MARCH 8, 2016

OPERATING SYSTEMS MEMORY SIMULATION

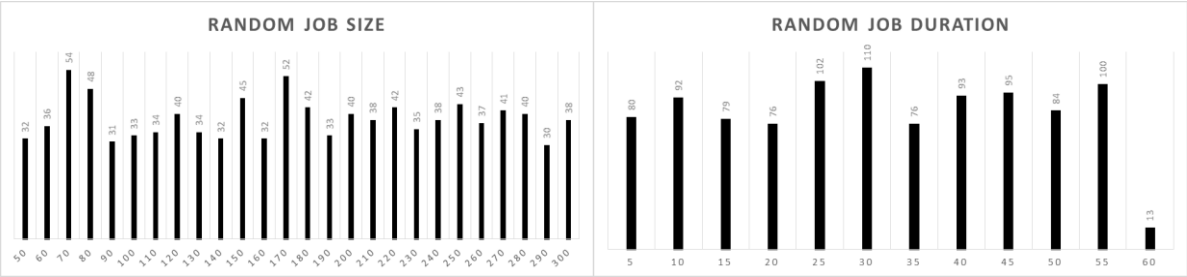
PROJECT – PHASE 1

NATHAN LEA
CS4323

Random Number Generator:

The random number generator I used in my project is the SecureRandom Object. This random number is a cryptographically strong random number generator which produces a nondeterministic output. This ensures that the jobs that are being produced by the simulation are truly random and cannot be guessed based on the last job.

Below is an output of the secure random number generator scaled to the correct mappings for job size and duration respectively each was over 1000 runs:



This graphs show that the random number generator is not symmetric and does not have a normalized distribution, which means that it is truly a random number generator.

Approach to the simulation

Simulation

There are several parts of the simulation that all work together to simulate a computer system. Each part is responsible for a different aspect of the simulation: initializer, memory manager, memory, ready queue, dispatcher, CPU.

Initializer

The initializer is called when a new job has entered the system. The initializer will call the random number generator to get the duration and size, assign the job the next PID, and produce the job header. Once the initializer has created a job it passes the header to the memory

manager to put it into memory; this job is assigned the name JobAtDoor because it has not yet been brought into the system.

Jobs

A job is represented by a block of memory that is equivalent to the size of that job. For easy of locating and gathering data about the job, each job had a job header that looked like this:

PID	Process Identifier
SIZE	Size of the Job
DURATION	Duration of the Job
VTU	Time job entered memory
PID	Process Identifier

Every job in memory will have this header, and it is the smallest size that a job is allowed to be so it will work for all jobs. If the job is larger than 50K then the rest of memory occupied by the job is filled with the PID of the job.

Job PIDs are implemented in a similar way to a basic UNIX system where as each job is added to the system the PID is increased by one. If a job is rejected by memory that PID is attempted to be used on the next job to allow for seamless counting of jobs in the system.

Memory Manager

The memory manager takes a job that has been created by the initializer and based on what algorithm the simulation is using, first, best, or worst fit, it will attempt to place the job in memory. If it is able to place the job in memory it adds the job PID to the ready queue that allows the job to be processed by the CPU. If it is unable to place the job in memory it looks if it will ever be able to find a hole large enough for the job. If it can eventually place the job, it will continue execution and after every job completion check and see if it is able to be placed in memory. If there is never going to be a spot in memory for it, then it is rejected and the job PID is added to the rejected job queue.

Memory

My approach to the simulation was to emulate memory in a computer as much as possible. The main memory of the simulation is an array of integer, each integer is a 10K block of memory. As a job is inserted in memory it will take a spot in memory that is equivalent to the size of the job. When a job is completed, the memory manager will go find the job in memory using a linear search algorithm and replace the integers with a negative of the job PID.

Ready Queue

The ready queue contains a list of active jobs in memory, these jobs are ready to be processed by the CPU. This list is FIFO to keep the jobs processing in order.

Dispatcher

The dispatcher is the block that is responsible for controlling jobs to the CPU, it pulls a job from the ready queue and gives it to the CPU to process

CPU

The CPU takes in a job and processes it for the duration of the job and then calls the memory manager to remove the job from memory. The CPU could possibly idle if there is not job in memory and the next job arrival time has not yet occurred. If the CPU does idle this the worst possible thing for the simulation as nothing is getting done in the system, this should always try and be avoided.

Overview

With all these pieces the simulation is allowed to run through a specified time. The simulation method used in this project is strictly event based meaning that the virtual time unit was only increased if an event has occurred increasing the time of the system, such as job completion and idle CPU until next job. This simulation technique makes the simulation much

more efficient because it is only looping through the simulation loop when some part of the simulation has completed and it needs to unload or load something into memory or the CPU.

Memory Placement Strategies

First Fit

First fit is a memory placement strategy that looks for the first hole that is big enough to place the job in and the places the job there.

Best Fit

Best fit is a memory placement strategy that looks for the best possible current hole, meaning the hole that would create the smallest hole and places the job there.

Worst Fit

Worst fit is very similar to best fit except that it finds the spot that would create the worst possible hole, or largest left over hole and places the job there.

Implementation First Fit vs Best Fit vs Worst Fit

The implementation of First Fit was very easy as you are just required to look through memory until there is place large enough for the job and place the job there. Very simple to implement and requires less overhead than the other to implement as there is no variables needed to store the best location in memory. Overall, this was by far the easy memory placement strategy to implement.

Next, Best Fit and Worst Fit these two memory placement strategies are very similar in one is looking for the best hole and one is looking for the complete opposite. These were almost identical to implement, just a change of a sign. They were not significantly harder to implement than first fit but they were more difficult, in that you had to search the entire memory looking at

every memory block to find the best/worst place to put the job.

Performance First Fit vs Best Fit vs Worst Fit

In comparing performance many things have to be considered such as jobs completed, jobs rejected, average hole, fragmentation, and storage utilization. After much investigation into the memory strategies the only difference between them is the set up case for the memory, and after that it will almost always place the jobs in the first hole big enough, which is exactly what first fit is doing. And considering all cases of the random number generator, you could get the exact same result since the testing is not done on the exact same data set. Assuming that doesn't happen, there is still a significant performance difference between them.

Completed Jobs – Chart A

The completed jobs are very similar up until around 4000 VTUs and then worst fit drops much lower, at this point the fragmentation cannot keep up with completing jobs and falls off. Looking at best fit and first fit, these are very similar, even though best fit does complete more jobs there is not a statistical difference in completed jobs.

