Machine Learning Project 2

Genetic Algorithms

I used the Flip-Flop optimization problem to showcase Genetic Algorithms. Intuitively, it makes sense that they should do well here. As the parents lock unto the bit structure and the best parents pass on their genetic make-up to the children, they will increasingly resemble the Flip-Flop maximum that we are aiming for. As both parents get better/purer a single point crossover function would produce a similarly fit child. Since the location of the bits matter and the Genetic Algorithm with single point crossover maintains this pseudo structure it results in children having similar structure to their parents. For example, if the 2 parents were 1111 and 0000 a 1 point crossover would produce 1100 and 0011 which both improved in fitness. As mutations occur and the outlying bits get flipped and retained as they have better fitness scores the population gets better collectively. Contrast this with a uniform crossover scheme in which every other bit is chosen. This has high risk high reward; consider 1111 and 0000 and their children 1010 and 0101. Looks perfect, right? However, if we go another generation down we revert back to the parents with nothing gained. Mutations may help us here by randomly choosing a point and flipping it just as it helped us in the 1 point crossover case; here however, this would be the only useful tool that the Genetic Algorithm uses and obviates the need for the parent-child relationships. If we would just mutate we would end up with the same result here as with or without the uniform alternating crossover; in fact it would be better time wise and just as effective. As the International Journal of Machine Learning and Computing (Lim, Sultan, Sulaiman, Mustapha, Leong, 2017, p. 10) says about uniform crossover "Inheritance does not rely on position" and position is crucial here. The 1 point crossover on the other hand retains the structure that this optimization problem needs. I would say that there are no local optima or minima to get stuck on; it's just a chain of bits where each correct bit-pair corresponds to a better fitness score. What characterizes a "hilly" function is that it has spots that the algorithm thinks are good and they return a reward disproportionate to the steps to get there. Starting 1/2 way up the hill and going up gets the same reward as being at the bottom and going up. Additionally, a hilly function penalizes the algorithm for leaving any peak (hence the need for Simulated Annealing) whereas in Flip-Flop every location is just as important as every other. This is why I believe there are no local optima. This is borne out in Figure 1 where

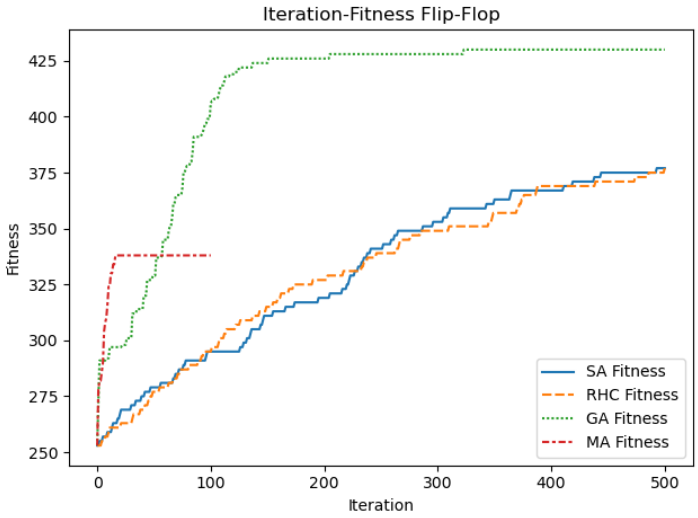

Figure 1: Iteration-Fitness Flip-Flop

the given problem size is 500. Very clearly Genetic Algorithms dominate in this case. Note how Simulated Annealing and Random Hill Climb are the same. As we mentioned there are no local optima here and the temperature factor that allows Simulated Annealing to explore does nothing here as no 1 location is more important than another. Since exploring doesn't help SA it ends up acting like RHC, in fact the exploration is what differentiates it from RHC and so without that factor making a difference it behaves just like it. Mimic does really well in the beginning even overtaking the GA but then flat-lines. This is because it is not tuned well enough and as Prof. Isbell said it

can represent a chain-rule based structure. However, it will require tuning to do better. The downside of Genetic Algorithms is the time as we see in Figure 2. The fitness score

