Knapsack Experiments

Molly Grove and Aaron Lemmon

February 19, 2016

Introduction

In this lab, we explored three different search techniques on a set of three knapsack problems. We gathered data for each problem with 1,000 maximum steps (the maximum number of scored answers). We also gathered data with 10,000 steps for a particularly difficult problem (knapPI_16_1000_1000_3).

We score generated answers using the total value if the total weight is at or under capacity. If the total weight is over capacity, the score is the negative of the total weight.

One of the methods we used, called random_search, was Nic's random search method. This method generates a random answer for each of the max-tries times and compares it with the current best. It returns the best answer of the answers generated.

Another method we tried was a basic hill-climbing algorithm, called hill_climb. This algorithm starts by generating a random answer, then proceeds to either add or remove a random item from the knapsack for each of max-tries times. For each iteration, only the answer with the better score is kept as a starting point for the next iteration. Each item was equally likely to be "toggled", meaning removed if it was in the knapsack and vice versa. After all of the iterations, the algorithm returns the last answer generated.

Our third method, called random_restart, involved performing hill-climbing several times and choosing the best answer among the hill-climbs. Each hill-climb starts with a newly generated random answer. In order to keep the maximum number of scored answers the same as the other methods, the number of steps for each hill-climb is set to be max-tries divided by the number of hill-climbs.

Experimental setup

We tested the methods using three knapsack problems:

- `knapPI_11_20_1000_4`
- `knapPI_13_200_1000_4`
- `knapPI_16_1000_1000_3`

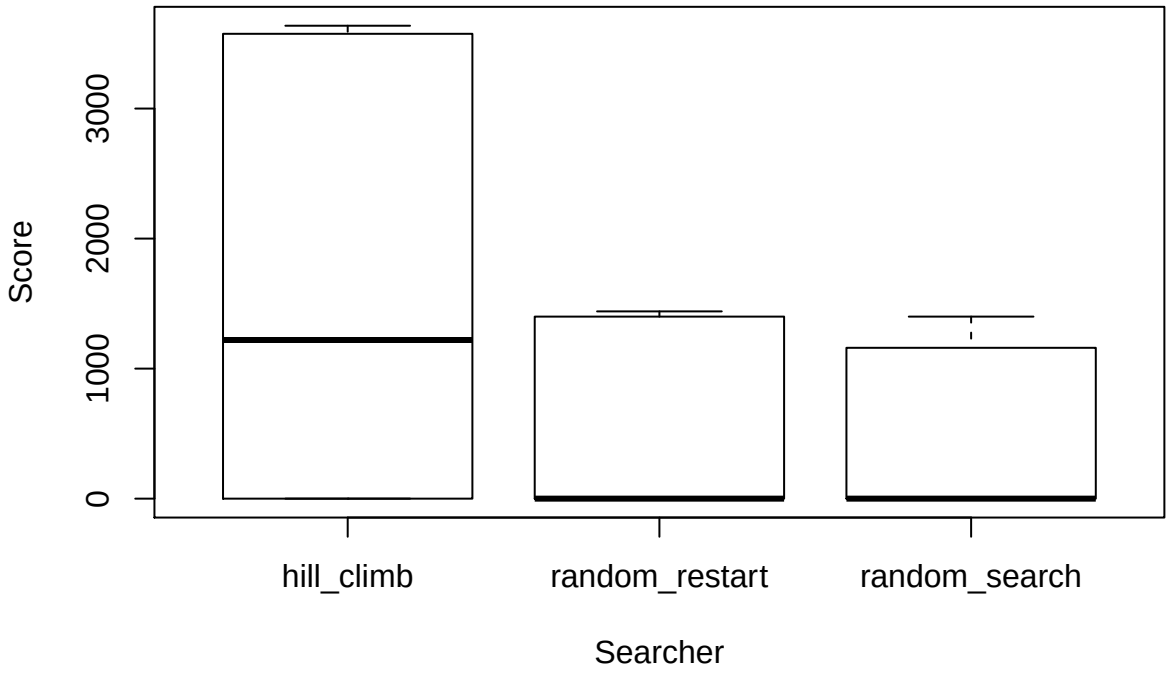
We first tried all three problems with 1,000 total steps. Because we noticed knapPI_16_1000_1000_3 never gave a positive score, we tried again with 10,000 total steps. We generated data for 30 runs for each problem/method combination.

