Main Memory Management

A comparison of first fit versus best fit

Kevin Jonaitis
36393693
CS 143B
Project 2

Introduction

The purpose of this project is to implement a main memory manager for variable size partitions using linked lists. This project will also compare different allocation strategies using a simulation.

The project simulates main memory by using an array of integers that can be allocated and deallocated. A driver would randomize allocations and deallocations, varying the size and distributions of the allocations.

For this project in particular, we will test and compare **first fit** and **best fit** against each other.

First fit finds the first spot in memory that can fit a piece an allocation, and places it there. Best fit goes through all of memory, finding the best utilization of an allocation, and places it there.

Hypothesis

It is expected that best fit will perform better than first fit with regards to memory utilization. This is because it's always trying to minimize "holes" in memory, and thereby maximize the usage of memory.

However, since best fit must scan through every hole in memory to find the best one, it will **always** search more holes than first fit.

Therefore, the general trend we should see is best fit will always utilize more memory, but first fit will always take less time to search.

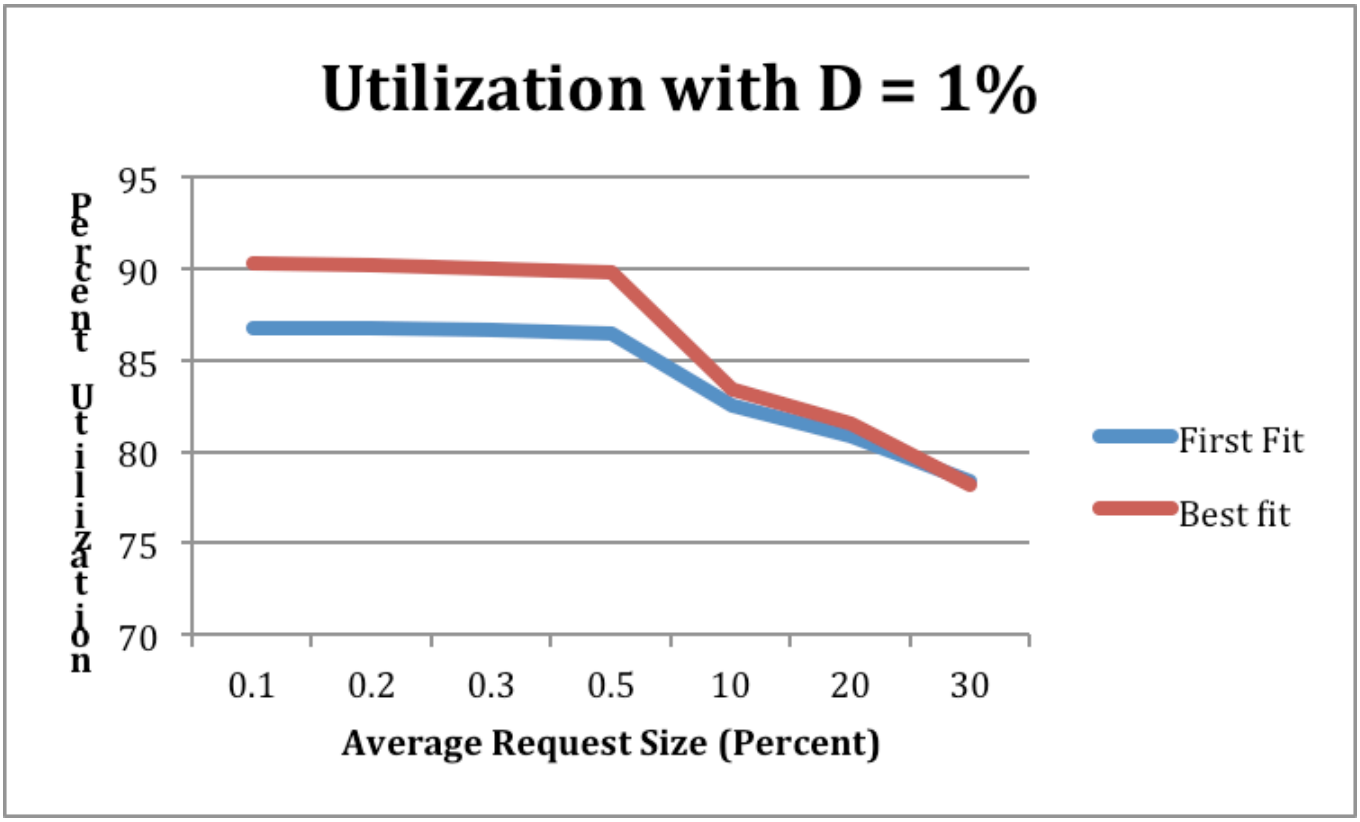
I also believe that as we increase **a**, we will see a general decrease in the utilization, as it'll become harder and harder to fill "large" allocations into small holes.
I also believe that an increase in **d** will cause **best fit** to increase the gap in efficiently with regards to first fit, because the more "variable" data will find more appropriate spots, while in first fit it will go into less-than-optimal spots that could be used for better data.