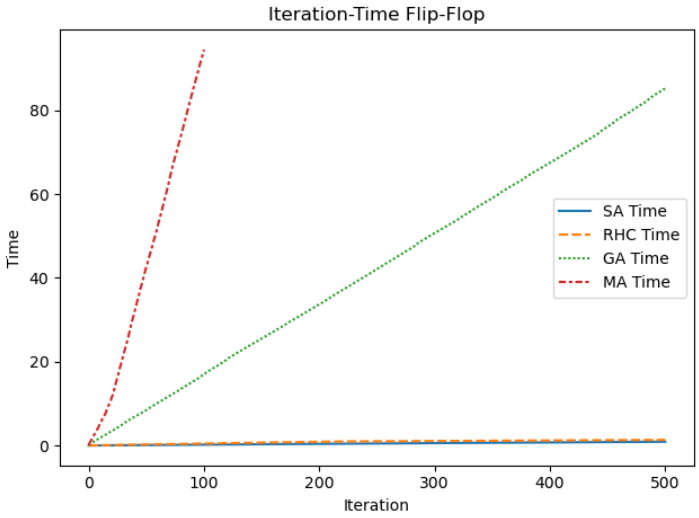

Figure 2: Iteration-Time Flip-Flop

is off by 100 or 20% compared to Simulated Annealing or Random Hill Climb; however, the time is orders of magnitude bigger which demonstrates the no free lunch concept. If we want a better fitness we pay for it in time. How badly we want that fitness will determine what we are willing to pay for it. Note that Mimic is also extremely time consuming due to all the probability calculations and despite it having 100 iterations to SA and RHCs 500 it still visibly took more time. I explored problem sizes of 100, 500, and 1000 and kept the population sizes to 50% for equality. The results (Figure 3) were very interesting. First of all they all start with a fitness of about 50%. This is because given a random bit string if it starts at 0 the next bit has a 50% chance of being 1 and 50% flipping again etc. until the end of the string. Having a bit string with all 1s would be $1/(2^{lengthofstring})$ which would be .0625 for a 4-bit string alone. So statistically the bits will alternate/be the same 50% of the time leading us to an easy 50% fitness score. The curves look similar and in fact they are. The problem vs population sizes proportionately affected the ability to solve the problem by holding the 50% population

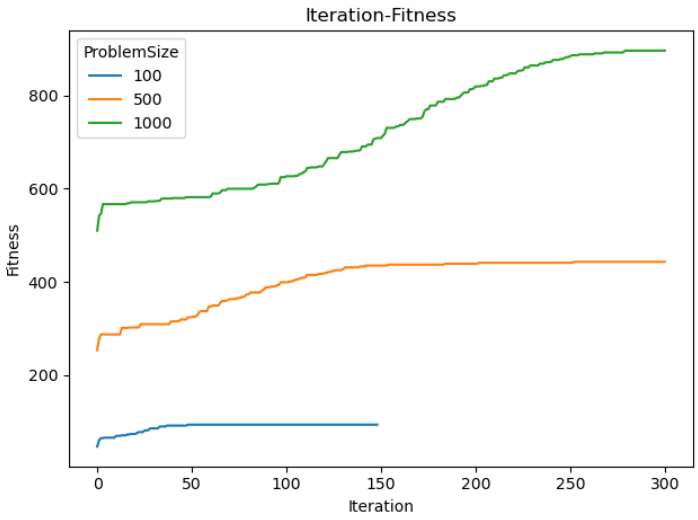

Figure 3: GA Iteration-Fitness Flip-Flop

size for equality. 100 had a max score of 93, 500: 443, and 1000: 897. These scores all have about 10% error rate and this makes sense as the mutation rate is .1 and the last mutation throws off our score as it is agnostic to the structure. Since proportionally they are equal I will focus on 500 and assume they will generalize to 100 and 1000. Interestingly, Mutation Rate does affect the Fitness but not adversely as we predicted. Digging through the Mutate code we see that it performs a swap mutate in which 2 bits exchange values. This can explain the

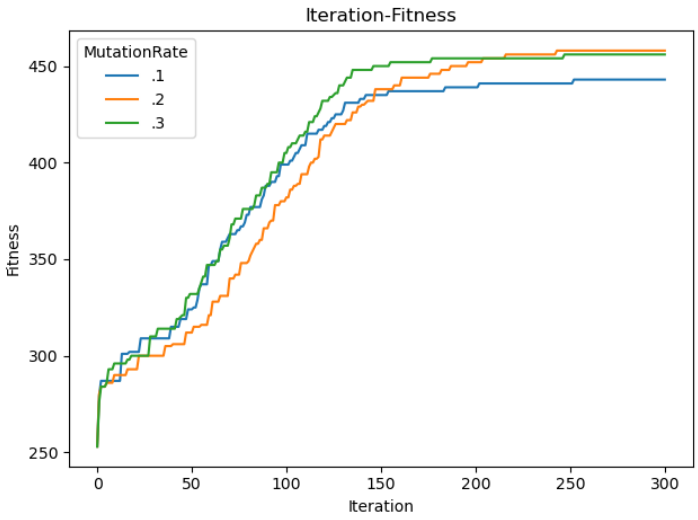

Figure 4: GA MutationRate Iteration-Fitness Flip-Flop

results we see. I was assuming the the Mutate function operated on one bit and if the