Results

A basic comparison of the searchers

The plot directly below shows the distribution of scores over all the knapsack problems seperated by method. For this plot, all negative scores were converted to 0, since negative values indicate answers that are over capacity. Each method run took 1,000 total steps.

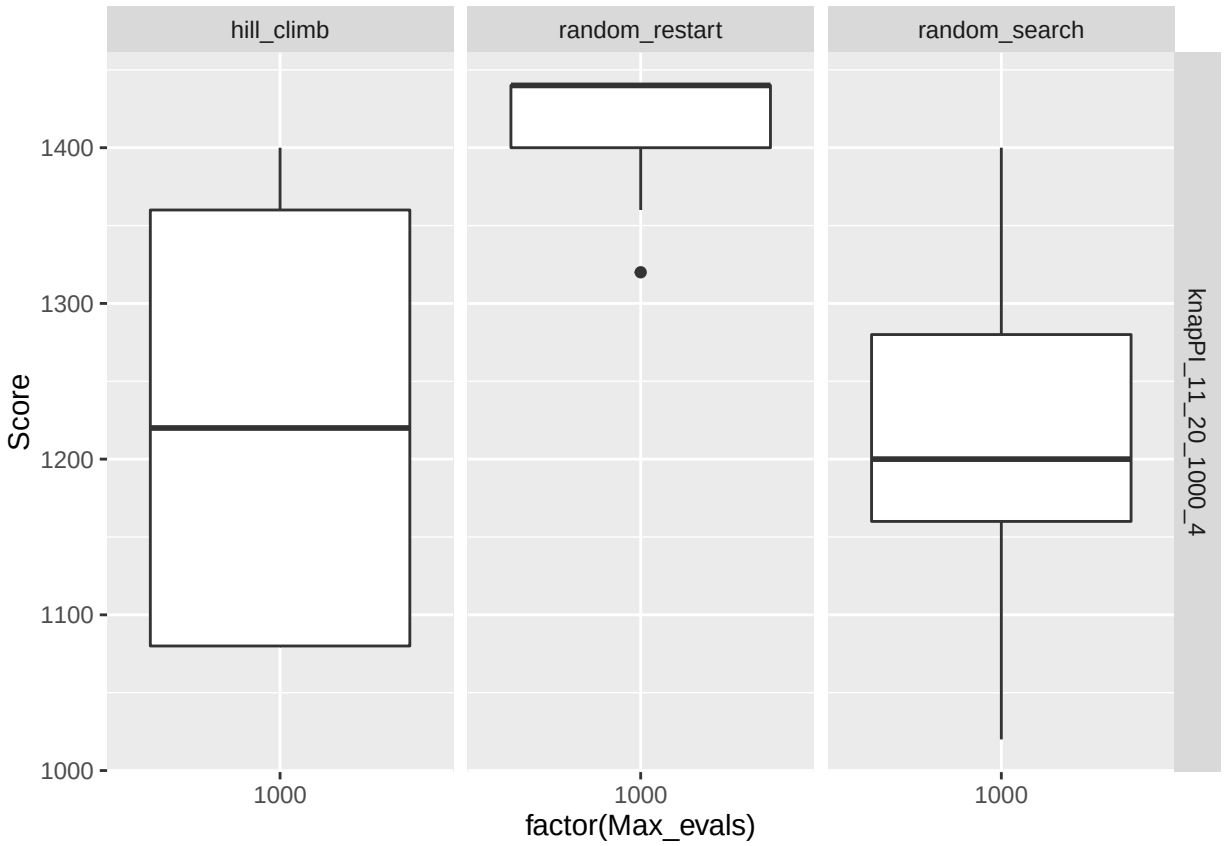
```
data_30_runs <- read.csv("knapsackproblem1000.txt", sep="")
data_10000_steps <- read.csv("knapsackproblem10000.txt", sep="")

data_30_runs$Non_negative_score = ifelse(data_30_runs$Score<0, 0, data_30_runs$Score)

plot(data_30_runs$Non_negative_score ~ data_30_runs$Search_method,
     xlab="Searcher", ylab="Score")
```



Overall, the hill_climb method appeared to be better than the other two. For both random_restart and random_search, the median scores were near 0.

