Knapsack experiment Full Report: Team Sleepless

Sam Miller, Richard Stangl, and Elsa Browning

November 7, 2016

Introduction

In this set of experiments, we explore the performance of a simple genetic algorithm and compare it to the performance of the basic hill-climber algorithm. Our algorithm is essentially an expanded version of hill-climber that works on an entire population instead of on individuals. The hill-climber starts with a randomly generated answer, and then repeatedly mutates the best answer found so far, keeping the better of the original and the tweaked answer at each step. Our method was to create a population of random solutions, creating our first generation, and then mutate that generation to create our second generation. We then combined the two generations, sorted them, and took the better half of the individuals. We repeat this process on each successive population until we find the ideal solution or until we reach the maximum number of tries. Our algorithm is much slower than the hill-climber algorithm in running time because it's running on populations instead of on individuals.

In this experiment, we used the lexi-score method to score our solutions. Lexi-score is a method Nic McPhee created. This method takes a potential solution to the knapsack problem, dumps the items out of the knapsack, shuffles them, and then scores all items that fit in the bag in the order that the appear in the shuffled list. The score is a subset of the items in the bag meaning the knapsack could be overweight. This is still a valid representation of the algorithms' performance as we used lexi-score for both hill-climber and our evolutionary algorithm. We used this because we wanted to be able to include all of the runs in our graphical representation rather than throwing away overweight solutions.

We used McPhee's mutation method as well, as described in "Knapsack experiments sample write-up", in which the mutation changes the probability of flipping a bit.

Experimental setup

We initially started with a `max-tries` value of 1,000 for the 3 different types of knapsack problems:

- `knapPI_11_20_1000_4`
- `knapPI_13_20_1000_4`
- `knapPI_16_20_1000_4`
- `knapPI_11_200_1000_4`
- `knapPI_13_200_1000_4`
- `knapPI_16_200_1000_4`

Half of these are 20 item problems, and half are 200 item problems. We did 5 independent runs of each treatment on each problem.

As we progressed, we switched to focusing on instances of the type knapsackPI_16 variety. We utilized 5 of the 20 item instances:

- `knapPI_16_20_1000_1`
- `knapPI_16_20_1000_2`
- `knapPI_16_20_1000_3`
- `knapPI_16_20_1000_4`
- `knapPI_16_20_1000_5`

and later we used 5 of the 1,000 item instances:

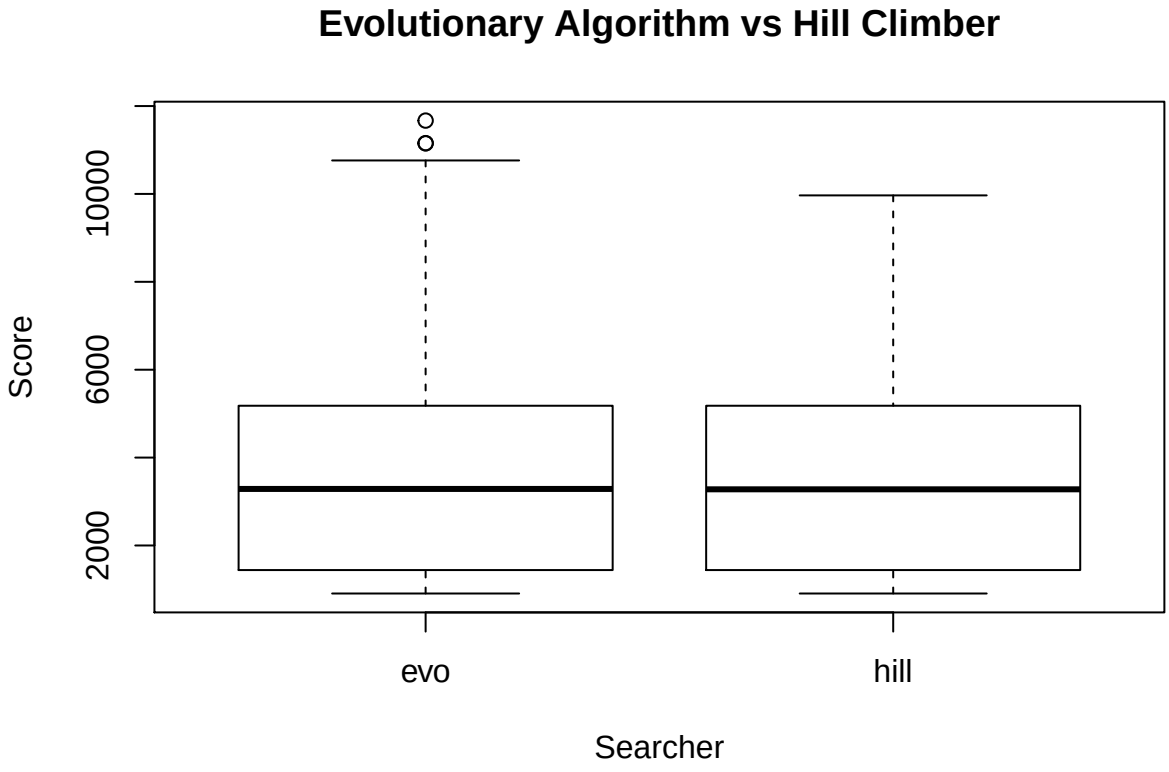
- `knapPI_16_1000_1000_1`
- `knapPI_16_1000_1000_2`
- `knapPI_16_1000_1000_3`
- `knapPI_16_1000_1000_4`
- `knapPI_16_1000_1000_5`