string was very pure a random single bit mutate would throw it off. However, a swap is more complex and would have to 1) pick a bit opposite the first bit and 2) have both locations pure to create a lower fitness. I seems this doesn't happen often enough to harm the score and in fact helps (though I don't understand why yet). Plotting the population sizes produced expected results with larger population sizes having a better fitness but taking longer; this demonstrates the trade off between complexity and time and the "No free lunch" theorem.

Simulated Annealing

One thing that Simulated Annealing excels at is the Continuous Peaks problem with a very small Peak. It has many local optima but only 1 global optima and Simulated Annealing exploring stage helps it do really well. I set the problem size to 100 and the `t_pct` to .05 so that the peak is only 5 bits long which makes it harder.

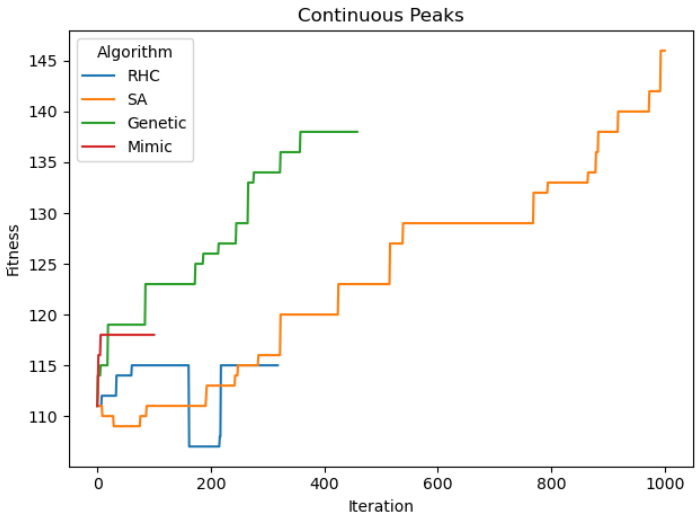

Figure 5: Iteration-Fitness

As we see SA does the best here and this is due to the exploring that SA does which allows it to find the Peak. The long flat spots indicate exploring worse places in which it keeps the best fitness function while looking for a better one. Once it finds a better place it uses that as the fitness and starts exploring again based on its temperature. The temperature set to 1 with a Geometric Decay of .99. GA does pretty well too

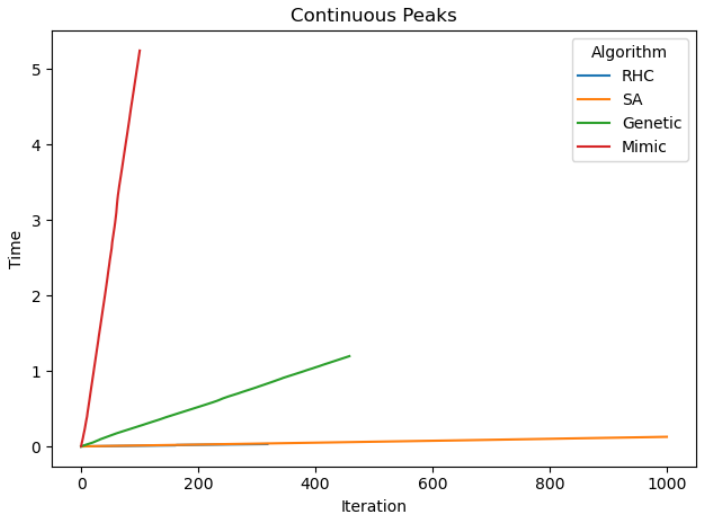

Figure 6: Iteration-Time

but not as well. This can be attributed to the fact that 1) there is a most important spot 2) it's really small. The GA does seem to be climbing the function but isn't able to pick up that some spots are more important than others; RHC predictably gets stuck in a local optima and Mimic beats everything for a short while just like in Figure 1 and then stalls out as it needs more calibration to do better. Time-wise SA also does extremely well making it the clear winner for this problem. It does well in time because