Rejected Jobs – Chart B

A difference is more apparent in the rejected jobs, worst fit has many more rejected jobs than best or first fit. In worst fit, memory was getting chopped into many small pieces and all the large jobs were being rejected. Best fit and first fit are still very similar, best fit does reject fewer jobs but not by much. This is very similar to the difference in completed jobs where it is better but not by enough to call it always better or even statically different.

Average Hole Size – Chart C

The graph is very interesting as first, best, and worst fit all seem to converge to the same average hole size of around 40K. In this metric the methods seems to be equivalent to each other

and no difference can be measured here.

Fragmentation – Chart D

The fragmentation over time shows a very interesting difference. The trend between the strategies are very similar up until 2500 VTU and then something that happens in memory that begins to fragment memory to an extreme amount and much of memory gets lost to fragmentation. This affects best fit the most, followed by first fit, and most interesting worst fit was affected the least. By this chart, the best memory method is worst fit and the worst one is best fit very different than the rest of the charts. This difference is most likely that best fit has completely switched to first fit and memory is getting completely fragmented. At this point in the rejected jobs the number for best fit also starts to increases. The slope of the rejected jobs at this VTU point in rejected jobs also changes from a slow increase to a very sharp almost exponential line showing that memory has because very fragmented and can only fit small jobs chopping memory into lots of little pieces.

Storage Utilization – Chart E

The final metric for comparing the methods is storage utilization, this is the measure of used memory (occupied memory) vs total memory (180KB). The metric is the best for observing how useful memory is for storing new jobs. Worst fit is the clear looser as toward the end of the time the jobs in memory is zero, the system can't store any jobs and the CPU is idle waiting for a job that it can process. Best fit and first fit follow much of the same trend between 20-40% utilized. This compared to the fragmentation shows that even that best fit had the most fragmentation, there are still many large holes and many jobs are allowed to fit in memory unlike with worst fit where most of memory is filled with small holes, not small enough to be unusable but just small enough where only a few job sizes can fit in the holes.

Selection First Fit vs Best Fit vs Worst Fit

After looking at all the metrics that were measured along with comparing the implementation difficulty and the run time. The best form of memory placement strategy has to be first fit, because it completes nearly the same number of jobs as best fit and is it a much easier strategy to implement. In all respects, best fit only beats first fit marginally and with the extra time and difficulty that is added with the best fit, first fit is a much better option for managing memory in this simulation.

Improvements Best Best Fit and Worst Worst Fit

After much thought about how to improve the memory placement in the simulation, I decided to look at what would happen if the memory manager would only place a job in the best ever going to be hole and the same for worst. This is what I coined to be best best fit and worst worst fit. This takes out the element in which after a while best and worst fit turned into a modified first fit which I felt would hinder the results of what a true best/worst fit would do. I made these same metrics comparing the previous first/best/worst fit to best best fit and worst worst fit.

Completed Jobs – Chart F

Best best provides a minimal addition to the number of jobs that the system was able to complete. Worst worst provided a significant loss in job completion. This is very interesting that first fit actually more times than not placing the job in a pretty optimal spot rather than a bad spot like worst worst is designed to do. First fit is looking like a much better after this metric.

Rejected Jobs – Chart G

This metric provides insight into why best best is better than the normal best fit.

However, best best fit may not have been able to complete very many more jobs than best fit but best best was able to keep way more jobs than any of the previous methods, it was able to find a spot for the majority of the jobs that arrived in the system. Worst worst fit has a very linear rejected jobs trend that demonstrates what always picking the worst possible hole does to the fragmentation of memory.

Average Hole Size – Chart H

This metric is very similar to before, the average hole size again converges to around 40K even for the best possible placement. No memory placement method is can avoid this convergence of the average hole size showing that fragmentation is unavoidable, no matter how good the placement strategy.

Fragmentation – Chart I

Best best provides minimal improvement in this area, similarly worst worst is minimally worst that worst fit. Best best does improve the fragmentation but it is approaching the best that it can do when memory is lost so easily.

Storage Utilization – Chart J

Storage Utilization provides a very good insight into exactly how good the method are, much like it did for the original methods. Best best follow much of the same trend as best and first fit, not really providing a significant improvement over the others. Worst worst fit is worse than worst fit but not by much. Both are very similar to their original counterpart and not much is improved with the extra addition.

Improvement Overview

Looking at the benefits of best best fit, while it does provide significant improvements to the rejected job count but other than that it does not really have a benefit that is desirable over

best fit or first fit. Best best fit, adds some pretty undesirable metrics that were not measured in this simulation but should be considered before implementation. Best best fit adds a large amount of wait time but not the wait time in the system, the time that a job is waiting to be inserted into memory. In every loop through the simulation the memory manager would wait until almost everything was completed in memory and then insert in the optimal spot in memory. This causes the simulation to not even see many of the jobs that first and best fit would have the chance to see but ultimately reject. Best best fit would not work very well if there was any level of multiprogramming in the system as the CPU would pretty much only have one job to work on.

Worst worst fit, is just the worst of everything but was an interesting exercise to see how close the normal worst fit is at always picking the worst spot and creating the most fragmentation, and it is pretty close on their levels of fragmentation.

Selection First Fit vs Best Fit vs Worst Fit vs Best Best Fit vs Worst Worst Fit

After the addition of what seems like a method that would complete many more jobs in the system, the selection of which that I would implement in my system would is still first fit because of its ease of implementation, and the minimal stalling that it would add to the system. First fit can also add a possibility of multiprogramming that best best could not have. Overall, this simulation has proved that while not always the best, first fit is still the strongest memory placement strategy for this type of memory simulation.

