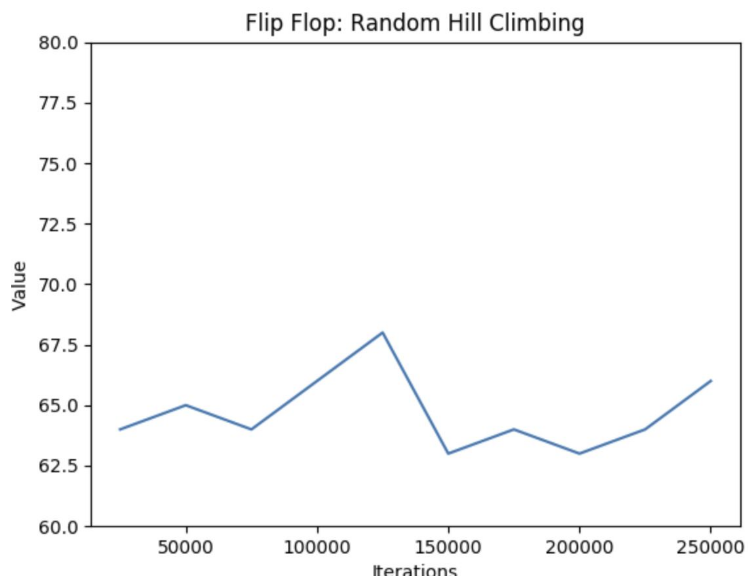
Randomized hill climbing, similar to genetic algorithms, provided a decent baseline for many problems but rarely (or never) performed best.

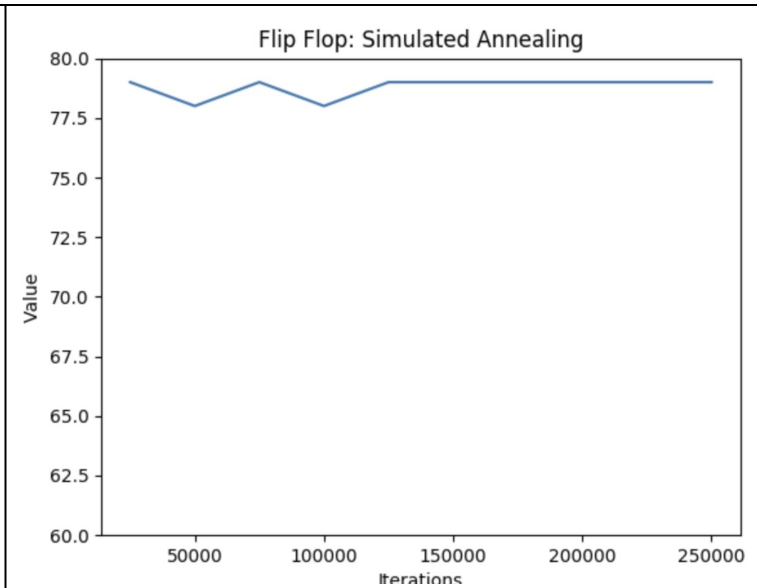
---

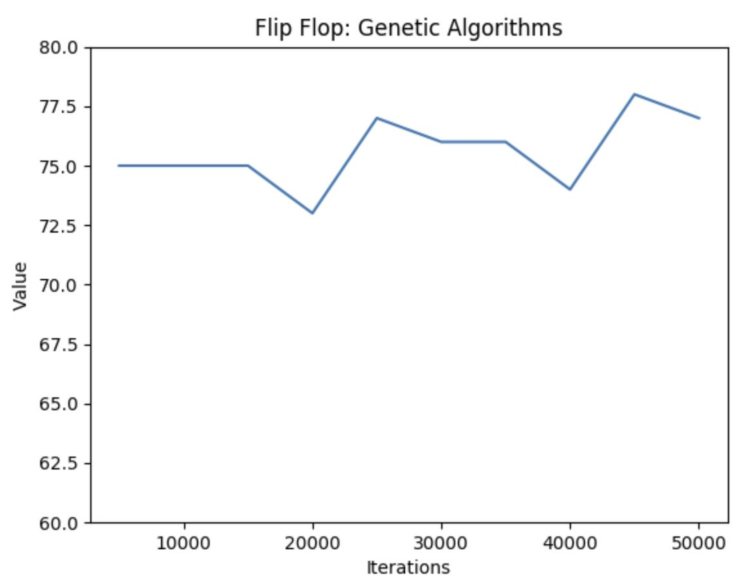
[1] https://stats.stackexchange.com/questions/249471/when-are-genetic-algorithms-a-good-choice-for-optimization