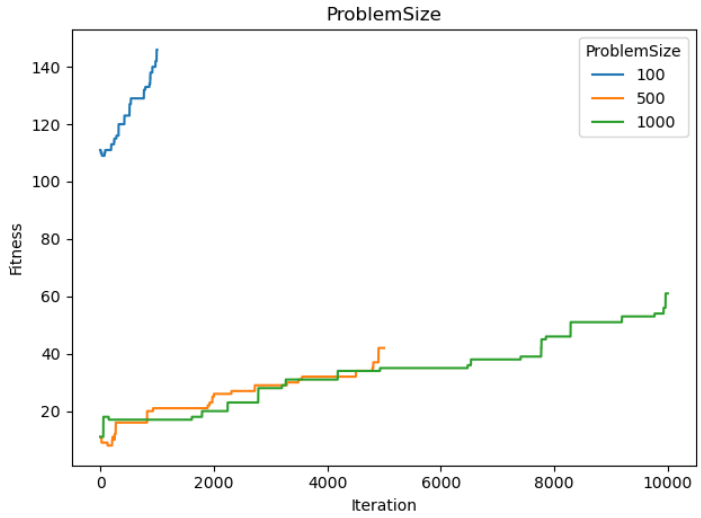

Figure 7: Problem Size

it acts like RHC which is also really fast as it just climbs the fitness function with some additional exploring which as we saw made

it so much better. GA has time consuming operations by breeding the population and mutating; Mimic even more so with its computationally intensive probability calculations.

Plotting for multiple sizes (Figure 7) leads to some disappointment. This can be attributed to bad temperatures that don't allow Simulated Annealing to explore enough as needed for the larger problem sizes. I wanted to focus on a larger problem size to allow us to find a good Temperature so I used Problem Size of 500. We increased the temperature to 100 and staggered the Temperature on a logarithmic scale to 1, 10, and 100 with the Decay set to .99. We got some interesting results. It seems there

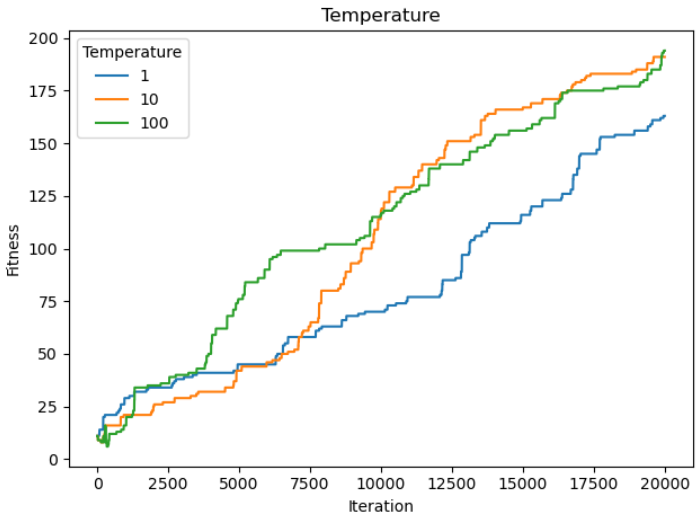

Figure 8: Temperature

were 2 problems with larger sizes for Simulated Annealing. The biggest problem is that we needed more iterations for more exploration as seen by the fact that the original Temperature/Decay Schedule worked reasonably well given enough time. Additionally, the interplay between Temperatures 10 and 100 portray the exploration very nicely. At first 100 does better as it is more open to exploring and that willingness to explore leads it to a better fitness faster. However, as time goes on it is still willing to explore a lot; as opposed to 10 which takes a little longer to find the peak, but once it does it hones in on it and finesses the fitness.

Mimic

Mimic does really well in structured problems and only needs a few iterations. This can be demonstrated in the K-Color problem that is highly structured. The graph has 200 nodes. I wanted to make an interesting problem so I made `max_colors` = 2 but `max_connections_per_node` = 4 so that there are multiple solutions and depending on the graph structure near impossible to get a perfect score. We used a population of 200 and kept 1/3 of the population. As