Appendix

Chart A

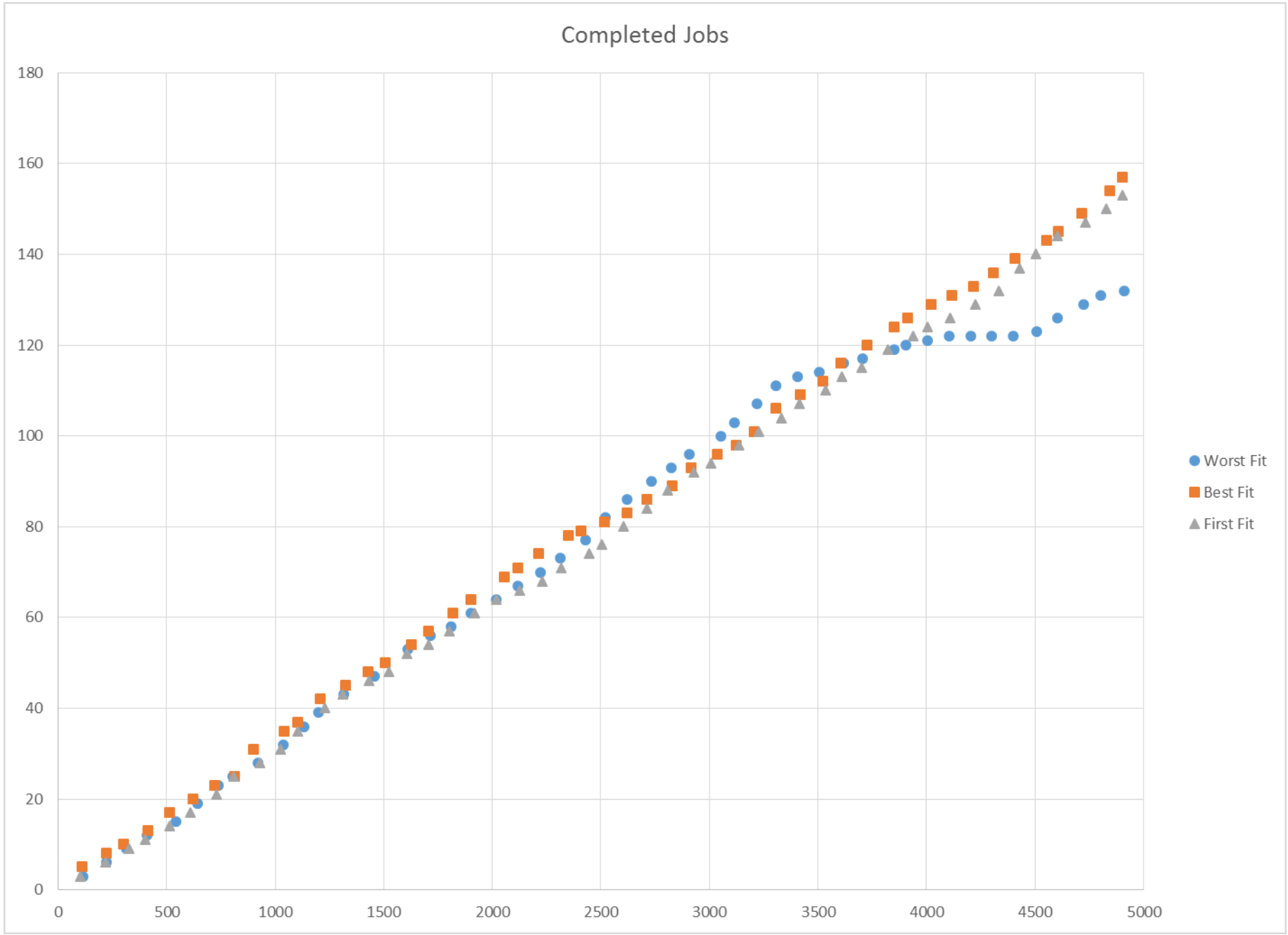


Chart B

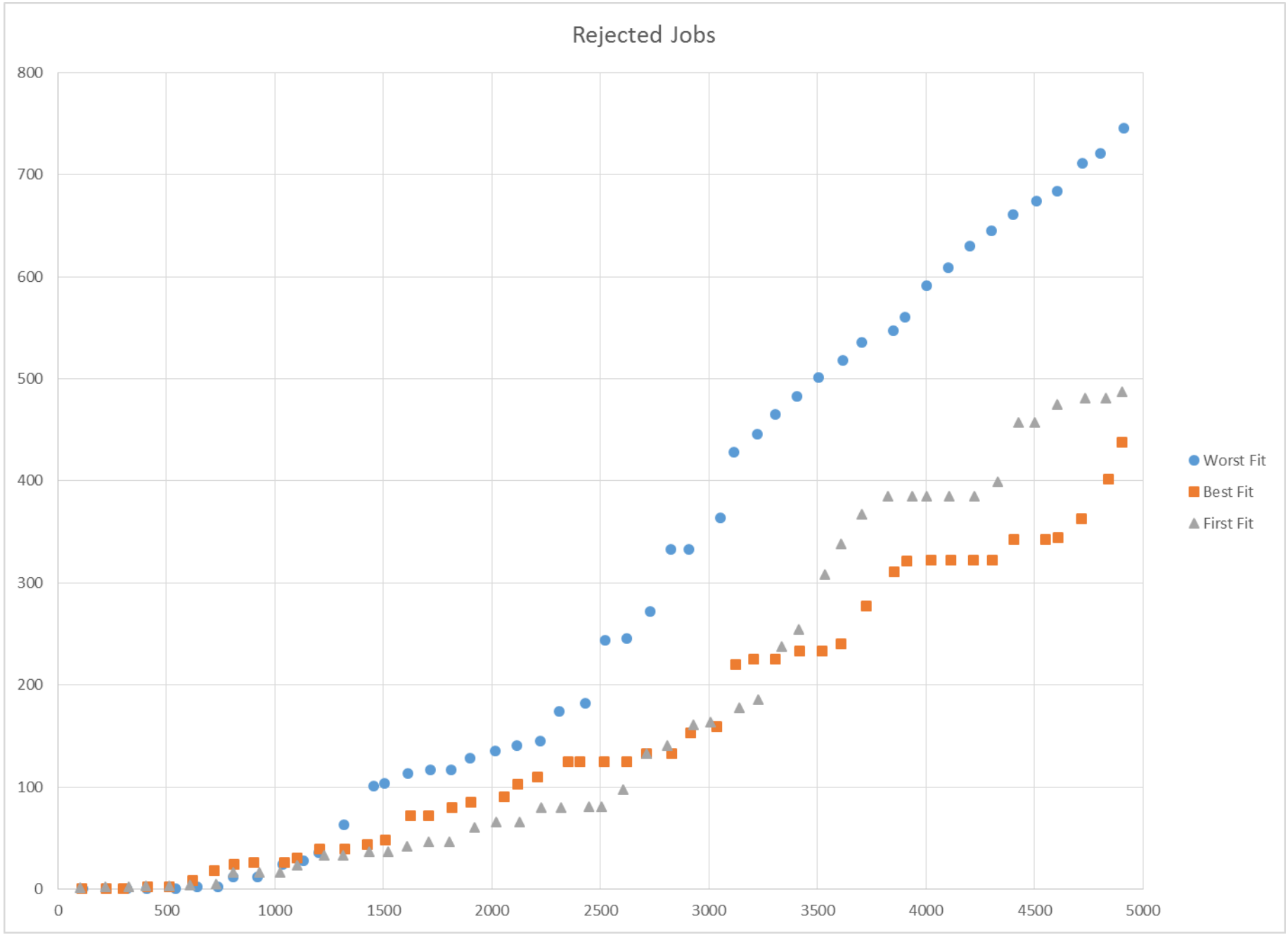

Rejected Jobs
800
700
600
500
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit

Chart C

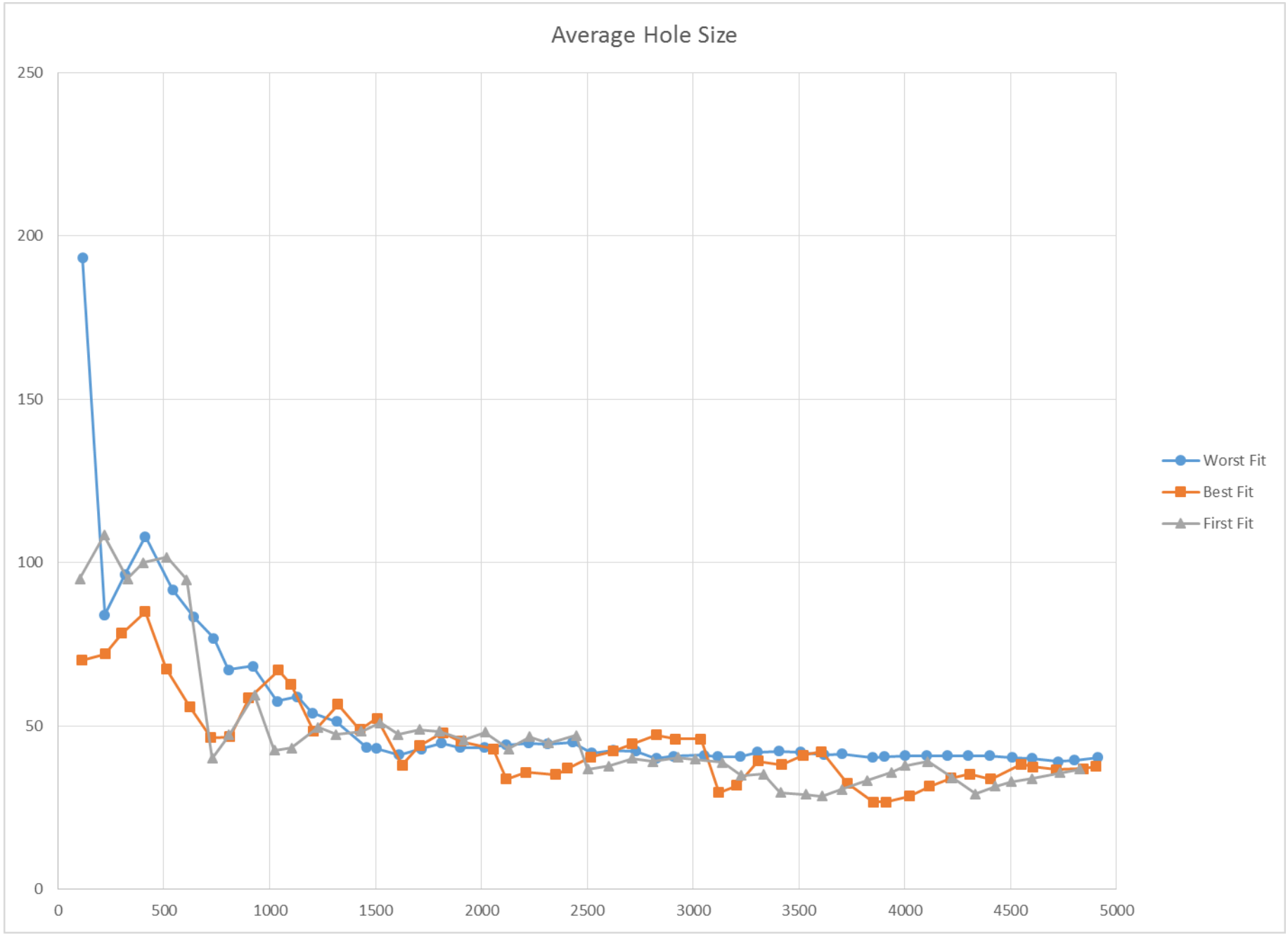

Average Hole Size
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit

Chart D

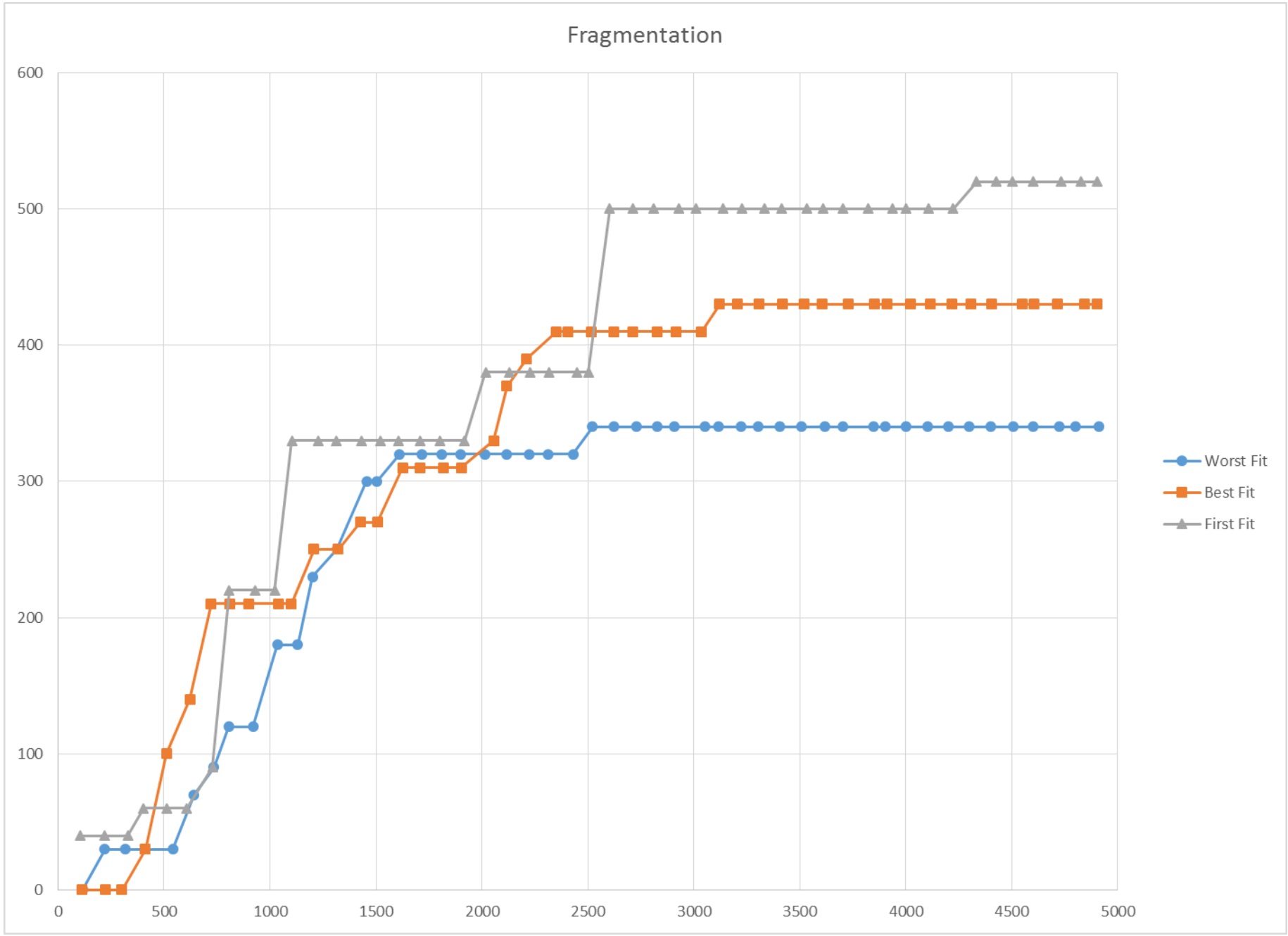

Fragmentation
600
500