We increased the `max-tries` value to 10,000 for these in order to see if there was a definite difference in the two algorithms. We ran this data 1,000 times to make sure we had consistent data. The names of the knapsack instances are abbreviated, e.g., `k_16_1000_1000_1`, in diagrams below. Also abbreviated are the search method names: 'evolutionary-algorithm' is shortened to 'evo', and 'hill-climber' is shortened to 'hill' in all variables involved in the following graphs.

Results

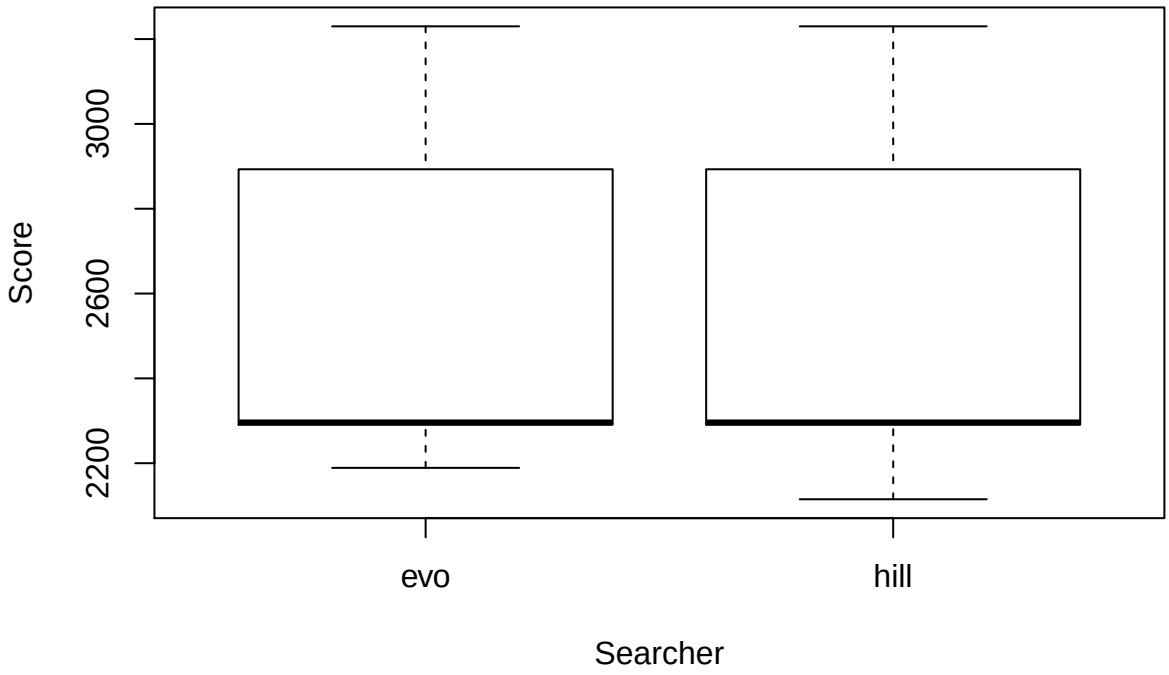
A comparison of our searcher to basic hill-climber

In our first graph, we show the comparison of our evolutionary algorithm and the basic hill-climber algorithm on the 3 different types of knapsack problem instances:



Here we can see that the algorithms mostly performed the same, but our evolutionary algorithm had a few outliers that were better than the hill-climber. Looking into this further, we discovered that our data performed the same on the knapsack problems 11 and 13, but our algorithm outperformed hill-climber on problem 16. This is where our few outliers came from on our graph.