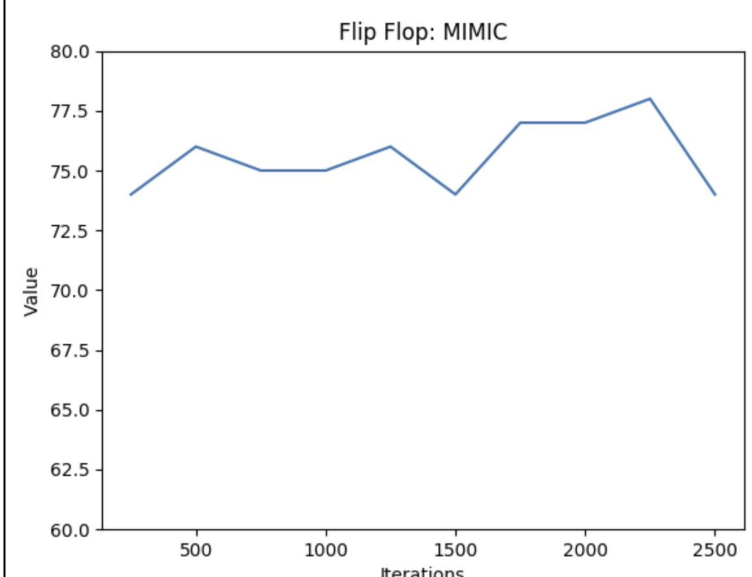
Flip Flop

Performance









	Iterations	**Time (Milliseconds)**
Randomized Hill Climbing	250000	35
Simulated Annealing	250000	146
Genetic Algorithms	50000	3959
MIMIC	2500	5705

Parameter Optimization

Simulated Annealing			Genetic Algorithm				MIMIC		
Initial T	Cooling	Value	Population	Mates	Mutations	Value	Samples	To Keep	Value
100000	0.9	79	150	50	10	68	100	2	72
100000	0.95	79	150	50	20	69	100	4	73
100000	0.99	79	150	50	30	73	100	8	69
100000	0.999	78	150	100	10	71	100	16	72
100000000	0.9	79	150	100	20	68	150	2	74
100000000	0.95	79	150	100	30	67	150	4	67
100000000	0.99	79	150	150	10	68	150	8	69
100000000	0.999	79	150	150	20	73	150	16	72
10000000000	0.9	79	150	150	30	71	200	2	72
10000000000	0.95	79	200	50	10	70	200	4	71
10000000000	0.99	79	200	50	20	72	200	8	72
10000000000	0.999	79	200	50	30	70	200	16	64
1000000000000	0.9	79	200	100	10	72	250	2	73
1000000000000	0.95	79	200	100	20	73	250	4	75
1000000000000	0.99	79	200	100	30	73	250	8	73
1000000000000	0.999	79	200	150	10	70	250	16	69
1000000000000000	0.9	79	200	150	20	69	300	2	73
1000000000000000	0.95	79	200	150	30	74	300	4	77
1000000000000000	0.99	78	250	50	10	68	300	8	68
1000000000000000	0.999	78	250	50	20	72	300	16	65
			250	50	30	72			
			250	100	10	70			
			250	100	20	73			
			250	100	30	72			
			250	150	10	69			
			250	150	20	75			
			250	150	30	73			

Analysis

I thought MIMC would perform best for this problem because of the inherent structure that the optima possess, but simulated annealing is the clear winner.

Simulated annealing performed very well for bitstring problems that had continuous fitness functions, such as this one. In fact, this is one of the few bistring problems I tried for which simulated annealing did not find the absolute maximum for any parameter combination or number of iterations.