A comparison of methods for knapPI_11_20_1000_4 with 1,000 steps

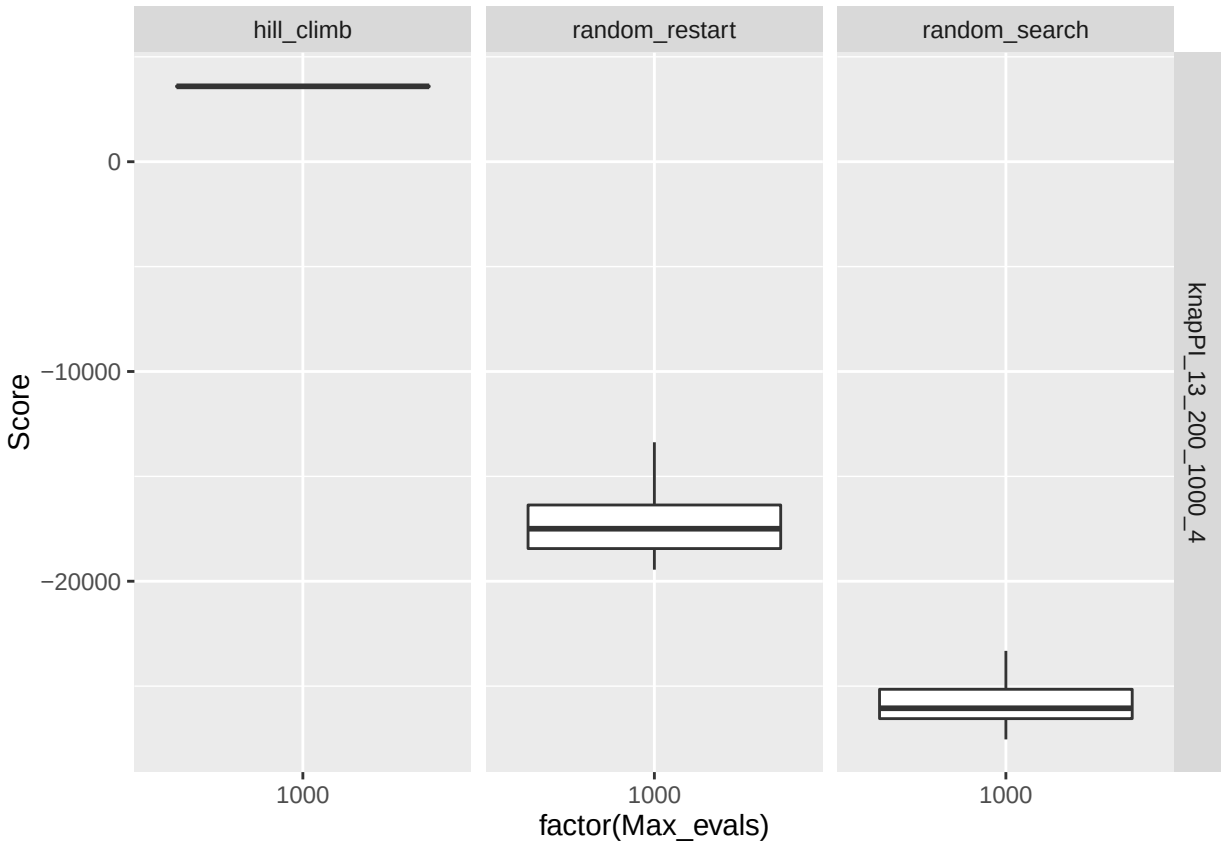
```
library("ggplot2")
ggplot(subset(data_30_runs, Problem=="knapPI_11_20_1000_4"),
       aes(x=factor(Max_evals), y=Score, group=Max_evals)) +
  geom_boxplot() + facet_grid(Problem ~ Search_method)
```



This is the only case where random_restart performed better than hill_climb. All methods performed reasonably well and obtained positive scores.