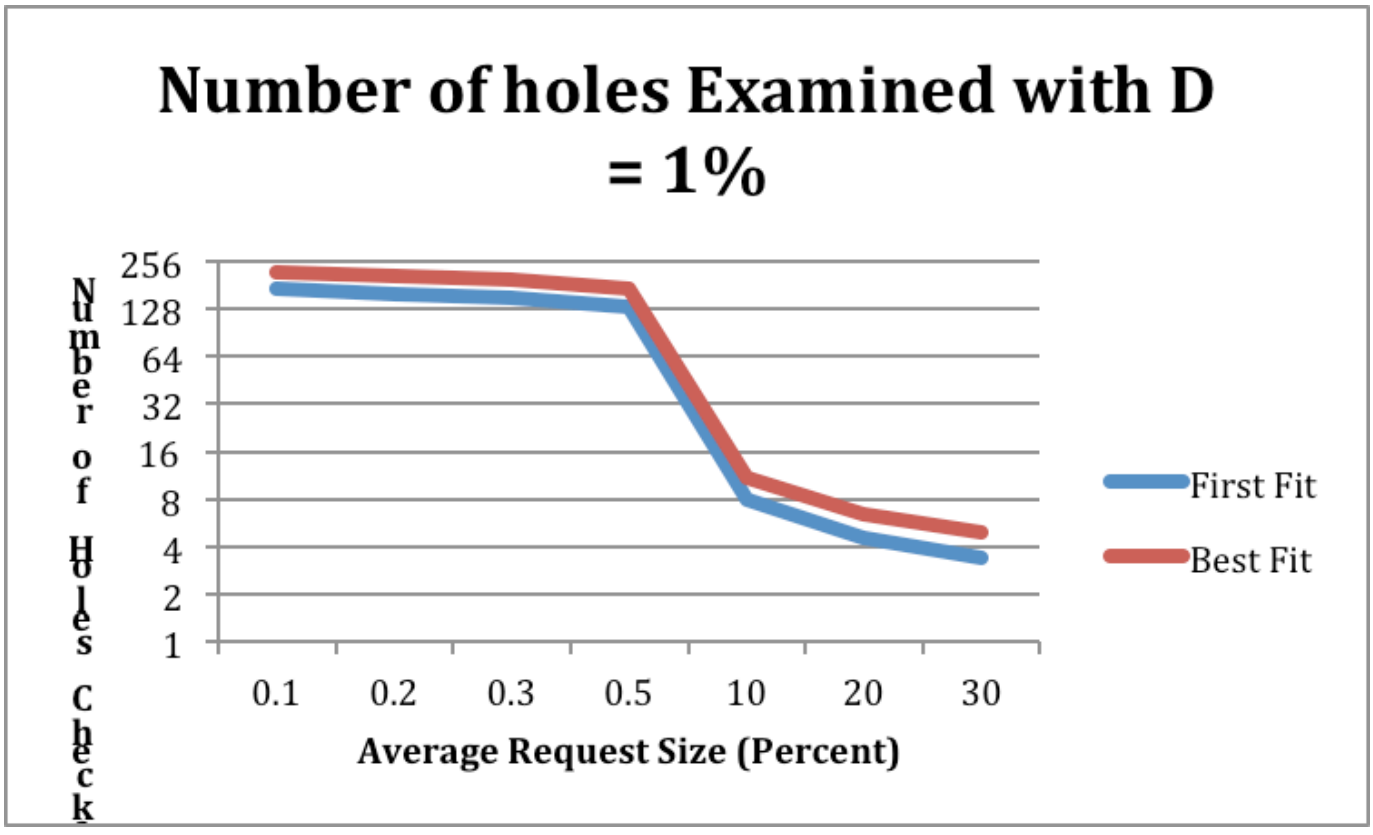
Results

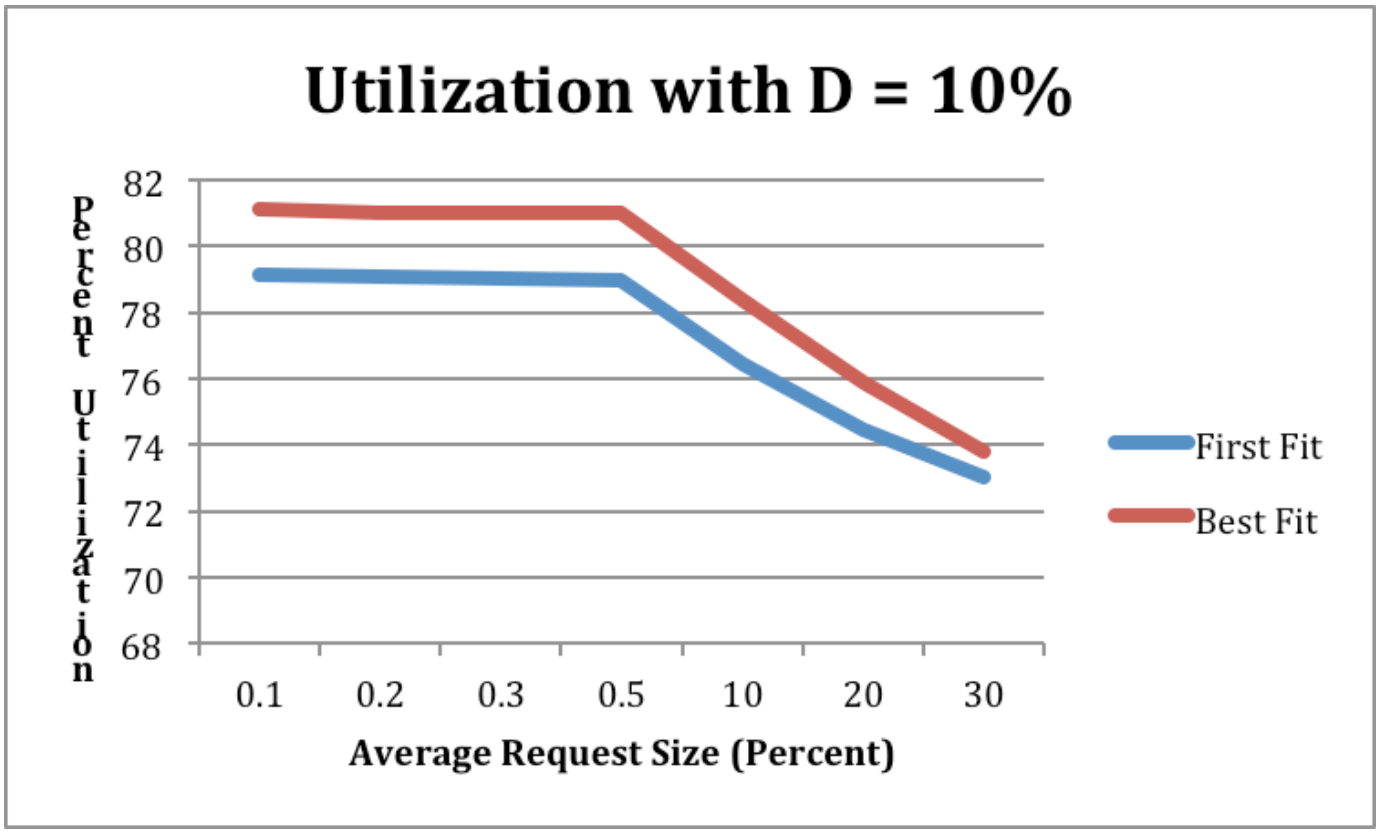
The following graphs show the result of simulation for different values of a and d, where a is the average requsest size, and d is the standard deviation of this request. Both d and a were run as a percentage of total memory.
The simulation was run with 100,000 steps and a 100,000 word memory size.