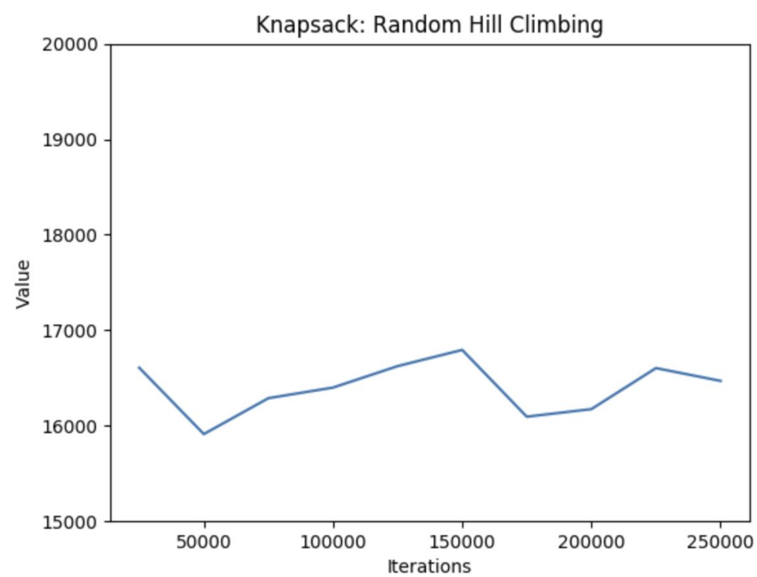
Genetic algorithms performed rather well relative to its usual performance, most likely due to the fact that the optima are string patterns rather than maximum values of a numerical function, which is analogous to finding beneficial mutations.

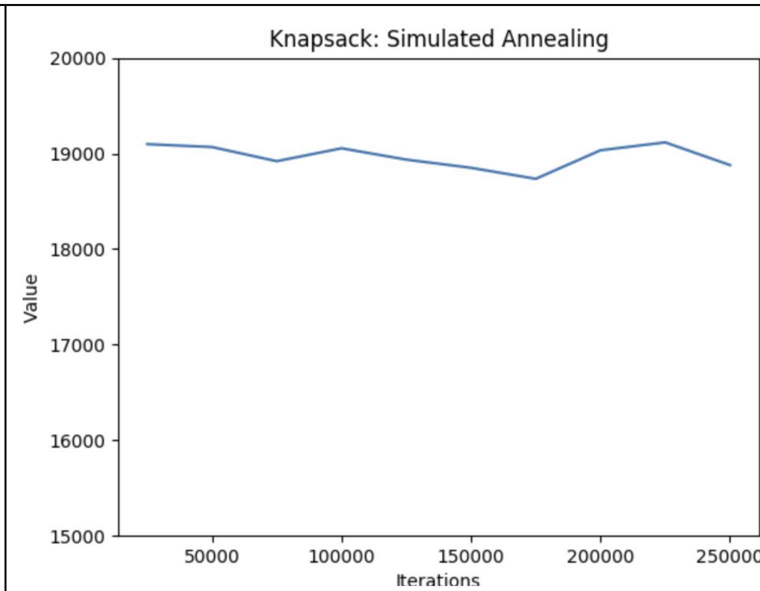
The randomness for high temperatures in simulated annealing is clearly very important, as straightforward hill climbing performed the worst by far. Hill climbing has no way to determine if “10000” -> “10010” or “10000” -> “10100” is a better change, which could presumably lead it to many false (local but not global) optima.

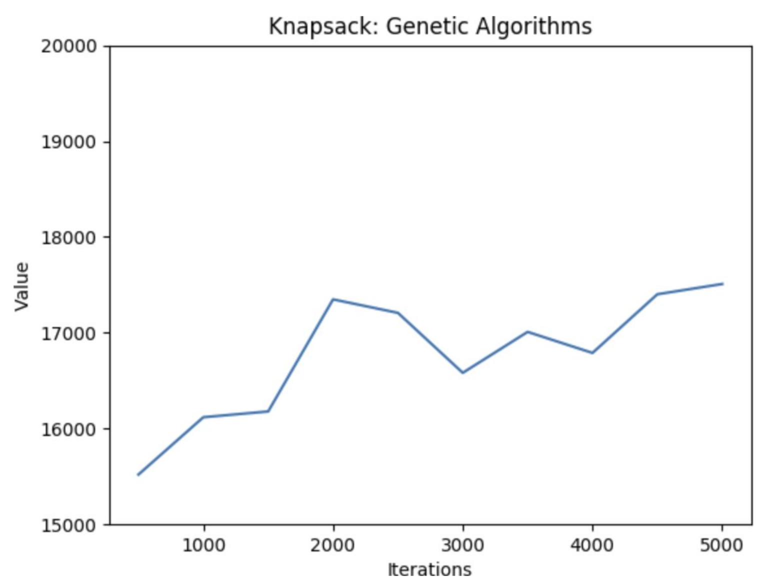
MIMIC may not have performed the best due to the unlikelihood of the exact global maximum being sampled.