A comparison of methods for knapPI_13_200_1000_4 with 1,000 steps

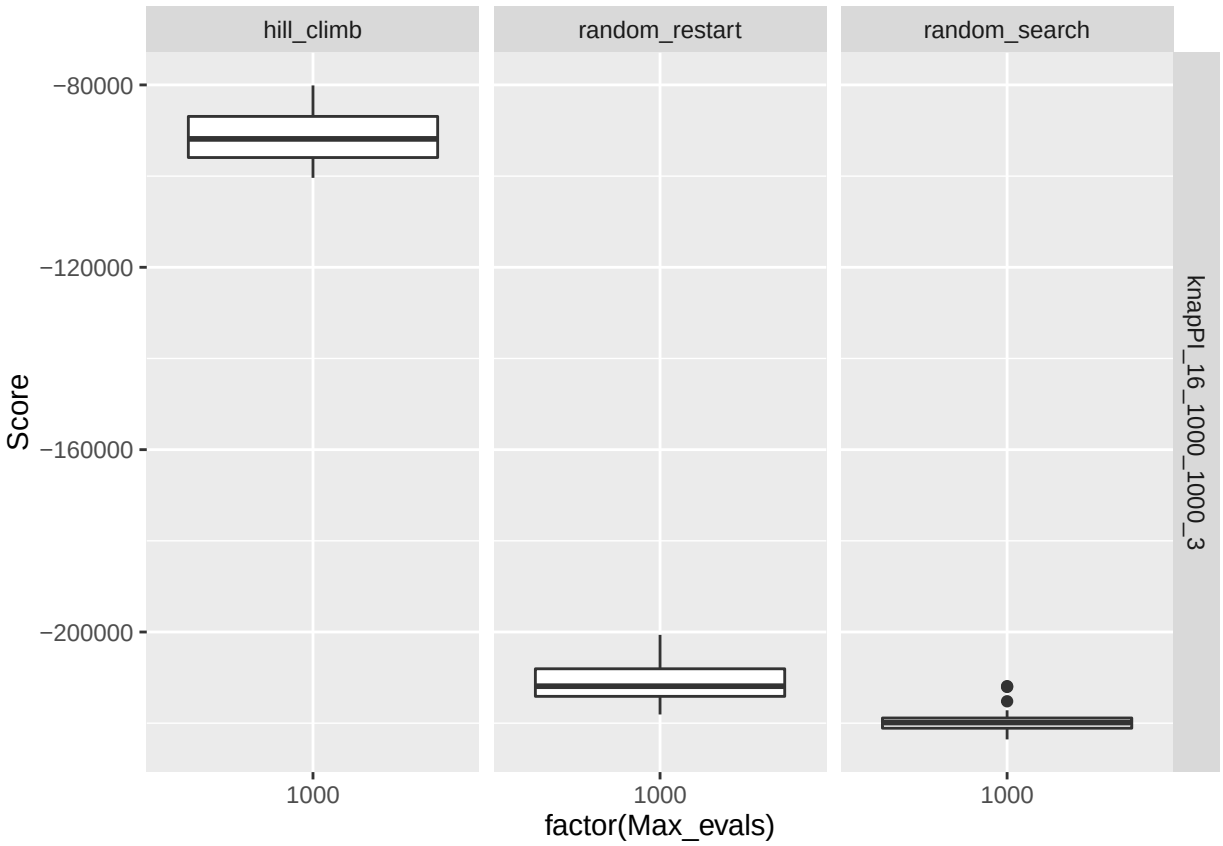
```
ggplot(subset(data_30_runs, Problem=="knapPI_13_200_1000_4"),
       aes(x=factor(Max_evals), y=Score, group=Max_evals)) +
  geom_boxplot() + facet_grid(Problem ~ Search_method)
```



The hill_climb method gave results that were around 3000. The random_restart and random_search methods both returned negative scores, although random_restart was not as bad as random_search.