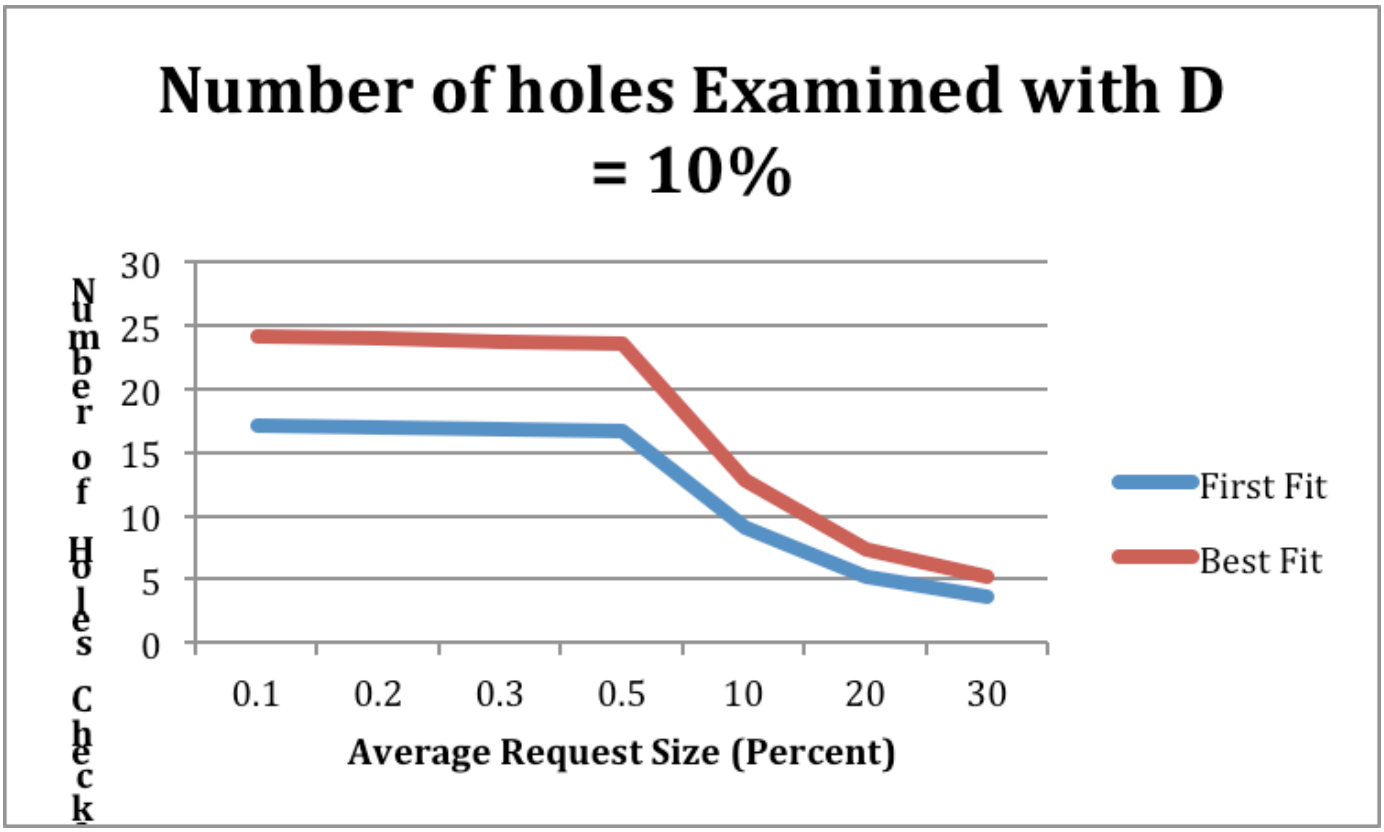
Average request size (a) and standard deviation (d) are a percentage of this total memory size (100,000 words).

