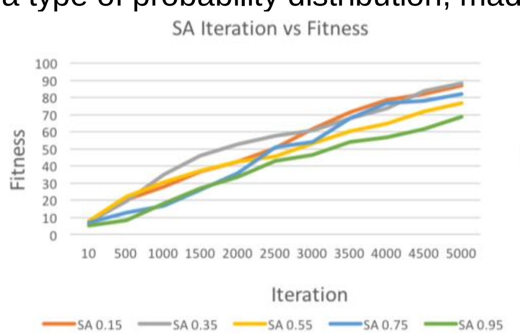
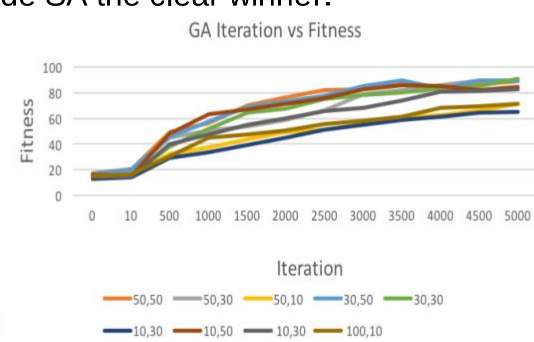
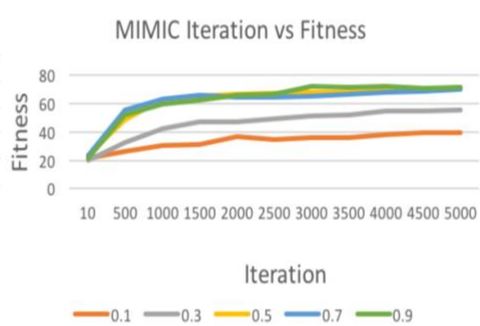
Continuous Peaks:
This problem is related to the four peaks problem. This problem tries to achieve consecutive 0s at the beginning, and consecutive ones at the end of the string, and these sums account for the fitness function. But, if the number of 0s and the number of 1s are above some threshold, then there is a bonus added to the fitness score. So two peaks are when we have lots of 0s, or lots of 1s that result up to a threshold, and these are the smaller peaks, while the two other, and bigger peaks, are the bonus where both 0s and 1s have to pass the threshold. I used a bit string of size 100, and my threshold, T, is 49, and the problem ran for 5000 iterations for every algorithm. I used the same hyperparameters as before for SA, but I introduced some more combinations of hyper parameters for GA, since bit string type problems are often by GA, so I wanted to explore it more. For MIMIC, the I always took 100 samples from the uniform, and kept 50 of those best fit samples, and tuned the Bayesian estimate parameter, with values: 0.1, 0.3, 0.5, 0.7, 0.9. This parameter is used when constructing the dependence tree. It tells me the likelihood of a node, by fitting a correlation with this parameter and the sample's probability/distribution. All algorithms were run for 5000 iterations.

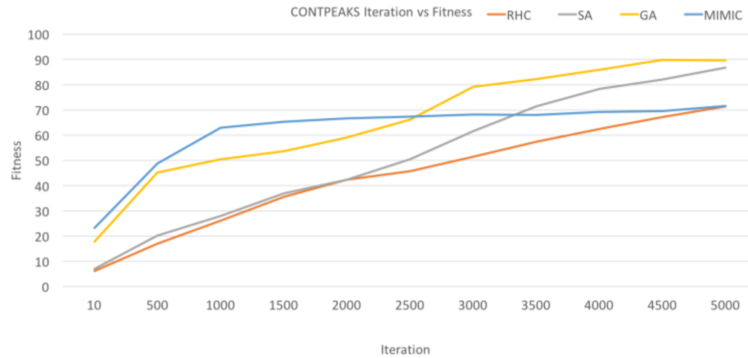
For SA, the best cooling factor was 0.35. You can see that .15 was not far behind, but increasing the rate more definitely decreased the performance, meaning more exploiting was needed in this case to find global optima's basin of attraction. For GA, 30 mate and 30 mutate did the best. These values show that mating 50 at a time took in too much randomness, and mating 10 took just did not get enough of the best fit, even after mutating. The sweet spot was picking 30 and mutating 30. As for MIMIC, both 0.5 and 0.9 for epsilon performed the best. Low likelihood for picking a node of exploration doesn't help us choose the global optima, so in this case, we had to set it a little higher.