A comparison of methods for knapPI_16_1000_1000_3 with 1,000 steps

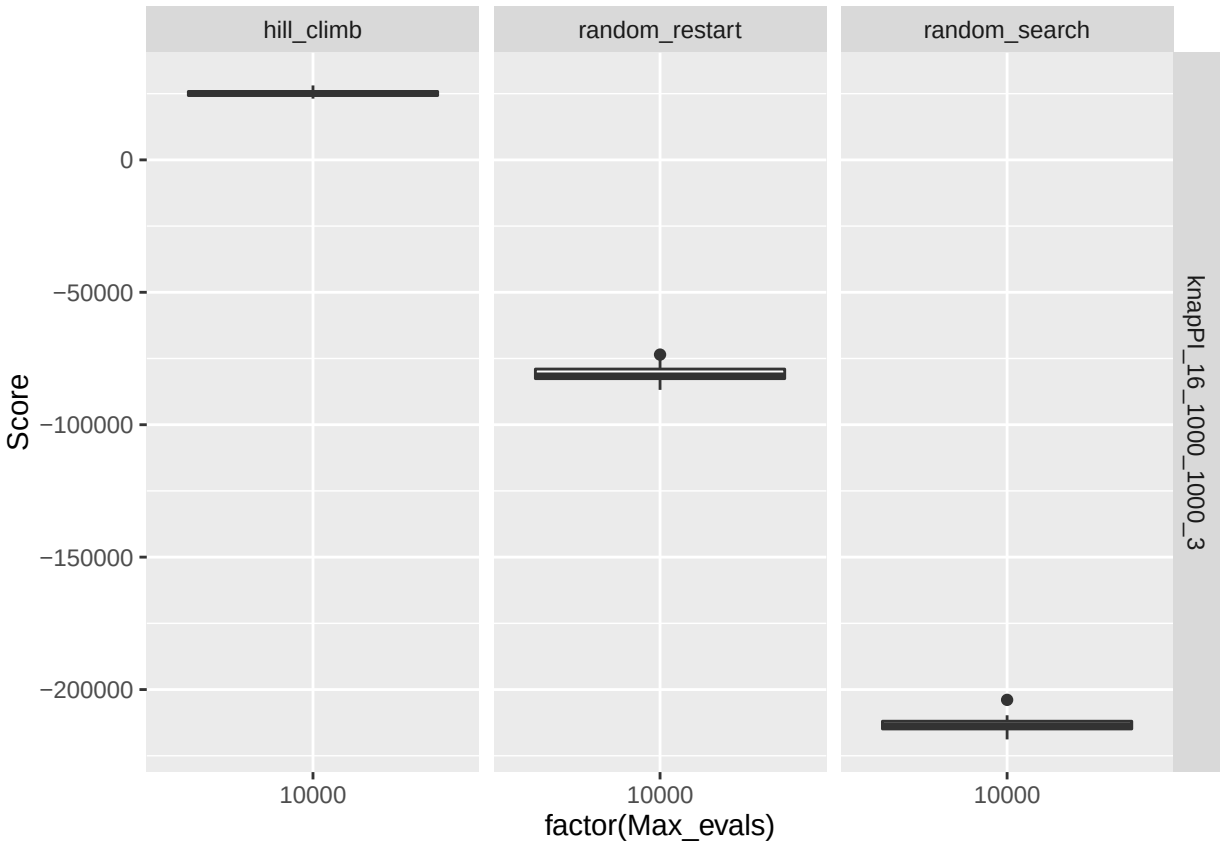
```
ggplot(subset(data_30_runs, Problem=="knapPI_16_1000_1000_3"),
       aes(x=factor(Max_evals), y=Score, group=Max_evals)) +
  geom_boxplot() + facet_grid(Problem ~ Search_method)
```



On this knapsack problem, all three methods performed very poorly. No method was able to achieve a non-negative score. However, the hill_climb method did score better than the other two, which appeared to have similar performances.

A comparison of methods for knapPI_16_1000_1000_3 with 10,000 steps

```
ggplot(data_10000_steps,
       aes(x=factor(Max_evals), y=Score, group=Max_evals)) +
  geom_boxplot() + facet_grid(Problem ~ Search_method)
```



This is a repeat of the previous plot, but with 10 times the iterations. With this many iteration, hill_climb was able to get a positive score, while the other two methods remained negative. The random_restart appears to have improved with the additional iterations, while random_search didn't seem to do much better than before.