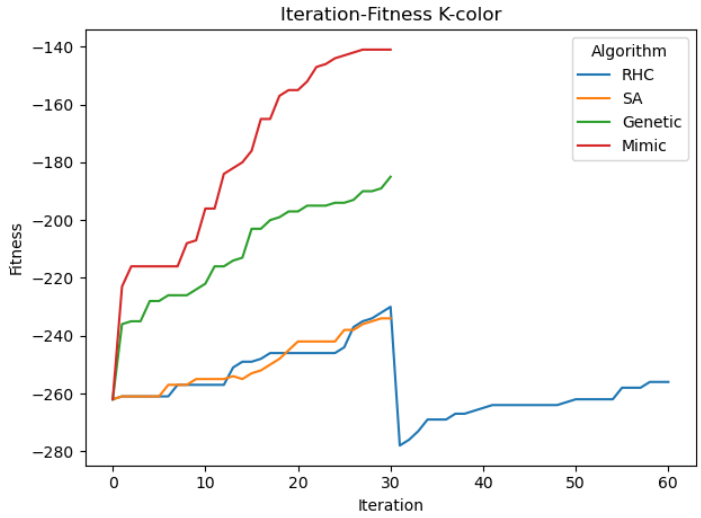

Figure 9: Iteration-Fitness K-Color

we can see Mimic does really well as it can portray the connectivity of the graph via its dependency trees. It maximizes the fitness with very few iterations via its probability evaluations. Limiting the amount of iterations can be essential in certain cases and allow Mimic to outperform other algorithms. Here however the evaluations don't take so long and Mimic as usual takes orders of magnitude longer than any other algorithm. This is due to all of the probability and dependency calculations. However, as we mentioned if the iteration cycle takes a long time – one example Professor Isbell (Isbell Littman) gave is if we are waiting for a response from a human – it can be faster as it needs fewer iterations.

Exploring Mimic in different problem sizes leads to an interesting graph. In other

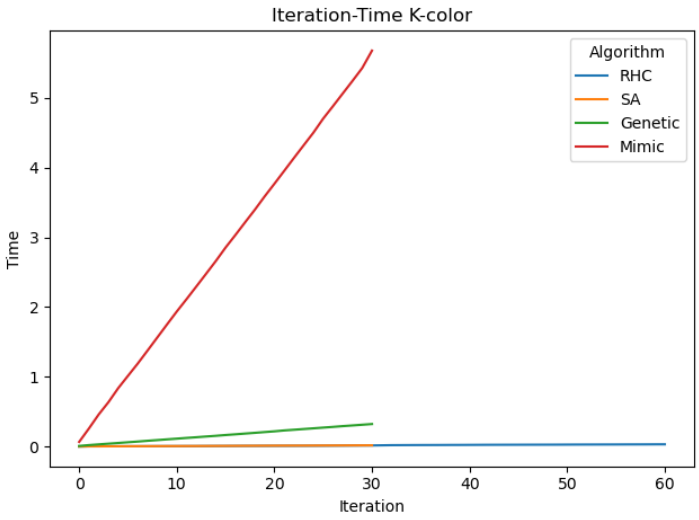

Figure 10: Iteration-Time K-Color

problems and algorithms that we explored the fitness seemed near linear to the problem complexity – assuming equivalent hyper parameters. Here however, the largest

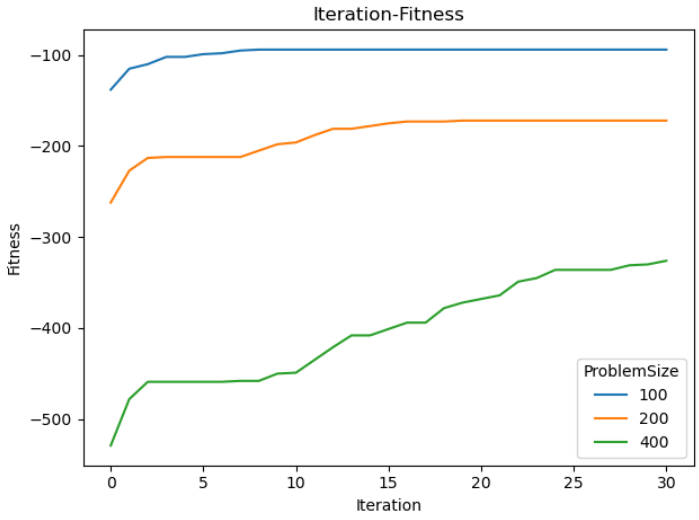

Figure 11: Mimic Iteration-Fitness K-Color

problem size has a disproportionately better fitness despite proportionately equal hyper parameters. This can be attributed to Mimics maintaining and iterating over the entire structure. So a problem size of 400 with population of 200 will do better than a problem of 100 with population of 50. Each individual in the population maintains its version of the best probability and tree structure for the entire problem. In contrast, Genetic Algorithms for a problem size of 4 would need a population of 16 to represent each possible state. So if we increase the size linearly the representing population grows exponentially. In our experiments though we only increased the population linearly to the problem size and this was able to maintain the error rate of the problem size. In Mimic however, each individual maintains and represents its own best state of the entire problem so increasing the problem size does not require us to grow the population as much as Genetic Algorithms would.