You can see that MIMIC and SA did the best for fitness, but for this problem, I am choosing SA as the best optimization problem. GA and RHC perform the worst, although GA performed slightly better because it converge to the best fitness around 1000 iterations, while it took RHC 5000 iterations to do random restarts to be able to find the same optima. Performance wise, GA was better than RHC, but it still could not reach as high a score as SA and MIMIC, since it got stuck at a local optima, and takes a while to find a new string to escape, since it does not understand structure of the problem. RHC and SA both start of similar, but around 2000 iterations, SA starts increasing in fitness at a higher score. This is because it had a high enough T and small enough cooling factor to explore the whole space and not get stuck when it saw the local optima. Now we compare SA and MIMIC and see why SA is better even though they eventually result in the same fitness score, using Iteration vs Time. MIMIC by far takes more time per iteration, while SA is linear. As we said before, this is because we redo the distribution each time before exploring. In this case, SA performs just the same, with the same number of iterations, so we don't need the time complexity of MIMIC. Even though we would think MIMIC would perform the best by understanding the structure and getting information about all four maxima to pick the best, the time complexity, and the fact that SA does use a type of probability distribution, made SA the clear winner.



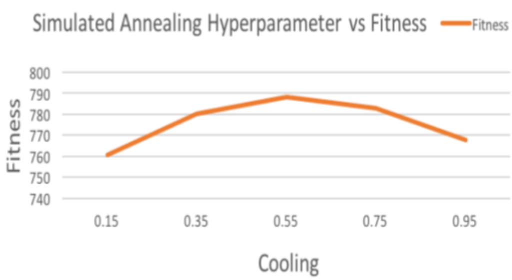


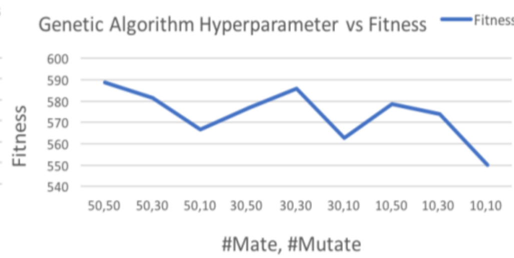


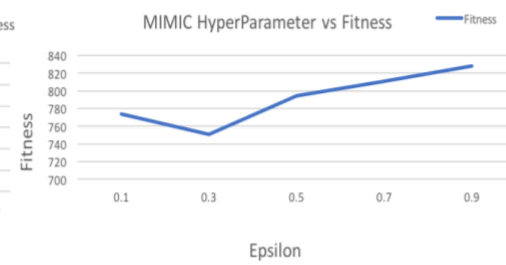


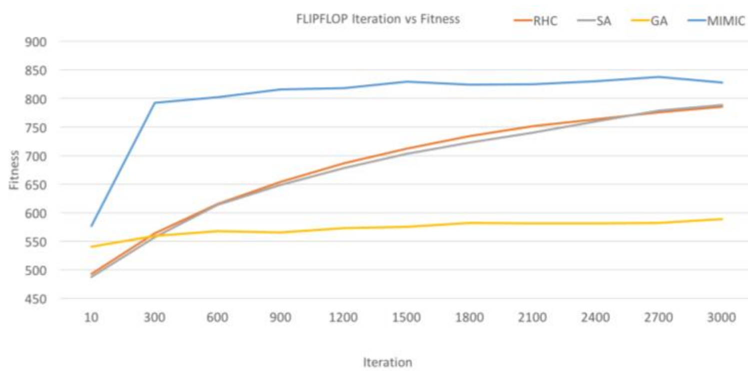
iterations	RHC Time	SA_0.15 Time	GA_100_50_30 Time	MIMIC_100_50_0.5 Tim
10	0.0001342494	0.0000455804	0.001044506	0.128999714
500	0.0015633766	0.0006190452	0.020471118	2.710435239
1000	0.0026430998	0.0038575774	0.040716963	5.075629563
1500	0.0036916038	0.0042424028	0.060304623	7.281881804
2000	0.0047700244	0.0046262684	0.078749188	9.409218289
2500	0.0057964052	0.0050188156	0.096748088	11.50242444
3000	0.0065374300	0.0053750310	0.113966971	13.59279008
3500	0.0072363264	0.0057343402	0.13149352	15.68405502
4000	0.0079130560	0.0060776188	0.148664491	17.76470291
4500	0.0086466650	0.0064247024	0.165736049	19.81327964
5000	0.0093248386	0.0067713006	0.183521588	21.83931507