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit

Chart E

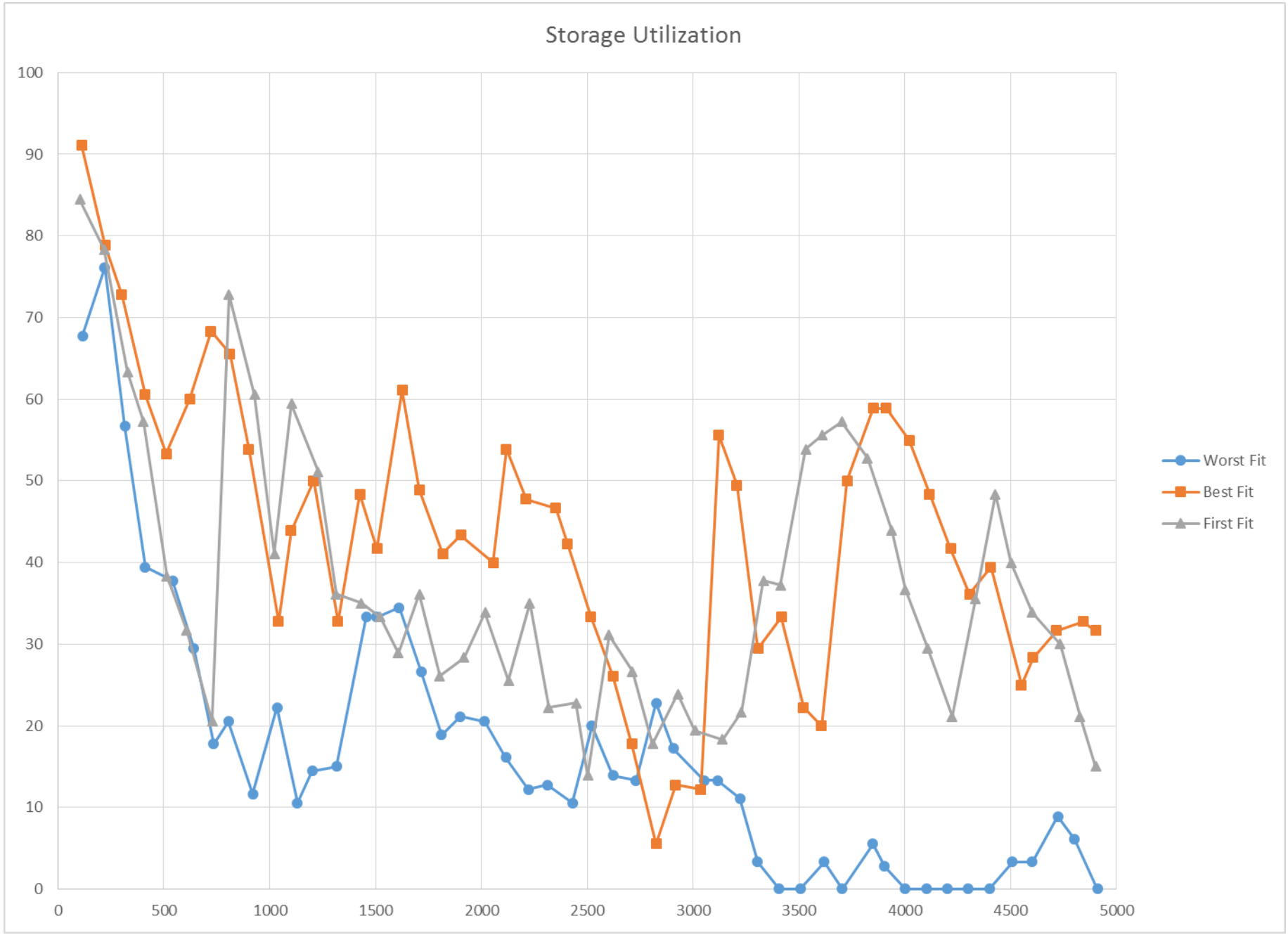

Storage Utilization
100
90
80
70
60
50
40
30
20
10
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit

Chart F

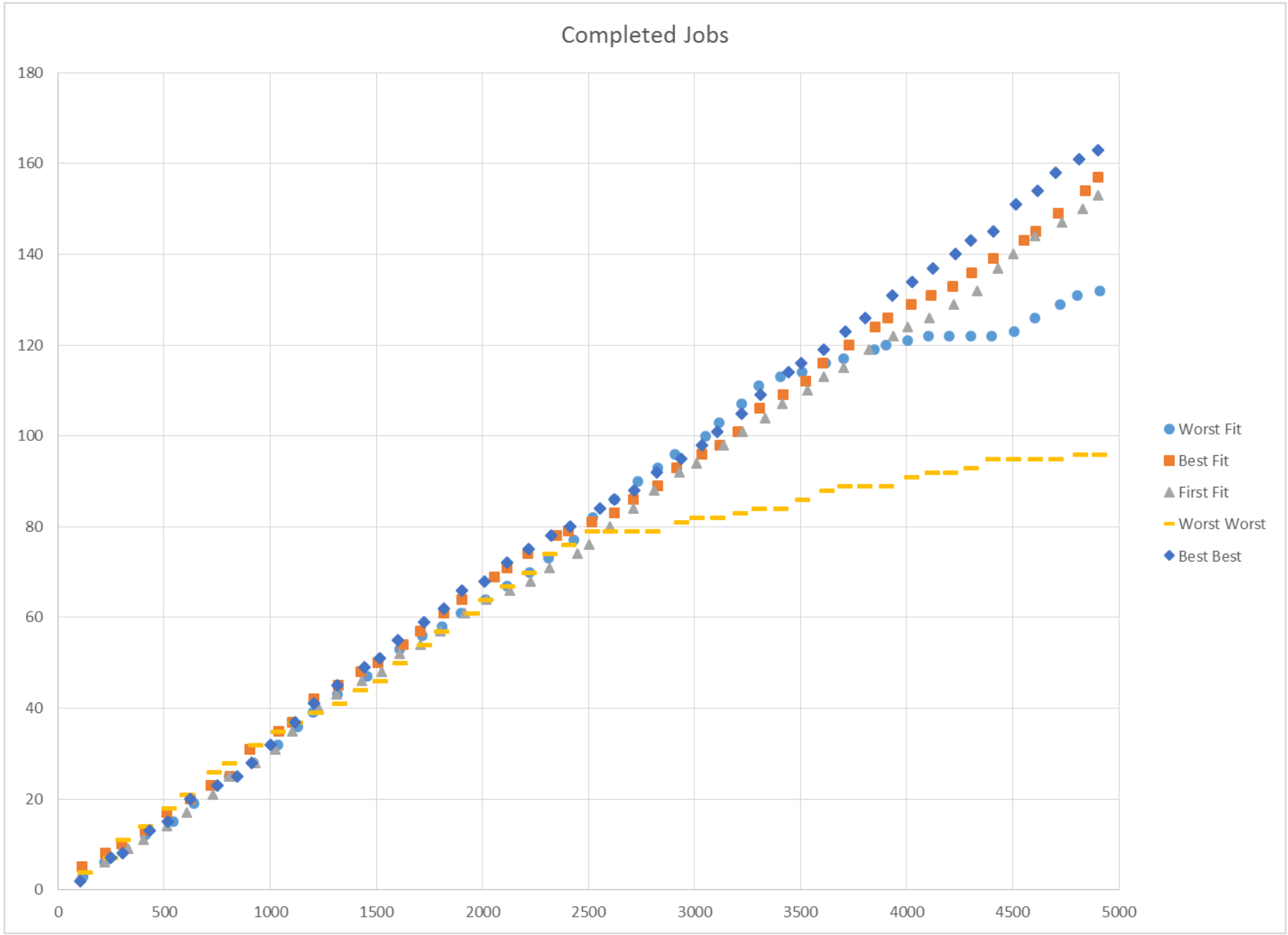

Completed Jobs
180
160
140
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit
Worst Worst
Best Best

Chart G

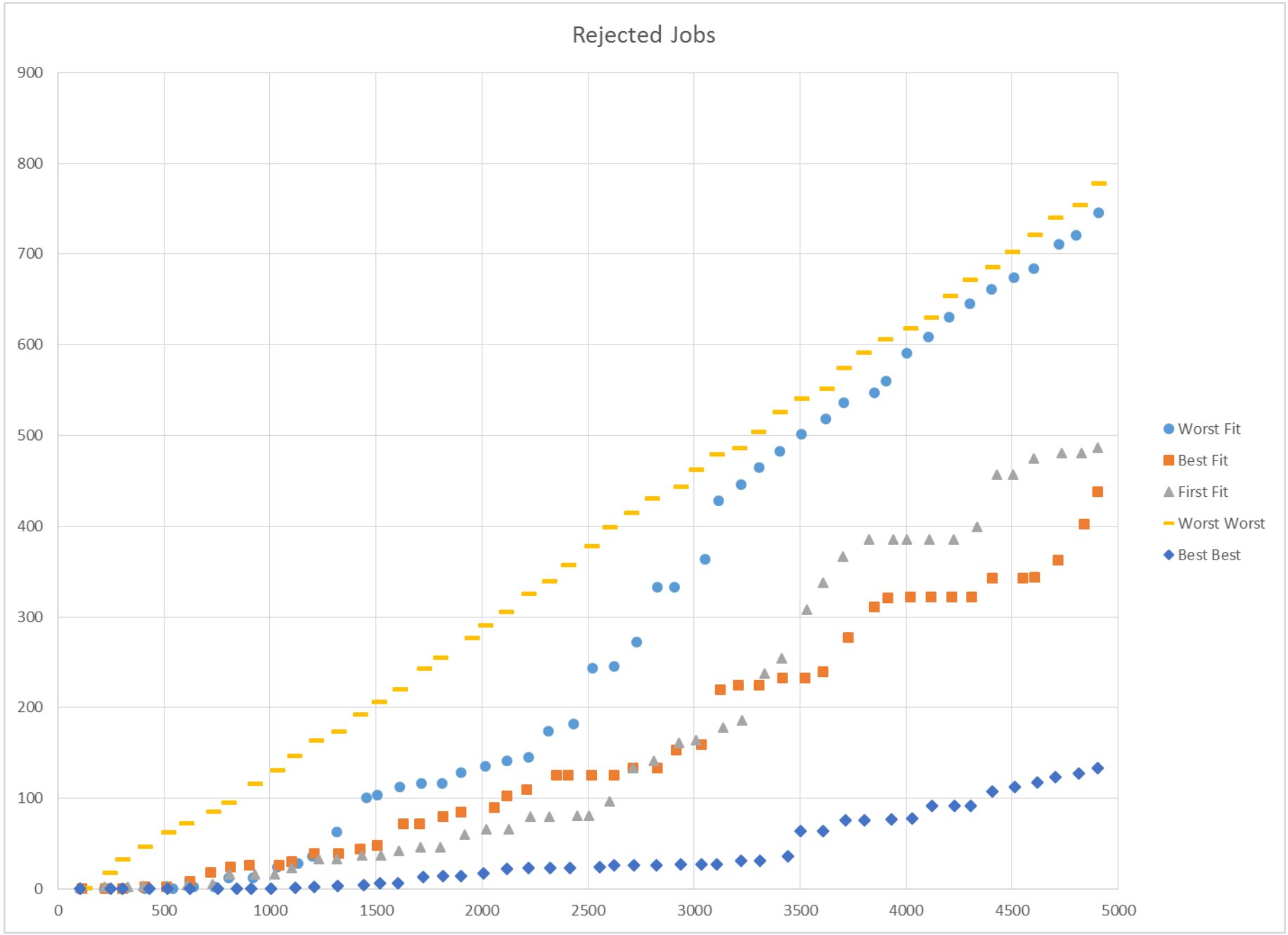

Rejected Jobs
900
800
700
600
500
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit
Worst Worst
Best Best