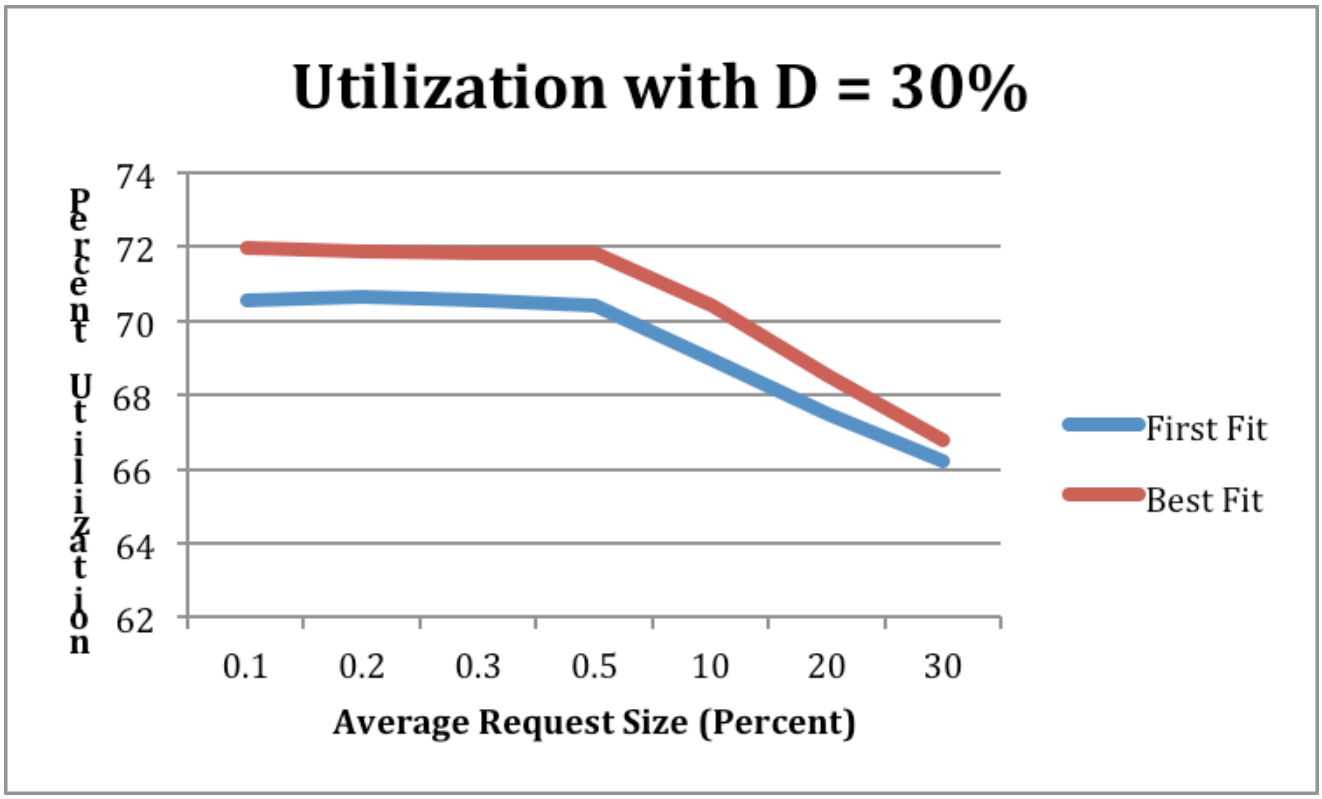
Each value of d was run with both algorithms (first fit and best fit). Two graphs are shown for each value of d, with the first being **the percentage utilization of memory** vs request size, and the second graph being **number of holes checked** vs request size.

