Christopher
Erattuparambil
ce7347

Gale Shapley Report

By contradiction we can prove a stable matching always exists. Suppose there exists an instability. One possibility is that u is assigned to s and u' is assigned to no server and s prefers u' over u. Then u' and s will be a new pair and u will not have a pair. The second type of instability is when s is paired with u but prefers u' and u' is paired with s' but prefers s. Then a switch will be made with the new pairing of s and u' and s' and u one pair gets their preference and even though s' and u don't the matching is still stable. The ability to leave unwanted users by the servers will always give you a stable, server-optimal matching.

Initially all the server slots are free and no jobs have been assigned

while (the number of slots used is less than the total)

for each server

while there are open slots

{ check if top-user preference is matched)

↳ if not match it to the server and add to ma

↳ if user is matched

(continue on next page)

(cont)
↳ if user is matched

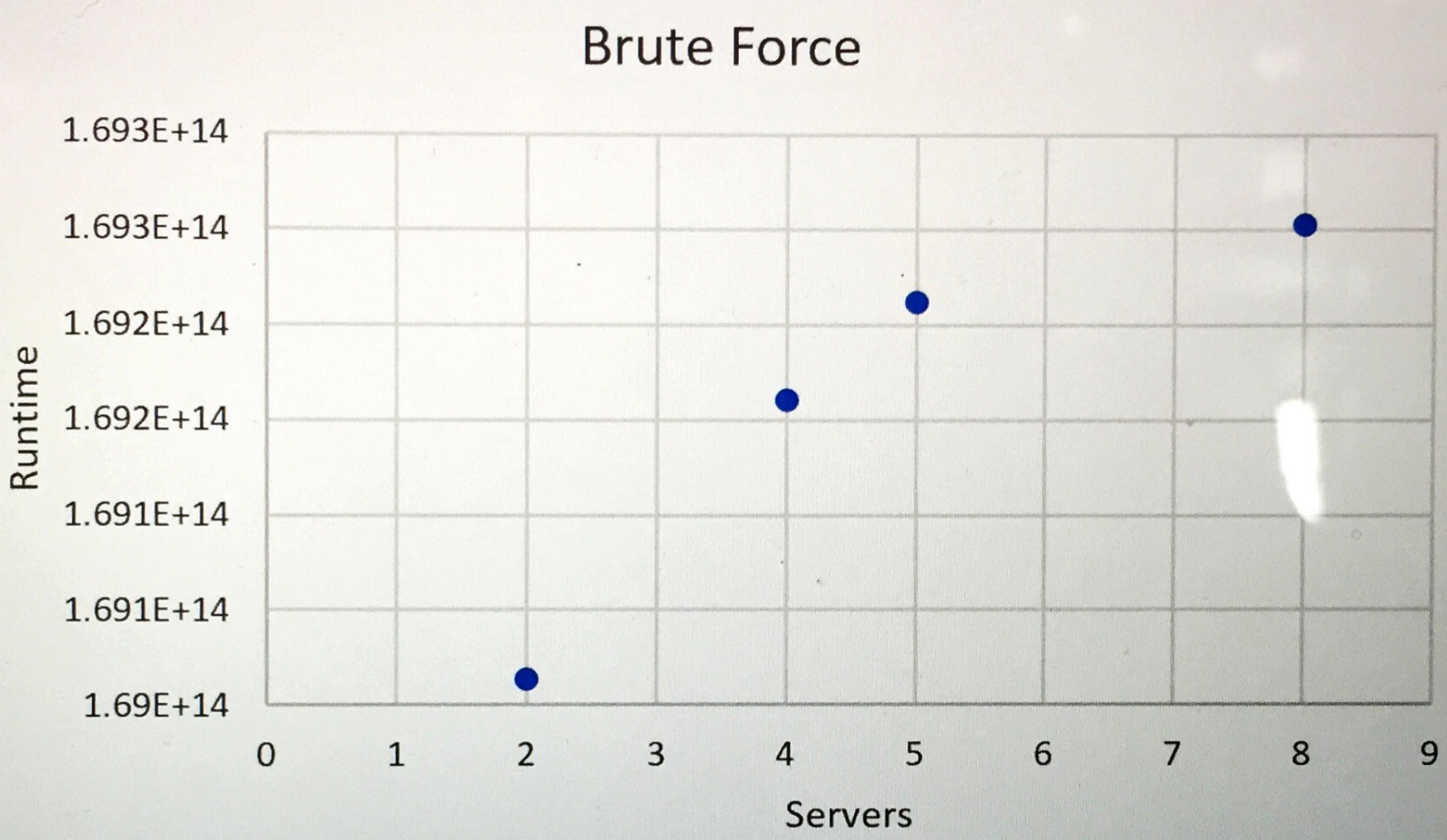
Check if user prefers current server over asking server
↳ if current server is preferred
↳ asking server asks next user in preference list
↳ if asking server is preferred
make asking server and user a pair
and add it to matching ArrayList
and change user matching value to -1.