Chart H

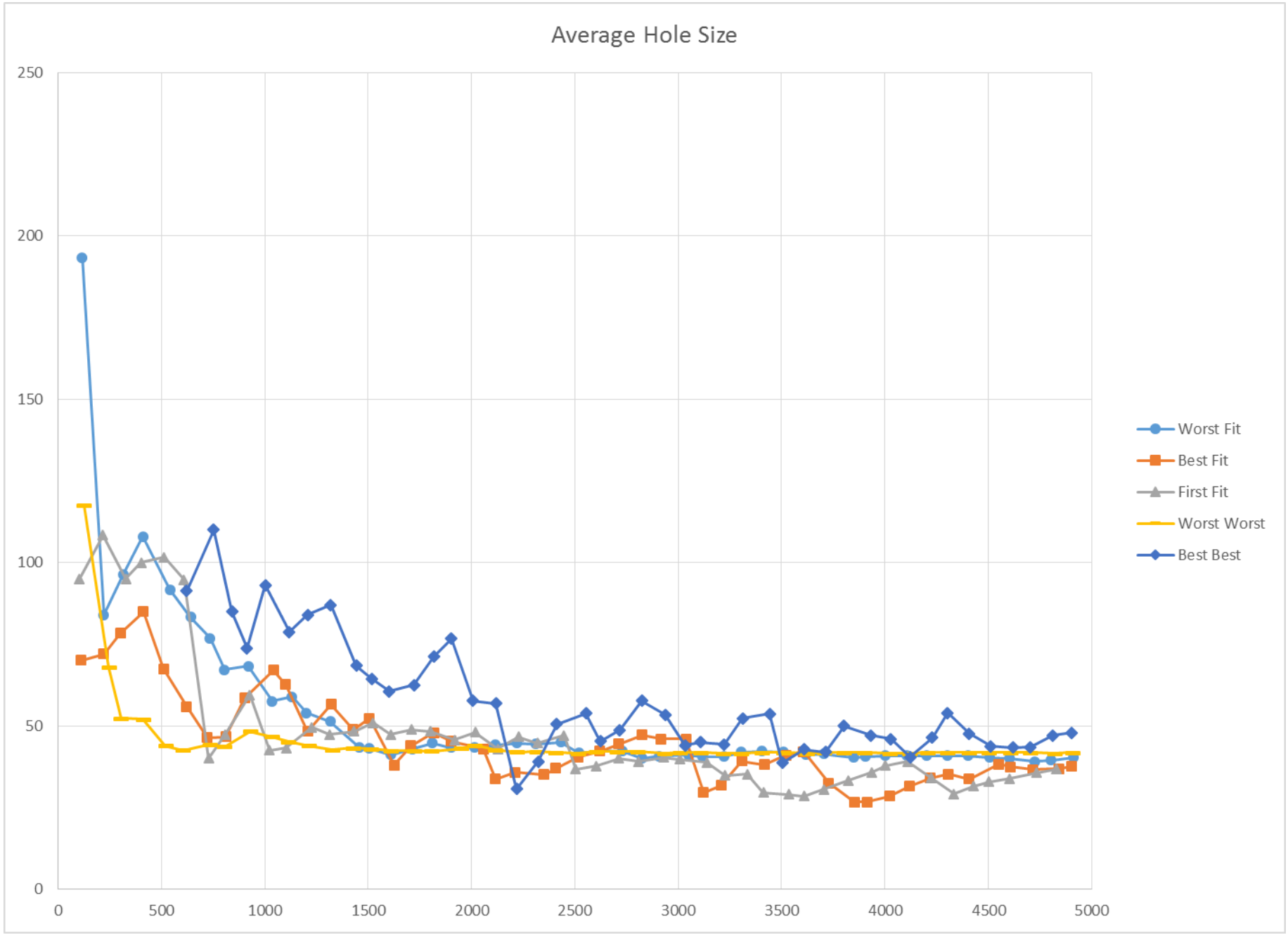

Average Hole Size
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit
Worst Worst
Best Best

