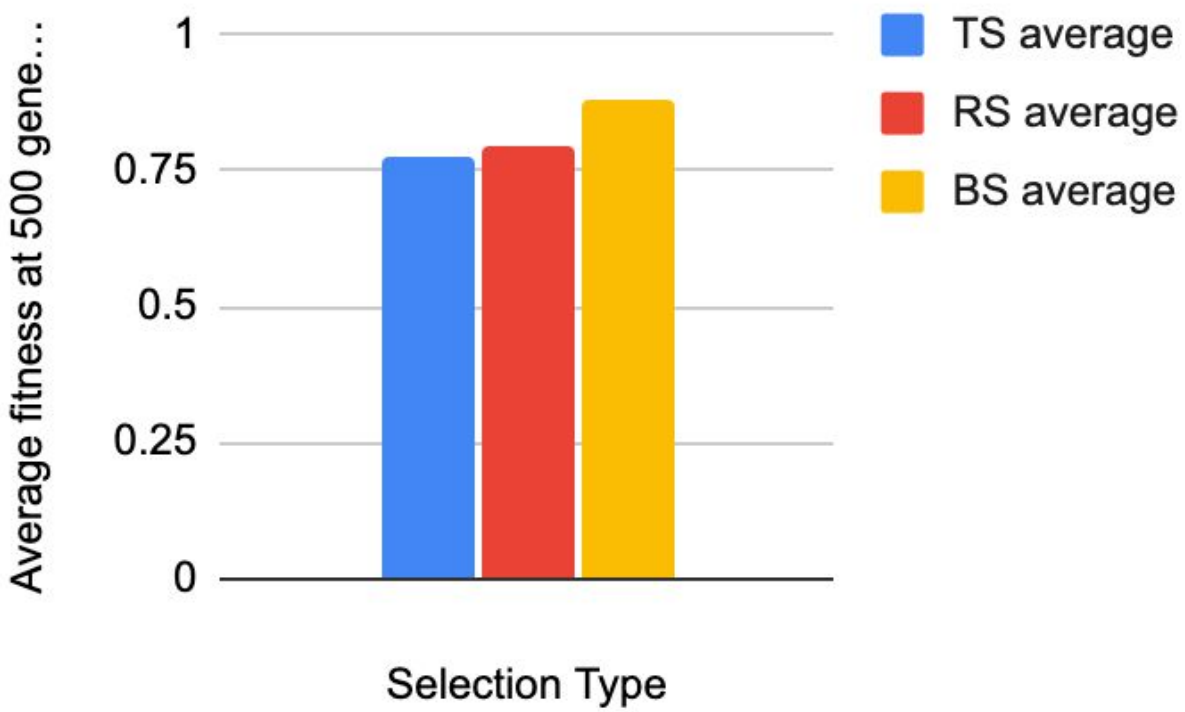
TS average fitnesses at gen 500:
• 0.763
• 0.741
• 0.833
• 0.719
• 0.826
• Total average: 0.7764

RS average at 500:
• 0.8
• 0.756
• 0.821
• 0.785
• 0.818
• Total average: 0.796

BS average at 500:
• 0.782
• 0.977
• 0.886
• 0.883
• 0.88
• Total average: 0.8816



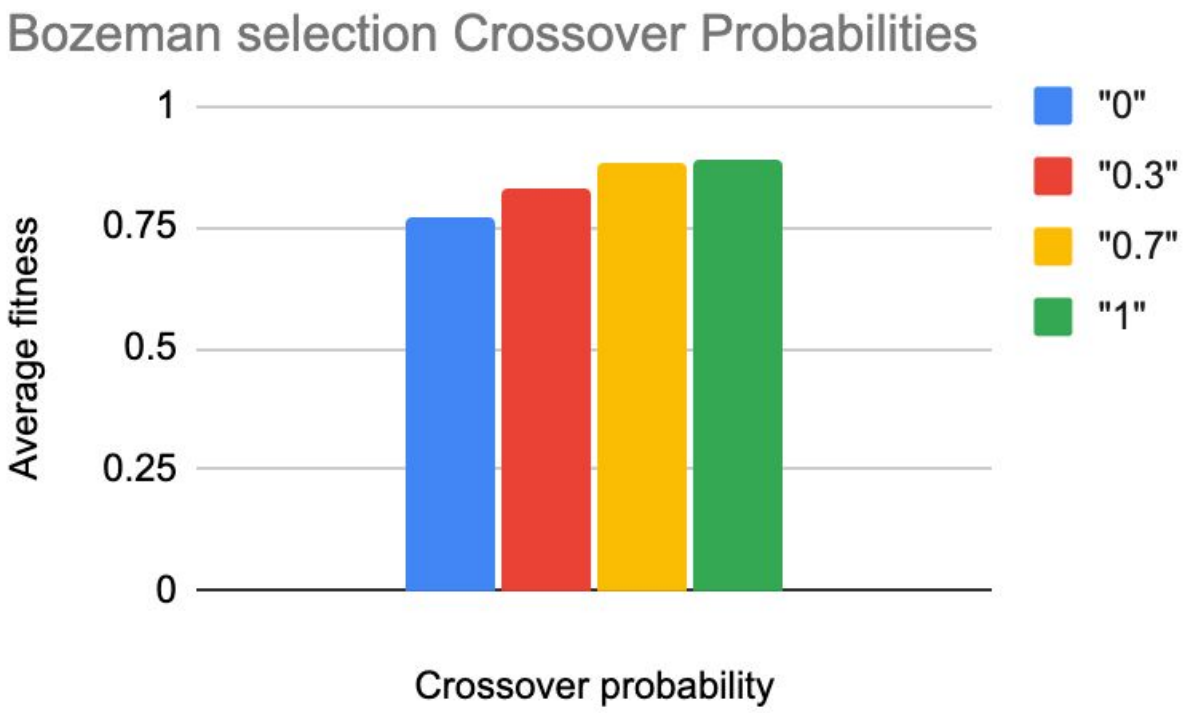
From these calculations it's clear that Bozeman preforms the best. I think this is because more fit individuals are weighted more heavily and/or accurately. While ranked selection accounts for which individuals are more fit than others, the difference in fitness between two individuals could be massive but the selection process wouldn't account for anything more than the fact that one is better. Bozeman, on the other hand, uses to weighted fitnesses to account for relative differences in fitness between individuals and ergo gives greater preference to fit individuals.