Pairwise comparisons using Wilcoxon rank sum test

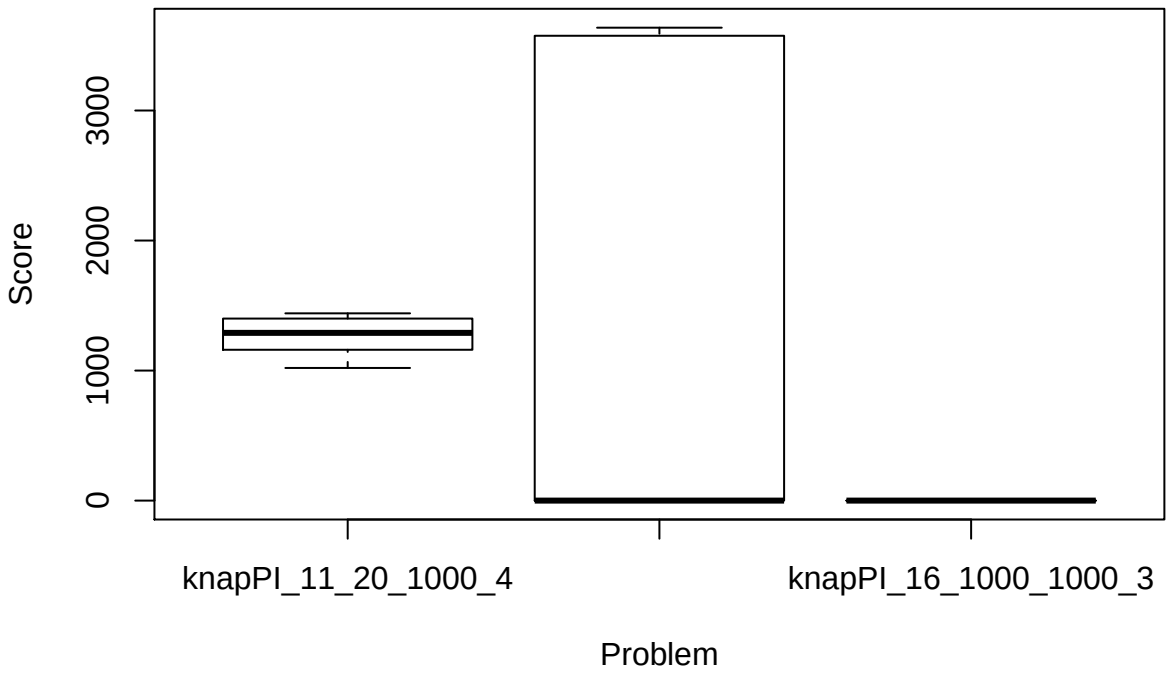
```
pairwise.wilcox.test(data_30_runs$Non_negative_score, data_30_runs$Search_method)
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test
##
## data:  data_30_runs$Non_negative_score and data_30_runs$Search_method
##
##                hill_climb random_restart
## random_restart 3.7e-05    -
## random_search  1.1e-07    0.14
##
## P value adjustment method: holm
```

These pairwise tests show that there is a statistically significant difference between hill_climb and the other two methods. However, it may be possible that there is no statistically significant difference between random_search and random_restart since they produce a value of .14.

A comparison of scores obtained per problem

```
plot(data_30_runs$Non_negative_score ~ data_30_runs$Problem,
     xlab="Problem", ylab="Score")
```



For this plot, all negative values were converted to 0. Additionally, this plot only shows the scores from the runs with 1,000 iterations. This plot shows that there was a small spread of positive scores for the first problem, a large spread of scores (with a median of 0) for the second problem, and only scores of 0 for the third problem.