Chart I

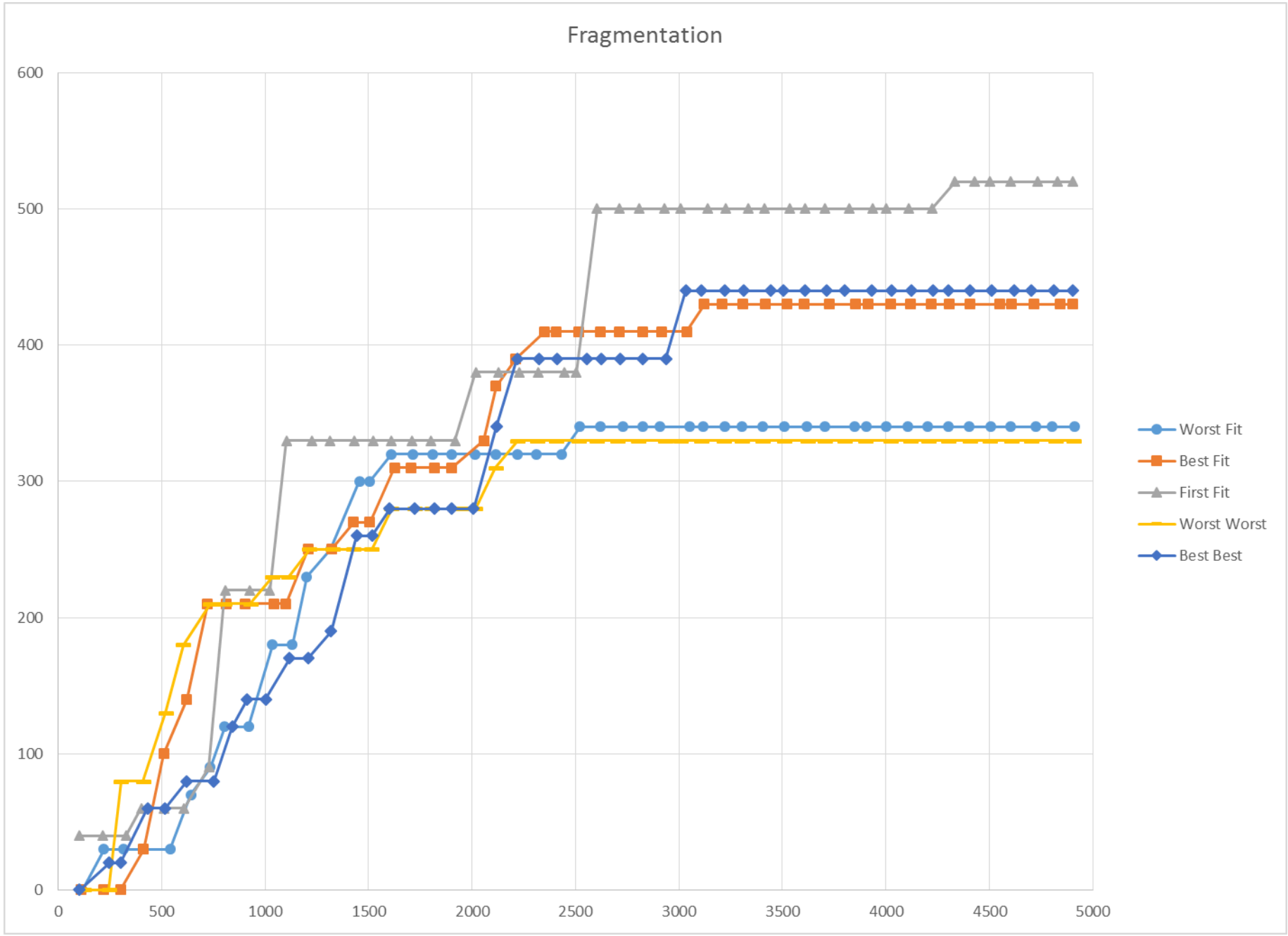
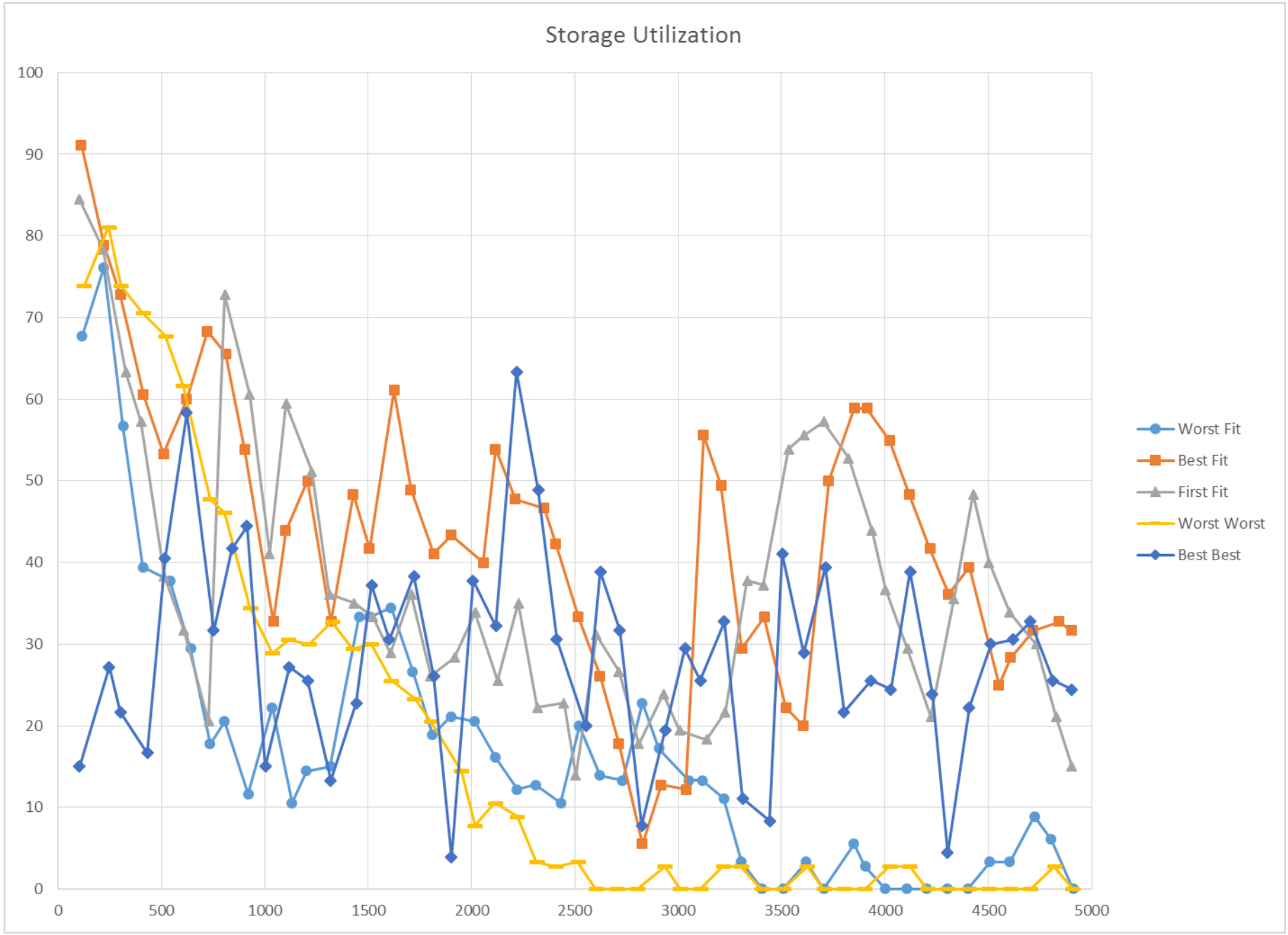

Fragmentation
600
500
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit
Worst Worst
Best Best

Chart J


Storage Utilization
100
90
80
70
60
50
40
30
20
10
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Worst Fit
Best Fit
First Fit
Worst Worst
Best Best