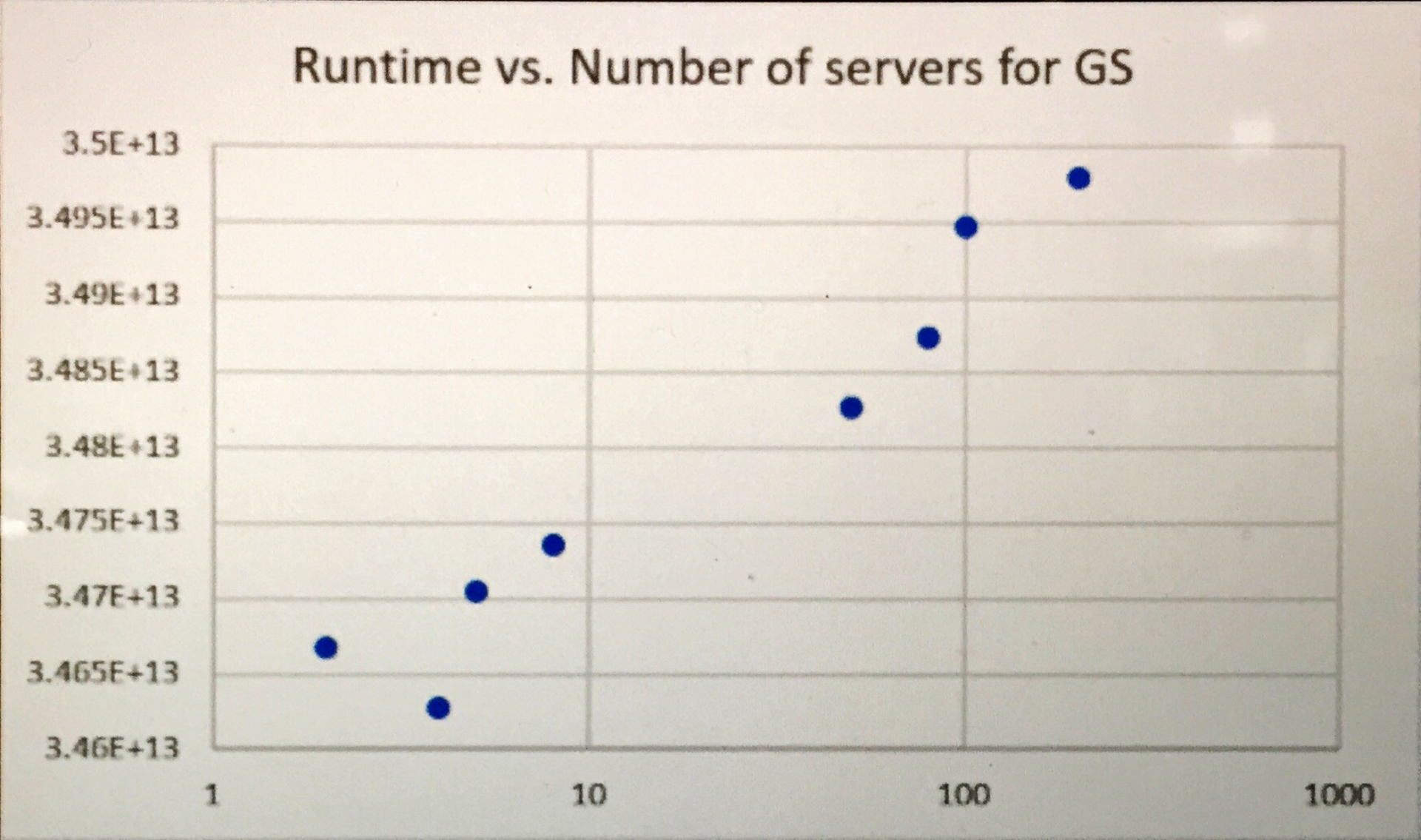
The complexity is $m \cdot n$ where m is the number of server slots and n is the number of users. This is because each slot will propose to every user as a worst case.

After every server slot has proposed to n user there have been $m \cdot n$ proposals. At this point every server has propseed to every user and there are either the same number or more jobs than the open servers. By $m \cdot n$ proposals all the server slots will be full and my code will terminate. To prove a stable correct matching will be output we will assume that there is an instability. That means there is a user who is unmatched and prefers a server. Since this is server optimal only the the server prefers the user than the matching would exist, but since the server is already paired with a user, that means it does not prefer the user that prefers the server, so a contradiction

arises if the server did like the unmatched user, because then it should've asked it earlier to pair compared to the user that it is already with

The runtime complexity of the Brute force Algorithm is $O(n! n^2)$. The $n!$ comes from all the possible user matching combinations. The n^2 part comes because for each user matching a nested for loop checks each set of (user, server) pairs in the matching. Therefore the complexity is $O(n! n^2)$.


Brute Force
1.693E+14
1.693E+14
1.692E+14
1.692E+14
1.691E+14
1.691E+14
1.69E+14
Runtime
0
1
2
3
4
5
6
7
8
9
Servers


Runtime vs. Number of servers for GS
3.5E+13
3.495E+13
3.49E+13
3.485E+13
3.48E+13
3.475E+13
3.47E+13
3.465E+13
3.46E+13
1
10
100
1000