FLIPFLOP



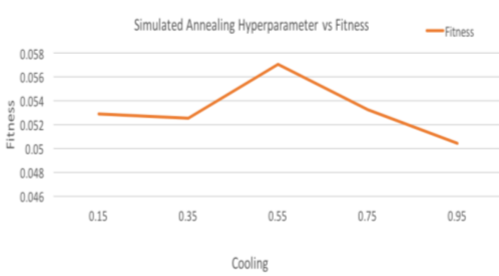


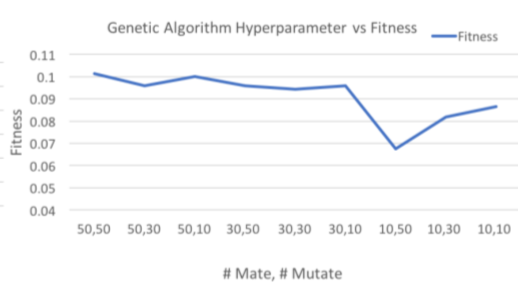


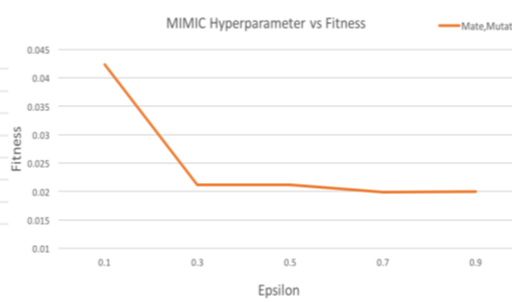


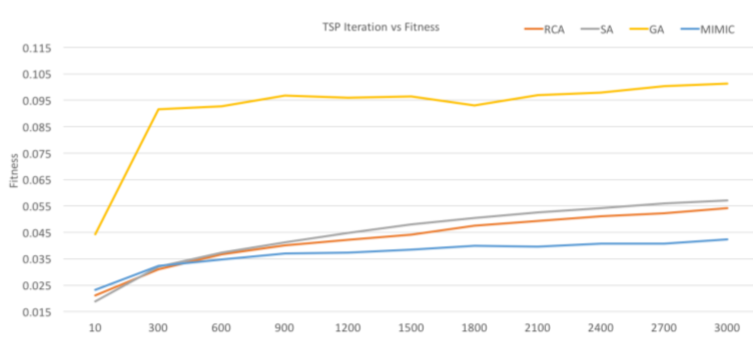
iterations	RHA Time	SA_0.55 Time	GA_100_50_50 Time	MIMIC_100_50_0.9 Time
10	0.0013131868	0.0000848230	0.016146759	18.98554423
300	0.0136201616	0.0014025308	0.112512832	227.1742803
600	0.0246763094	0.0086833672	0.199514038	423.988904
900	0.0284188732	0.0099144296	0.282368915	617.6996899
1200	0.0345154702	0.0111945762	0.360532571	811.1821851
1500	0.0374372284	0.0125162250	0.438662091	1004.588038
1800	0.0402665776	0.0139953372	0.516835592	1197.436824
2100	0.0428598376	0.0157462020	0.594347724	1390.806233
2400	0.0456774456	0.0172133550	0.671988948	1589.616333
2700	0.0487204934	0.0187740824	0.747495598	1785.62343
3000	0.0517747000	0.0202707712	0.825717747	1980.160193

TSP









iterations	RHC Time	SA_0.55 Time	GA_100_50_50 Time	MIMIC_100_50_0.1 Time
10	0.0001077974	0.0000264024	0.013767561	3.523209776
300	0.0008061458	0.0004205404	0.102845579	48.15745386
600	0.0013373992	0.0008496144	0.178479717	92.64539271
900	0.0018477220	0.0012717598	0.251439564	137.2549604
1200	0.0022565930	0.0017045538	0.321647517	181.8233112
1500	0.0026355612	0.0023093980	0.388664805	226.2398834
1800	0.0029742690	0.0029215380	0.459339177	270.6651653
2100	0.0033429536	0.0035352074	0.530354589	315.1071136
2400	0.0036992372	0.0041371364	0.596325485	359.517892
2700	0.0040473524	0.0047625496	0.660866936	403.8844761
3000	0.0043746822	0.0053398974	0.726006307	448.2881006