One under estimated hyper parameter I wanted to explore was the `keep_percent_list` which is the percentage of the population to keep. The reason why I found this interesting was that more/less was not necessarily better. For example, in population size, if we just focus on fitness (not time) a bigger population is better. Here, a middle ground must be found. I went .2 up and

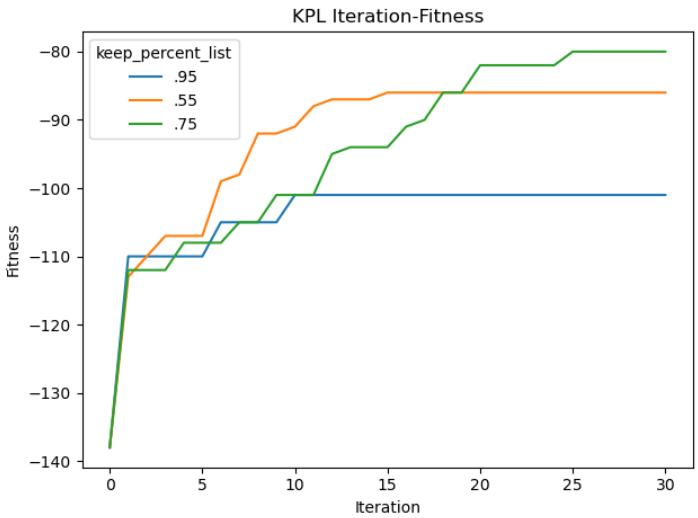

Figure 12: KPL Iteration-Fitness

down to demonstrate this. If we keep to many samples and don't generate new ones we stagnate. On the other hand, if we generate to many new samples we end up not learning from the previous population. Interestingly, keeping less of the population requires more time. I would've thought that because Mimic has to generate a new probability structure either way it would take the same amount of time. Why does gener-

ating a new probability take more time than refining based on a previous one? This requires further research.

Part 1 Conclusion

As we compared and contrasted these algorithms some things became apparent. Time wise Mimic can be the most consuming followed by Genetic Algorithms and then Simulated Annealing/Random Hill Climbing. Mimic can justify its time consuming process if there are other factors involved such as requiring input from people in which case it can be faster as it requires fewer iterations to converge. It is also very good at modeling complex structures as seen with the K-Color problem. Genetic Algorithms take less time than Mimic but more than Simulated Annealing. They are very good at what I think of as ”shallow structure” such as the Flip-Flop problem. Simulated Annealing performs well with an unstructured problem with local maxima such as Continuous Peaks. It uses its exploring stage to locate and separate a global maxima from local ones via its Temperature and Decay schedule. It is very efficient time wise.

ANN

I used the Contraceptive Method Used (Tjen-Sien, 1987) dataset from A1 and changed it to Contraceptive Used as described in A1.

For my ANN graph to work I had to change my learning rate to .1 for Random Hill Climbing and Simulated Annealing to get them in the same range as Gradient Descent whose learning rate was .002. This means that GD had to go through 50x the space than RHC and SA to get a similar error. Of course that can be optimized with momentum and we don’t need to search all spaces but this is still interesting that near equal results can be achieved with 50x less hypotheses. Additionally, it seems that there are at least 2 local optima where GD pushes through until it hits it’s last optima where it just reduces the loss as much as it can. Gradient Descent takes advantage

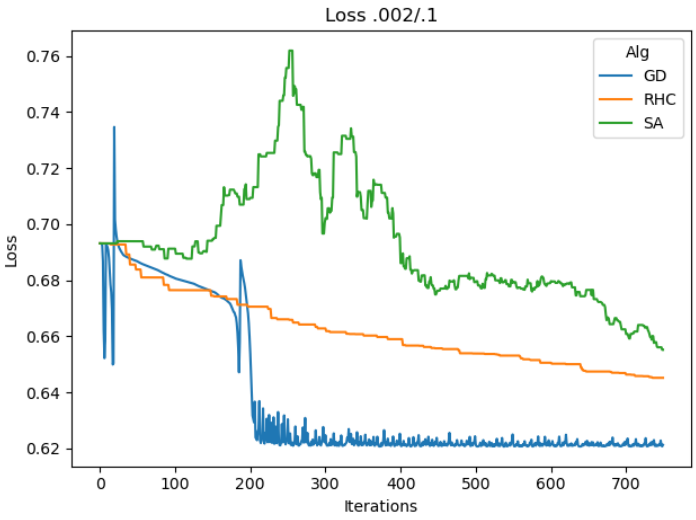

Figure 13: Loss Curve .1

of the weights continuous nature to converge faster. Simulated Annealing also very nicely demonstrates how its temperature allows it to explore in the beginning leading to an even higher error but cools down and improves its loss. Random Hill Climbing comes in with its simple approach of just trying to improve and steadily gets better.