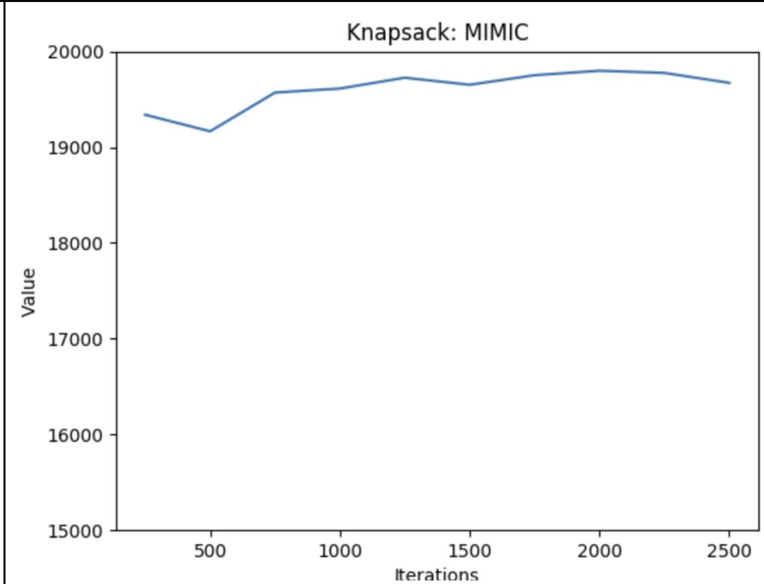
Knapsack

Performance









	Iterations	**Time (Milliseconds)**
Randomized Hill Climbing	250000	138
Simulated Annealing	250000	154
Genetic Algorithm	5000	1532
MIMIC	2500	33532

Parameter Optimization

Simulated Annealing			Genetic Algorithm				MIMIC		
Initial T	Cooling	Value	Population	Mates	Mutations	Value	Samples	To Keep	Value
100000	0.9	15765	150	50	10	14897	100	2	15346
100000	0.95	16871	150	50	20	14805	100	4	18749
100000	0.99	17199	150	50	30	14329	100	8	19039
100000	0.999	18945	150	100	10	15207	100	16	18760
100000000	0.9	16749	150	100	20	15338	150	2	15726
100000000	0.95	16653	150	100	30	14691	150	4	18861
100000000	0.99	16956	150	150	10	14894	150	8	19118
100000000	0.999	19146	150	150	20	15132	150	16	19169
10000000000	0.9	16517	150	150	30	15670	200	2	16812
10000000000	0.95	16828	200	50	10	15750	200	4	19253
10000000000	0.99	17059	200	50	20	15779	200	8	19487
10000000000	0.999	19024	200	50	30	15286	200	16	19240
1000000000000	0.9	16184	200	100	10	14747	250	2	16292
1000000000000	0.95	16588	200	100	20	15423	250	4	19405
1000000000000	0.99	16577	200	100	30	16077	250	8	19418
1000000000000	0.999	18775	200	150	10	16044	250	16	18947
1000000000000000	0.9	16779	200	150	20	15981	300	2	16997
1000000000000000	0.95	17078	200	150	30	15668	300	4	19378
1000000000000000	0.99	16374	250	50	10	15858	300	8	19370
1000000000000000	0.999	19016	250	50	20	16206	300	16	19500
			250	50	30	16141			
			250	100	10	16599			
			250	100	20	15643			
			250	100	30	17011			
			250	150	10	16281			
			250	150	20	16257			
			250	150	30	16485			

Analysis

MIMIC clearly outperforms all other algorithms significantly for this problem.

Finding an optimal combination of items is a very tricky problem because adding an item that increases value can be extremely misleading. Optimizing this problem is very hard without the ability to consider all combinations of items - or the whole input space - simultaneously, which MIMIC exclusively can do.