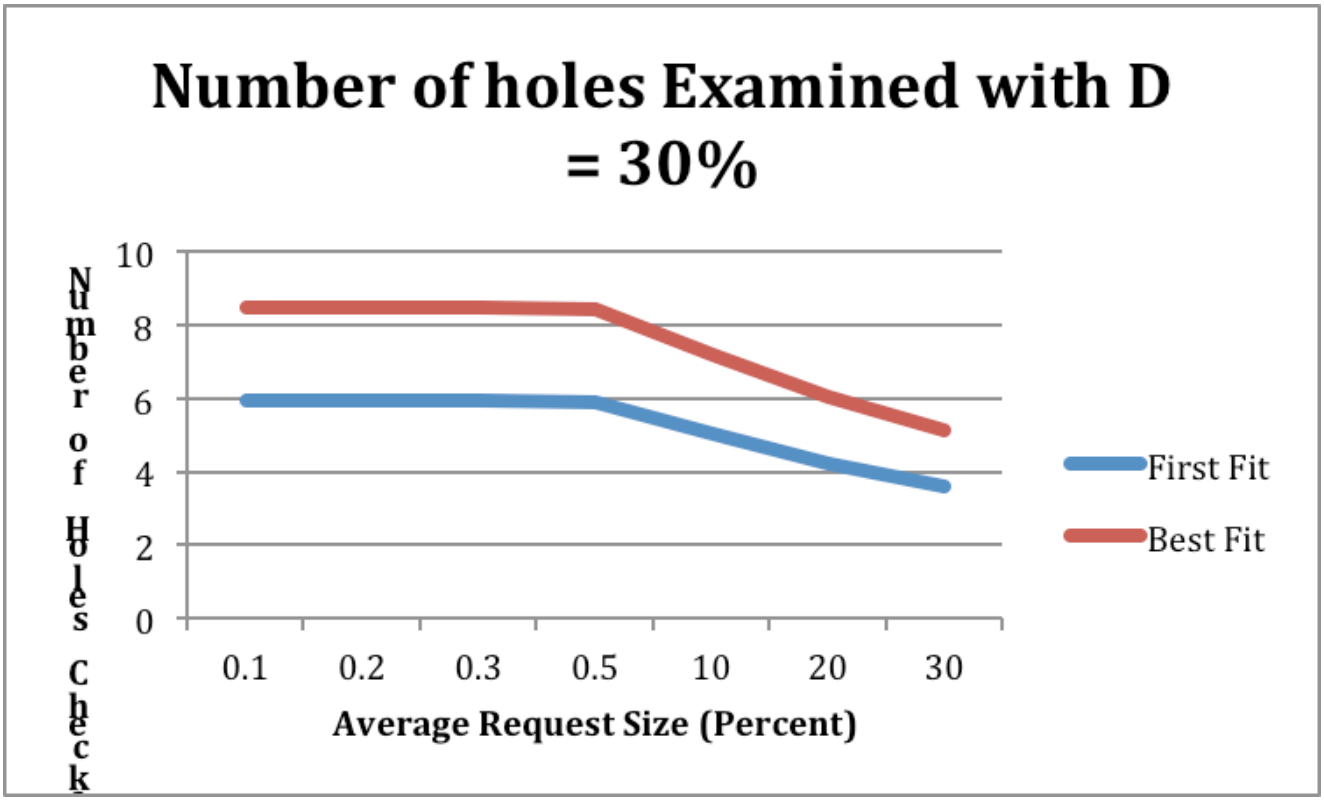

Utilization with D = 1%
Percent Utilization
95
90
85
80
75
70
0.1
0.2
0.3
0.5
10
20
30
Average Request Size (Percent)
First Fit
Best fit


Number of holes Examined with D = 1%
Number of Holes Check
256
128
64
32
16
8
4
2
1
0.1
0.2
0.3
0.5
10
20
30
Average Request Size (Percent)
First Fit
Best Fit


Utilization with D = 10%
Percent Utilization
82
80
78
76
74
72
70
68
0.1
0.2
0.3
0.5
10
20
30
Average Request Size (Percent)
First Fit
Best Fit


Number of holes Examined with D = 10%
Number of Holes Check
30
25
20
15
10
5
0
0.1
0.2
0.3
0.5
10
20
30
Average Request Size (Percent)
First Fit
Best Fit


Utilization with D = 30%
Percent Utilization
74
72
70
68
66
64
62
0.1
0.2
0.3
0.5
10
20
30
Average Request Size (Percent)
First Fit
Best Fit


Number of holes Examined with D = 30%
Number of Holes Check
10
8
6
4
2
0
0.1
0.2
0.3
0.5
10
20
30
Average Request Size (Percent)
First Fit
Best Fit

Conclusions

As you can see, best fit always maintained better utilization, and first fit always checked fewer holes.