Investigating this further, we generated data solely for instances of problem 16. Here, we plot the results of 1,000 runs with 10,000 `max-tries` on five of the 20 item problems:



It is difficult to see any difference between the two, as they have similar performance. Here is a summary:

Hill Climber

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2115    2291    2296    2576    2893    3230
```

Evolutionary Algorithm

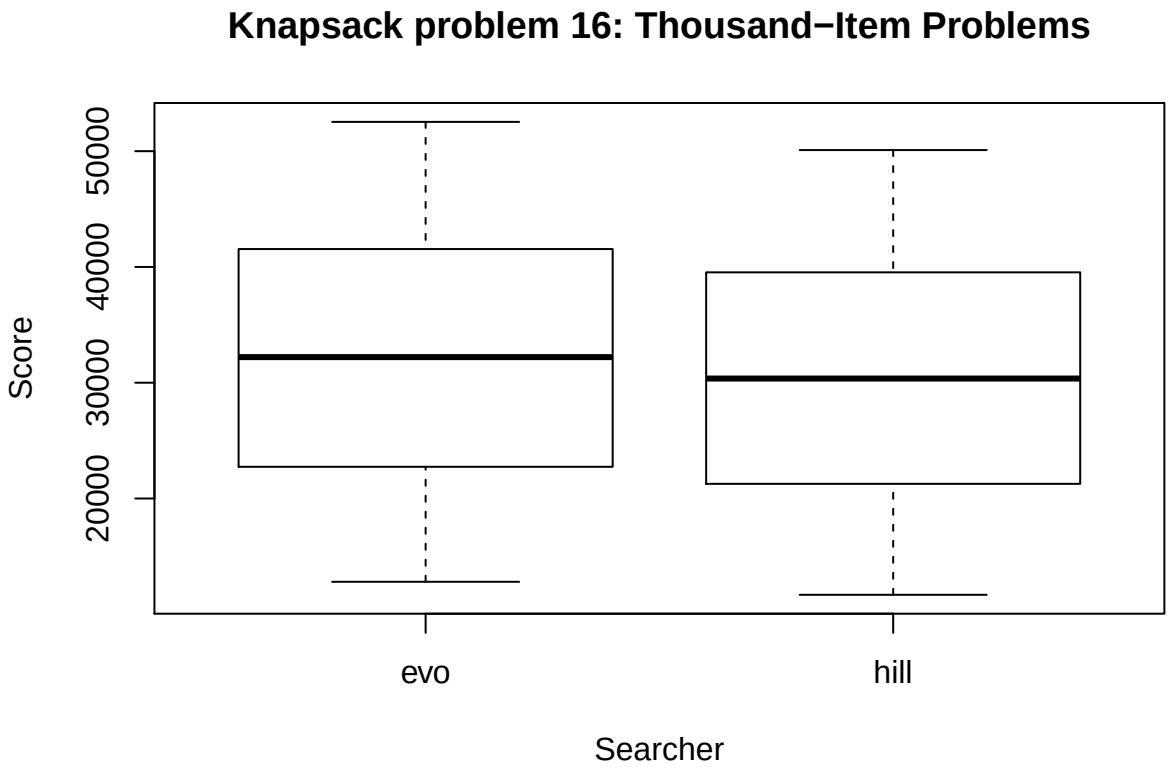
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2189    2291    2296    2580    2893    3230
```

```
    Pairwise comparisons using Wilcoxon rank sum test

data:  twenty_item$Score and twenty_item$Search_method

     evo
hill 0.2

P value adjustment method: holm
```

In this case, our algorithm is *not* statistically significantly different. We received a p value of 0.2. We think that we received such a high p-value because of our number of max-tries; we must have found the absolute max (or close to it) with both algorithms, meaning neither algorithm could outperform the other because they both found the best possible result.

We decided to investigate this a little further by running the search-methods on some of the problem 16 instances with 1,000 items. This meant we would most likely not reach the absolute max because we had more complex problems which would mean we could see if our algorithm was reaching higher than hill-climber consistently or if they truly were about the same and not statistically significant.