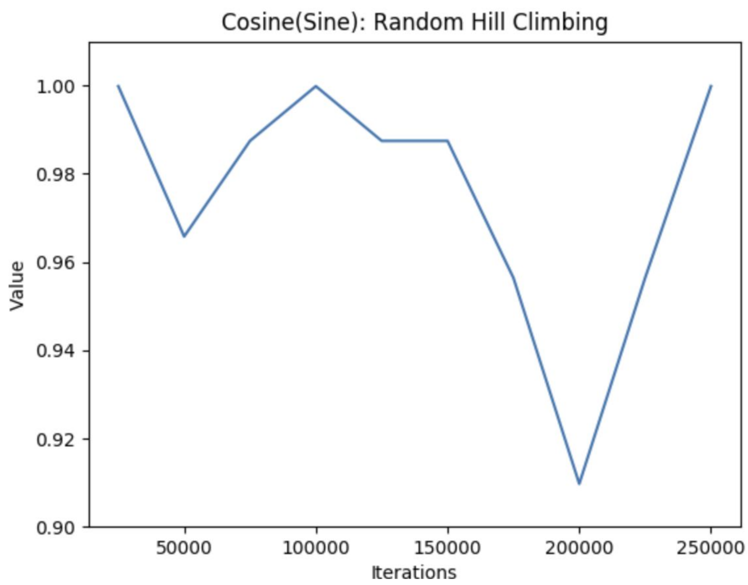
The genetic algorithm steadily performs better with more iterations, which makes sense because iterations allow it to try more combinations of items. It also performs reasonably well because, at the end of the day, it is a relatively unguided approach to trying a huge number of combinations and selecting the best one, which is not an ideal approach but also not a terrible one.

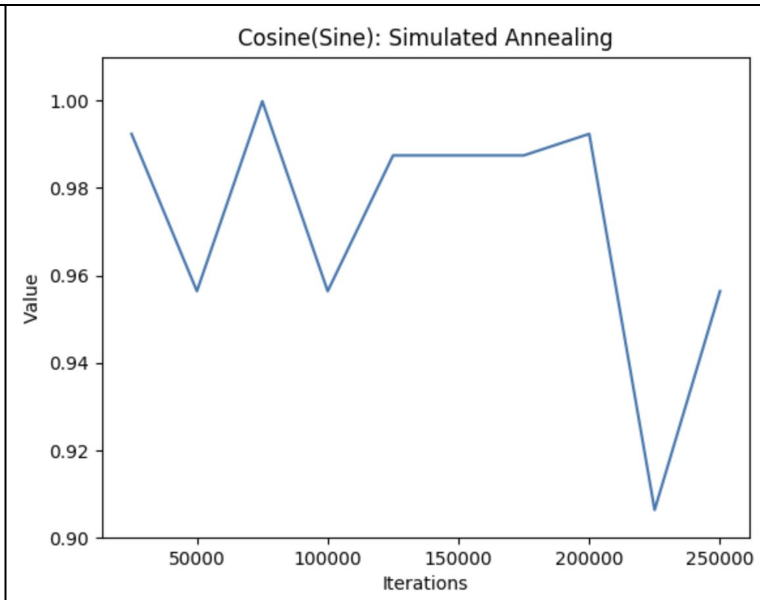
Randomized hill climbing presumably gets stuck by local maximums where the algorithm commits to including a particularly valuable item that would not be included in the most valuable knapsack.

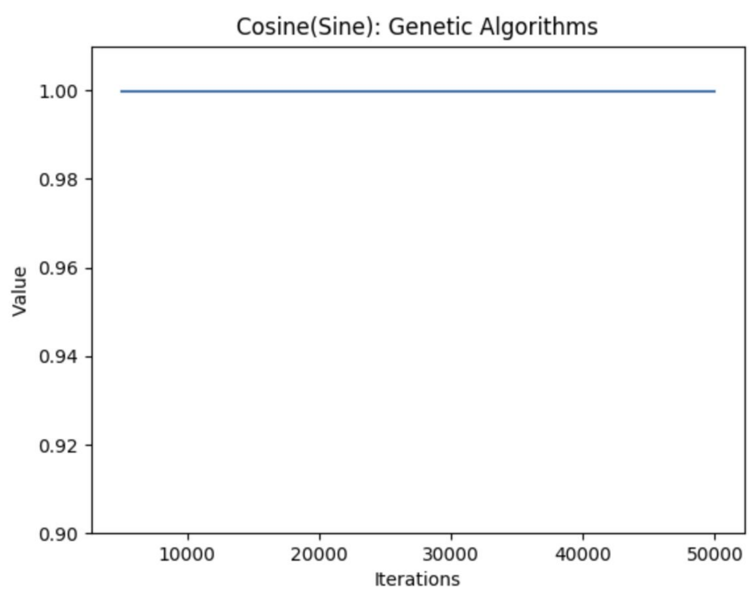
Simulated annealing performs surprisingly and consistently well. If the initial phase of simulated annealing allows the algorithm to traverse the base of many peaks and stabilize on the largest peak, it makes intuitive sense that the algorithm might settle on local optima that are close to the global optima. However, MIMIC still performs consistently much better than simulated annealing, which means that simulated annealing is not finding global optima.