Bozeman Crossover probs:
Average fitness at 0.0:
- 0.703
- 0.706
- 0.799
- 0.832
- 0.796
- Total Average: .7672

Average at 0.3:
- 0.871
- 0.794
- 0.806
- 0.839
- 0.844
- Total Average: .8308

Average at 0.7:
- 0.782
- 0.977
- 0.886
- 0.883
- 0.88
- Total average: 0.8816

Average at 1:
- 0.896
- 0.898
- 0.864
- 0.902
- 0.881
- Total Average: .8882



It looks like the crossover probability and average fitness of individuals are directly proportional. In the case of string matching, I think cross-over is very helpful to address mismatched characters as the chances of mutation mutating the right character into the desired character are very slim, however, doing crossover can address multiple characters at once.

Average fitness for Mutation 0:
- 0.671
- 0.674
- 0.695
- 0.676
- 0.732
- Total Average: .6896

Averages at mutation = 0.001:
- 0.654
- 0.616
- 0.697
- 0.814
- 0.655
- Total Average: .6872

Average at mutation = 0.005:
- 0.636

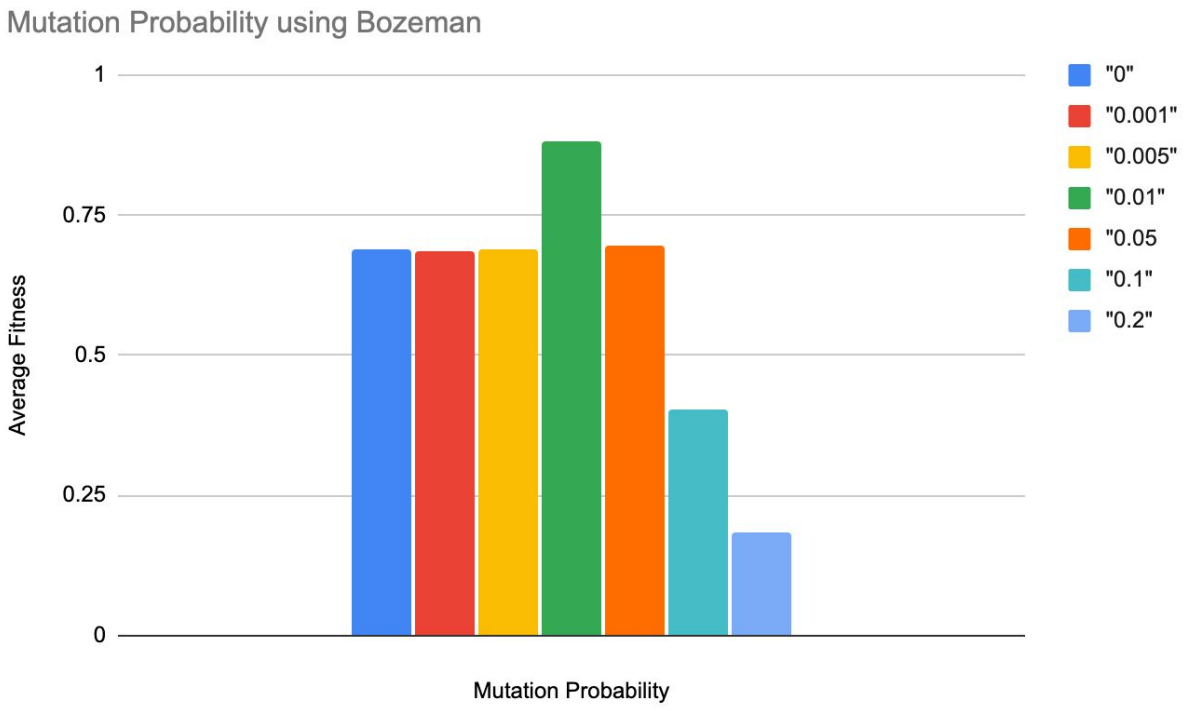
• 0.719
• 0.694
• 0.616
• 0.774
• Total Average: 0.6878

Average at mutation = 0.01
• 0.782
• 0.977
• 0.886
• 0.883
• 0.88
• Total average: 0.8816

Average at 0.05:
• 0.707
• 0.637
• 0.682
• 0.738
• 0.718
• Total Average: 0.6964

Average at 0.1:
• 0.406
• 0.381
• 0.402
• 0.404
• 0.420
• Total Average: 0.4026

Average at 0.2:
• 0.169
• 0.190
• 0.193
• 0.170
• 0.204
• Total Average: 0.1852



It looks like there is a definite sweet spot for mutation probability; too low and crossover by itself has a hard time expanding the state space and creating new combinations of characters. However, too high and the excessive mutation becomes counter productive and prevents finding a solution by changing promising candidates too much

Alternative fitness evaluations:

- You could evaluate by number of correct words within the string, however this would stratify both the population and the learning process, and by that I mean learning would happen in less frequent leaps
- You could still evaluate using correct letters but perhaps do some sort of euclidean distance calculation instead of just using the flat value?
- I'm honestly having a hard time coming up with anything that makes sense