The discrete nature of the learning rate can greatly affect the Gradient Descent Algorithm. The smaller a learning rate is the more it acts as if it’s in a continuous space. As the rate gets bigger it will act and be more discrete. A larger learning rate as seen in Figure 13 will oscillate back and forth in proportion to the learning rate as it hones in and drives the error down. As we can see the smallest learning rate produces the smallest line. Gradient Descent calculates the steepest descent and it achieves this by altering the weight in proportion to the derivative. A larger learning rate will drive down the error fastest; however, it will also have bigger issues as it starts to oscillate around the minima. I zoomed in the graph to better show this effect. As we see the largest learning rate gains momentum the fastest. The way I view it is in terms of acceleration and weight. The larger rate will have a faster acceleration but less weight and is more prone to its direction being reversed. A smaller learn-

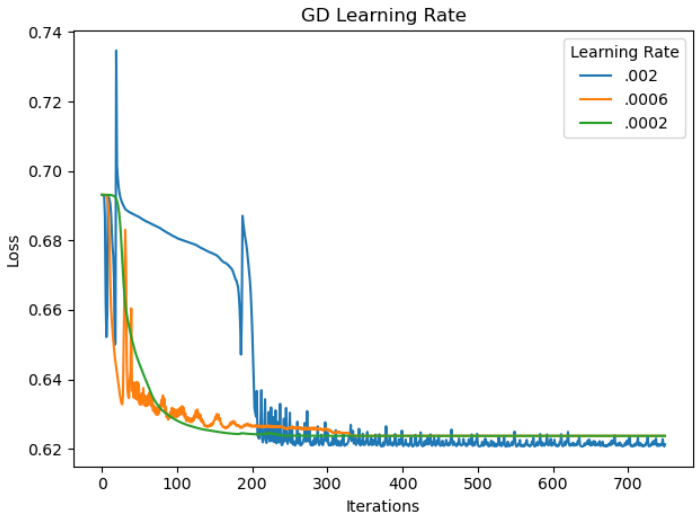

Figure 14: GD Learning Rate

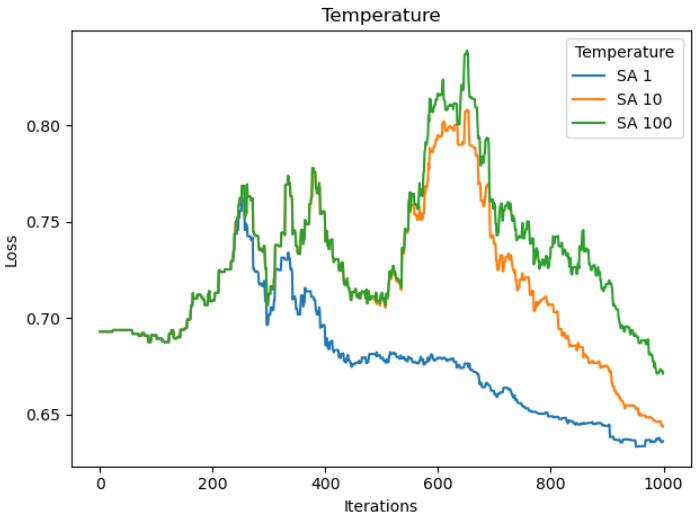

Figure 16: Temperature-Loss Curve

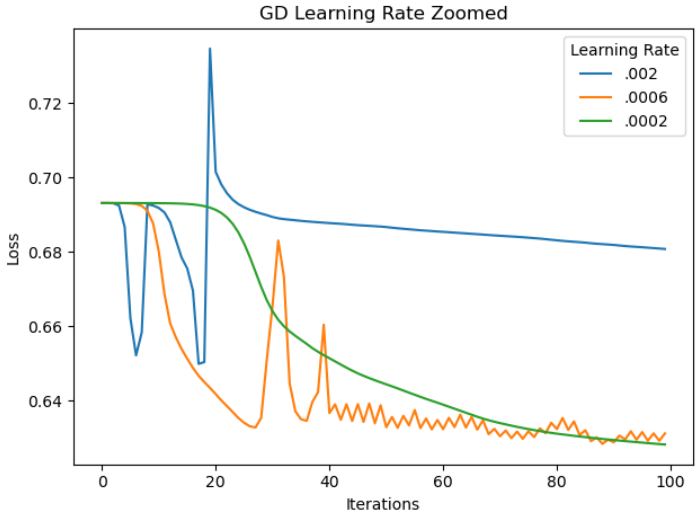

Figure 15: GD Learning Rate Zoom

ing rate will accelerate slower but has more weight and once on course veers much less. In fact, it hits the lowest point first.