Here we plot the results of 50 runs with 10,000 `max-tries` on 5 of the 1,000 item problems:



The graph shows that, overall, we are performing slightly better than hill-climber. A summary of our results:

Hill Climber

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  11680   21270   30360   30370   39530   50090
```

Evolutionary Algorithm

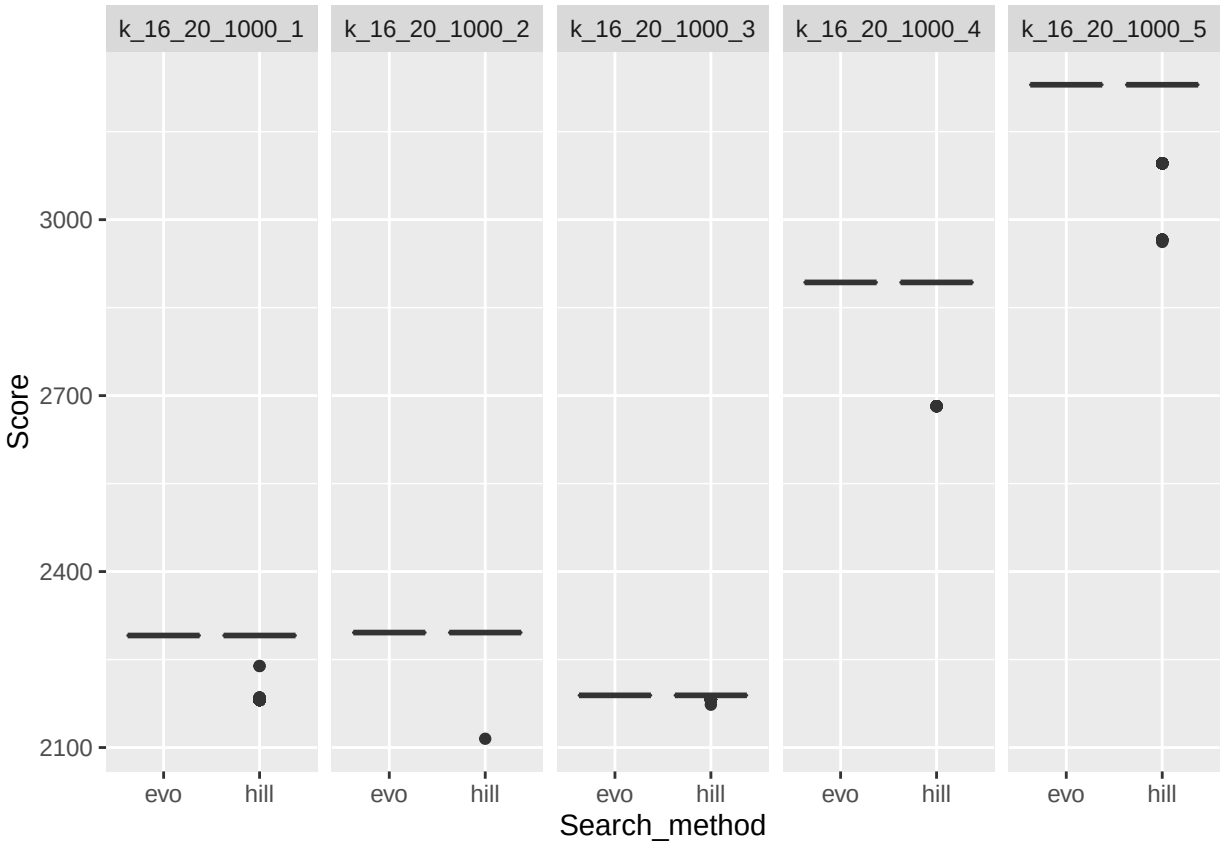
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  12810   22740   32210   32250   41540   52530
```

We are clearly performing better, *but are we finding a difference that is statistical significant?* Below we run a wilcoxon-test to see if our p value has improved:

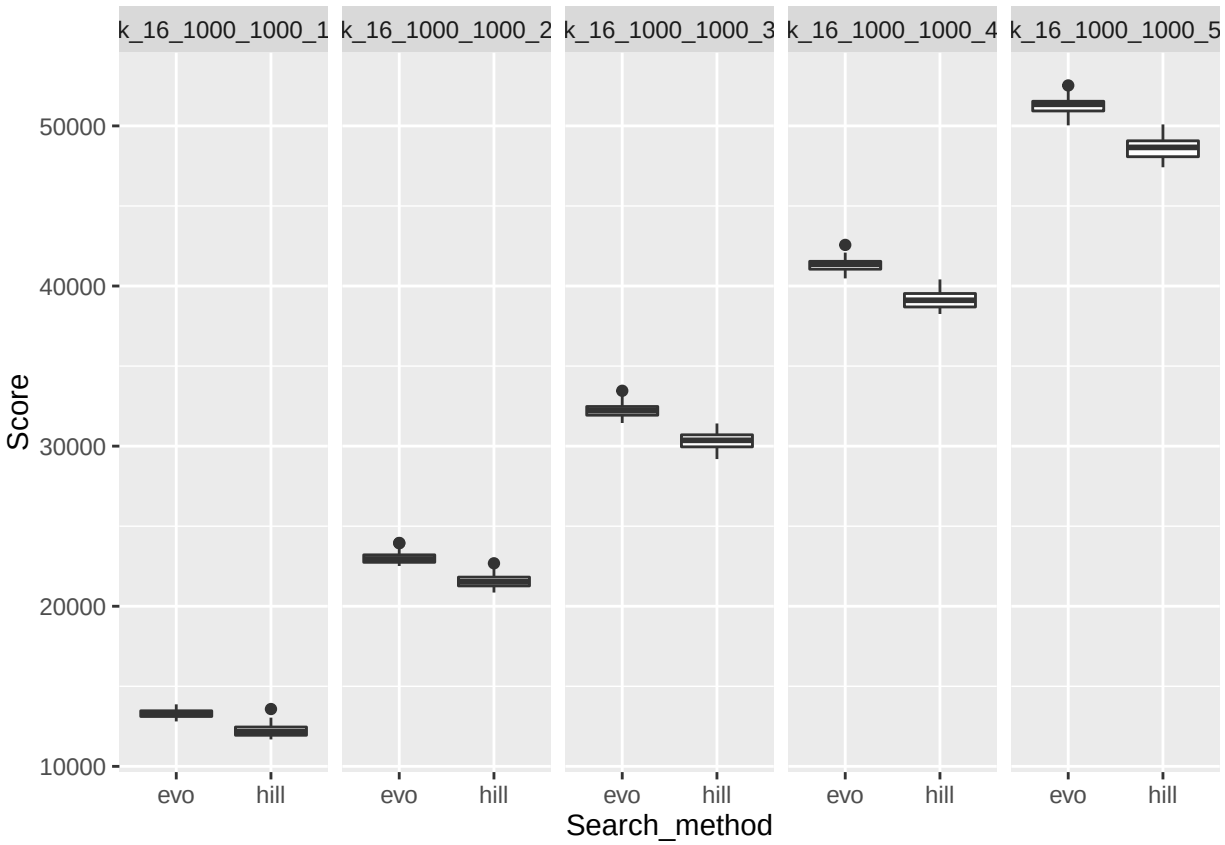
```
    Pairwise comparisons using Wilcoxon rank sum test

data:  thousand_item$Score and thousand_item$Search_method

     evo
hill 0.00013

P value adjustment method: holm
```

Our p-value is 1.3^{-4} which is more significant than our last experiment which included problems of all knapsack types. We can conclude there is a fairly significant difference between the two algorithms, especially when looking at the more complex problems.

Here is a comparison of score for each search method for the 1,000 runs of the 20 item instances:



Here is a comparison of score for each search method for the 1,000 runs of the 20 item instances:



How do things change by max evals?

When we do not reach the absolute max of a problem, we can see a significant difference in data betweeen our evolutionary algorithm and the hill-climber algorithm. This means that if we cut down on max evals, we will likely see an increase in the statistical significance between the search-methods. This is because the evolutionary algorithm reaches a better score slightly faster because it explores more optima than hill-climber. Sometimes hill-climber can also get stuck, and our algorithm is better here because having a population of individuals allows us to explore more local optima in the hopes of finding the absolute maximum.

Conclusions

Our algorithm runs much slower than the hill-climber algorithm. This is because we are essentially running hill-climber on an entire population instead of on a single individual. However, our evolutionary algorithm is slightly better than hill-climber in that it can achieve higher scores with the same inputs. If both algorithms reach the absolute maximum of a problem, we will not see a statistically significant difference in our data because we are only checking the highest score at the end of the runs. Despite this, our algorithm will outperform hill-climber in all cases and this is easiest to see in the complex problems where finding the absolute maximum is much more difficult. For improvements to our experiment, we should run eveything again with fewer max evals to see if we can find when our algorithm reaches the absolute maximum compared to when hill-climber reaches the absolute maximum.