For small request sizes, there is a large improvement of best fit over first fit with regards to utilization. This is most likely due to the improved fitting of holes with best fit. However, these small request size is always where we see the biggest gap in the number of holes examined for best fit vs first fit. This is due to the fact that there are more “allocations” to check, with small holes popping up everywhere, and best fit must check all these holes, while first fit only needs to check a few until it finds a suitable match.

As we decrease the number of requests, the utilization and number of holes checked decrease for both best fit and first fit. This is caused by the fact that there are less holes to fit because there are fewer allocations/deallocations, and its harder to fit large blocks rather than small ones, because there are less spaces to choose from.

We also see as d increases, there is a curious trend of utilization decreasing, as well as the number of holes checked. Most likely again this is caused by the fact that there are more “larger” requests that can’t be fulfilled(even though it’s randomized between large and small requests), and that brings down the overall average utilization and number of holes to check (because there are a few large allocation blocks, rather then a bunch of small ones).

Our prediction that an increase in d widens the gap in utilization between first fit and best fit was not necessarily true. The gap in utilization between best and first fit stayed roughly at 2% to 5%between all values of d. If anything, increasing d slightly decreased the gap between best fit and first fit. This may be due to the fact that “randomized” data simply causes both algorithms to become ineffective in allocations, effectively destroying the efficiency that first fit saw over best fit.

When does it matter if we use best fit or first fit? For large requests will little variations, taking a look at the 1st and 2nd graph in this report(D = 1%), we see that requests around 10% or more have roughly the same utilization and holes examined(search time). So if your application tends to use large amounts of memory with little variation, it would not matter which algorithm you use.

The area where we see the largest discrepancy between the algorithms are requests that have a medium sized distribution and are small. Looking at the 3rd and 4th graphs(D = 10%), we see the biggest gap in utilization and search time between best fit and first fit. Here, the requirements of your application would likely take precedence over which algorithm you use.

We also see a large gap in the search time of first fit vs best fit in the 2nd graph(D = 1%), where first fit searches roughly 170 holes and best fit searches 220 for small-sized requests with little variation. In this case, it would probably be better to go with best fit.

Overall, neither algorithm is inherently better than the other. You would use best fit if you had need for high memory utilization, and first fit if you needed fast search times.

Data

d	a	Average Memory Utilization	Average Search Time	Strategy
1	0.1	86.77349923	170.72821	0
1	0.2	86.74684764	159.53325	0
1	0.3	86.62994668	150.98025	0
1	0.5	86.46720669	132.22868	0
1	10	82.57273527	7.94193	0
1	20	80.86272555	4.58541	0
1	30	78.34239565	3.42663	0
1	0.1	90.30371263	220.97222	2
1	0.2	90.1575207	210.24343	2
1	0.3	89.99666014	198.37301	2
1	0.5	89.84410521	174.7189	2
1	10	83.46214204	11.13756	2
1	20	81.5475765	6.55465	2
1	30	78.1598818	4.94352	2
10	0.1	79.15386917	17.05971	0
10	0.2	79.08924521	16.92964	0
10	0.3	79.06556684	16.79197	0
10	0.5	79.01454313	16.63026	0
10	10	76.46389965	8.98493	0
10	20	74.4980348	5.15763	0
10	30	73.01134804	3.59956	0
10	0.1	81.09953862	24.07462	2
10	0.2	80.99824936	23.92748	2
10	0.3	80.97135169	23.69899	2
10	0.5	80.99287475	23.49543	2
10	10	78.36190731	12.74638	2
10	20	75.87953836	7.36593	2
10	30	73.80131658	5.18773	2
30	0.1	70.54038614	5.94555	0
30	0.2	70.648557	5.94996	0
30	0.3	70.5394483	5.93772	0
30	0.5	70.42088554	5.91185	0
30	10	68.94672997	5.0258	0
30	20	67.49304784	4.23116	0
30	30	66.23391199	3.6084	0
30	0.1	71.98492807	8.46317	2
30	0.2	71.91179008	8.46779	2
30	0.3	71.84640415	8.47473	2
30	0.5	71.84528907	8.43836	2
30	10	70.41267401	7.16743	2
30	20	68.55784097	6.01496	2
30	30	66.763574	5.14199	2