I plotted Simulated Annealing with logarithmically larger Temperatures to explore its effects on my Neural Network. We can learn some things from this graph. Larger temperatures will explore more which leads to a higher error rate during the exploration stage. As they cool down they will end up with (hopefully) the global optima. Choosing a good initial temperature and decay schedule is key. It seems that the middle one with an initial temperature of 10 is the best in this case. The Decay schedules were equal (Geometric .99) so that we can attribute this to the temperature. 10 is the best as it explores nearly as much as 100 yet has an error very close to 1. 1 doesn't explore much which may lead it to miss some global optima and 100 may get there in the end but it's decay schedule is to slow given the high initial temperature and takes longer to converge to the minima. 10 is perfectly placed between these 2 factors of exploration and convergence and is the best choice in this case. If the problem was larger or smaller however it may be that a larger or smaller temperature would work better. Another interesting observation I had during my hyper parameter tuning was the implicit affect of learning rate on the exploration. A larger rate would produce a larger error for the exploration. The same amount of hypothesis are tested but a larger learning rate will create a larger scale of exploration despite it being the same amount of hypothesis.

The next hyper parameter I explored was the population size in Genetic Algorithms. Predictably, a higher population resulted in a lower score first. However, this is not true across the board. If we can't lower the error anymore a smaller population can catch up given enough iterations as seen in population size of 40. Yet, having a larger population is not always a good thing. A larger population will take more time to run so a

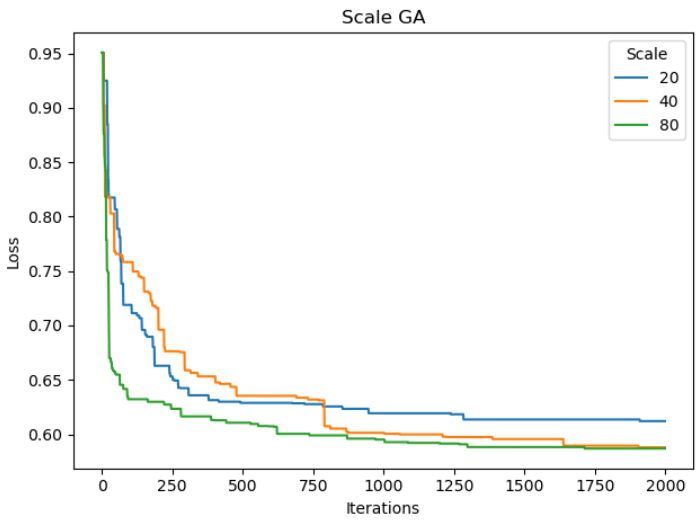

Figure 17: Loss Curve GA

cost benefit analysis of the time vs loss can be performed to see how much time are we losing for what amount of fitness. Also, the population size of 20 reduces the loss faster than 40. One reason for this can be due to *crowding*. As Mitchell (2013 p. 259) tells us that *crowding* is where an individual that is more fit takes over a large percentage of the population and can reduce its diversity slowing performance. A larger population size is less susceptible to this as it is more diverse from the outset and will therefore have more diverse offspring.

Part 2 Conclusion

We explored the affects of replacing Backpropagation with Random Hill Climbing, Simulated Annealing and Genetic Algorithms. We saw how Backpropagation does better in a continuous space and learned how to make a discrete space closer to a continuous space. We saw the affects of temperature and how it increases error while exploring but decreases as it cools down. Additionally, we observed the interesting phenomenon of how learning rate is implicitly tied to the scale of exploration. In Genetic Algorithms we explored the affect of population sizes on the loss and iterations.

References

Isbell, C., Littman, M. (n.d.). Retrieved from https://gatech.instructure.com/courses/159302/pa practical-matters-two?module$_{item}id$ = 1427498

Lim, S. M., Sultan, A. B., Sulaiman, N., Mustapha, A., Leong, K. (2017, 2). Crossover and Mutation Operators of Genetic Algorithms. International Journal of Machine Learning and Computing, 7(1), 10. Retrieved 3 7, 2021, from http://www.ijmlc.org/vol7/611-A8.pdf

Mitchell, T. M. (2013). Machine Learning. McGraw Hill Education.

Tjen-Sien, L. (1987). Contraceptive Method Choice Data Set. Retrieved from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Contracep