Recursive Partitioning

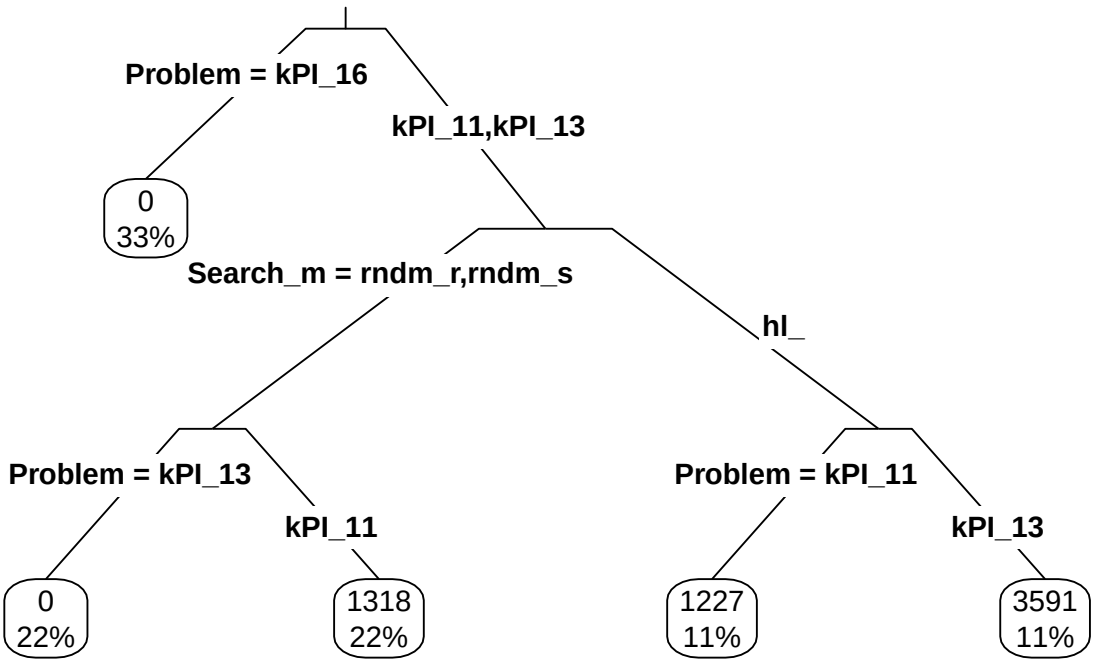
Below is the `rpart` tree for our data from all problems and search methods, with 1,000 total steps.

```
library("rpart")
library("rpart.plot")

rp <- rpart(Non_negative_score ~ Search_method + Problem + Max_evals, data=data_30_runs)
rp
```

```
## n= 270
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 270 352519600.00  828.2111
##    2) Problem=knapPI_16_1000_1000_3 90         0.00    0.0000 *
##    3) Problem=knapPI_11_20_1000_4,knapPI_13_200_1000_4 180 259918500.00 1242.3170
##      6) Search_method=random_restart,random_search 120  53054280.00  659.0000
##       12) Problem=knapPI_13_200_1000_4 60         0.00    0.0000 *
```

```
##       13) Problem=knapPI_11_20_1000_4 60    940560.00 1318.0000 *
##      7) Search_method=hill_climb 60  84371260.00 2408.9500
##       14) Problem=knapPI_11_20_1000_4 30    484266.70 1226.6670 *
##       15) Problem=knapPI_13_200_1000_4 30     19363.37 3591.2330 *
```

```
rpart.plot(rp, type=3, extra=100)
```



The third problem is separated out right away, indicating that the data for that problem was significantly different. Aside from the third problem, the tree suggests that the most significant difference is between hill_climb and the other search methods. This is probably because hill_climb did much better than the other two search methods for the second problem.

Conclusions

Our results varied widely between problems. For the first problem, random_restart was the best search method, with hill_climb and random_search giving similar scores. For the second problem, hill_climb was the only method that gave any positive scores, with random_restart performing better than random_search. Initially, the third problem gave only negative scores for all search methods. This caused us to re-run the third problem with 10,000 total steps. This gave much better scores. Although only hill_climb gave positive scores, random_restart still performed significantly better than random_search.

We think what happened with the third problem was that the first time, the algorithm simply did not have enough steps to become positive. Each step adds or removes one element and compares the scores of the new and old answers. This does not change when the score is negative, so when the algorithm tries adding an element and the score is negative, that step is basically wasted. This could have also been what happened in the second problem. The random_restart search method did not do as well as hill_climb in this case because each hill climb within random_restart only had 1/10 of the number of steps of the hill_climb search method, since the steps in random_restart were divided by 10.