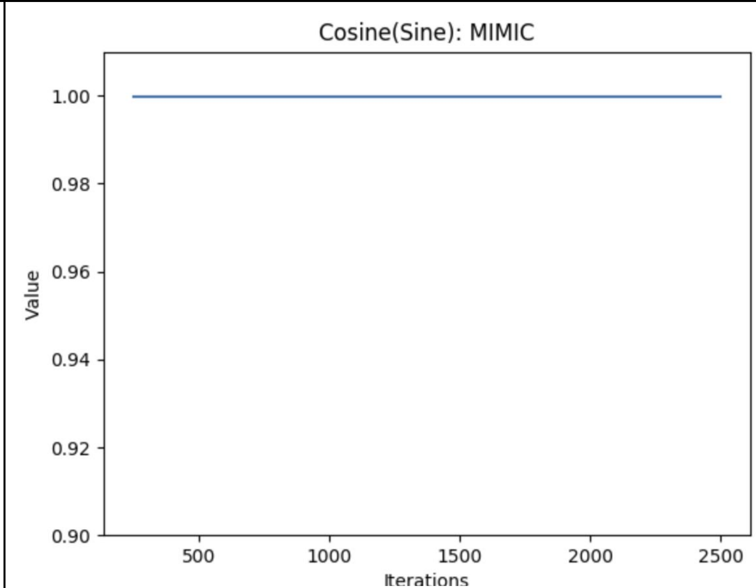
Cosine of Sine of Ones

Performance









	Iterations	**Time (Milliseconds)**
Randomized Hill Climbing	250000	54
SImulated Annealing	250000	154
Genetic Algorithm	50000	4252
MIMIC	2500	3474

Parameter Optimization

Simulated Annealing		
Initial T	Cooling	Value
100000	0.9	0.98745
100000	0.95	0.99984
100000	0.99	0.90972
100000	0.999	0.99984
100000000	0.9	0.99984
100000000	0.95	0.99237
100000000	0.99	0.95640
100000000	0.999	0.98745
10000000000	0.9	0.99237
10000000000	0.95	0.98745
10000000000	0.99	0.99237
10000000000	0.999	0.99237
1000000000000	0.9	0.99984
1000000000000	0.95	0.99984
1000000000000	0.99	0.95640
1000000000000	0.999	0.99237
1000000000000000	0.9	0.90972
1000000000000000	0.95	0.95640
1000000000000000	0.99	0.90972
1000000000000000	0.999	0.95640

Genetic Algorithm			
Population	Mates	Mutations	Value
150	50	10	0.98745
150	50	20	0.99984
150	50	30	0.99984
150	100	10	0.99984
150	100	20	0.99984
150	100	30	0.99984
150	150	10	0.99984
150	150	20	0.99984
150	150	30	0.99984
200	50	10	0.99984
200	50	20	0.99984
200	50	30	0.99984
200	100	10	0.98745
200	100	20	0.99984
200	100	30	0.99984
200	150	10	0.99984
200	150	20	0.99984
200	150	30	0.99984
250	50	10	0.99984
250	50	20	0.99984
250	50	30	0.99984
250	100	10	0.98745
250	100	20	0.99984
250	100	30	0.99984
250	150	10	0.99984
250	150	20	0.99984
250	150	30	0.99984

MIMIC		
Samples	To Keep	Value
100	2	0.99984
100	4	0.99984
100	8	0.99984
100	16	0.99984
150	2	0.99984
150	4	0.99984
150	8	0.99984
150	16	0.99984
200	2	0.99984
200	4	0.99984
200	8	0.99984
200	16	0.99984
250	2	0.99984
250	4	0.99984
250	8	0.99984
250	16	0.99984
300	2	0.99984
300	4	0.99984
300	8	0.99984
300	16	0.99984

Analysis

The goal with this problem was to find a problem that genetic algorithms performs very well on. Since this is a simple problem with many (infinite) global maxima, genetic algorithms consistently finds one of them.

The problem is also trivial for MIMIC apparently, which makes sense because the function would be very well represented by sampling.

Interestingly, the hill climbing algorithms get confused by the seemingly straightforward peaks and valleys of the sine-shaped wave. My guess there is that, where a simple “Count Ones” fitness function would have a trivial value ascent, optimizing the value of the sine of an increasing input might not be so obvious. Additionally, similar fitness values do not necessarily have similar bitstrings at all.