iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** VII **Month of publication:** July 2023

DOI: https://doi.org/10.22214/ijraset.2023.54844

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VII Jul 2023- Available at www.ijraset.com

Automating Branch Change: Developing an Efficient Algorithm for Solving the Branch Change Problem

Jemunigani Jishnu[1], Juttu Gajendra Anurag[2], Eturu Harshith[3]
[1, 2]*Department of Computer Science and Engineering, Indian Institute of Technology, Palakkad*
[3]*Department of Computer Science and Engineering, Vellore Institute of Technology, AP*

Abstract: ***The B.Tech program's Branch Change Problem is a key component, giving students the chance to transfer to their desired branch after the first year based on their performance. Since the academic division now solves this issue by hand, there is a possibility of inaccuracy, and it takes a lot of time and effort. In order to overcome these difficulties, the goal of this research is to create a computer algorithm that can successfully solve the branch change problem. The program will minimize the time and effort needed for resolution by automating the process, minimizing the chances of human error. The suggested method would improve branch allocation's efficiency and accuracy overall, guaranteeing a fair and efficient process for students looking to switch branches.***

I. INTRODUCTION

After the first year, every B.Tech student has the option to request a branch change. If the transfer does not contravene the guidelines established by the Institute Senate, the candidate may receive their preferred branch based on how well they performed in the first year. The Branch Change Problem is what we refer to it as. The academic division of our Institute has been manually resolving this issue ever since it was founded. When done manually, there is a high risk of mistakes and a high cost of time. Therefore, the goal of this project is to create a computer algorithm that can address the branch change issue.

A. Organization of The Report

A general overview of the issue and the need for a solution is provided in this chapter. The remaining chapters are set up as follows: In the following chapter, we will understand the problem in greater depth. In Chapter 3, we discuss the allocation verifier approach we have worked on. In Chapter 4, we go through the research papers we have studied and see how they will help us solve our problem. In Chapter 5, we discuss the algorithm designed using the ESDA approach, which we have adapted from the research papers studied, and we see the implementation of this algorithm and finally, in Chapter 6, we conclude with some future work.

II. UNDERSTANDING THE PROBLEM

A. Rules of Branch Change

The Senate of the Indian Institute of Technology and the Vellore Institute of Technology has established the following rules that must be followed in order for a change of branch to be permitted:

1) Depending on the number of openings, all students who successfully completed the First and Second semesters of the course will be eligible for consideration for change of branch.
2) The first year's performance will serve as the foundation for any branch change considerations. In case of a tie in CGPA, the Entrance rank is used.
3) When changing branches, a class's strength cannot drop below the current strength by more than 10% of the sanctioned strength or rise above the sanctioned strength by more than 10% of the sanctioned strength. Both instances of "strength" in this context refer to the total number of students in the class.
4) Regardless of Rule 3, a minimum of one student from each discipline will be qualified to be considered for a change of branch at the end of the first year.
5) If a student with a higher GPA is denied access to a certain branch due to

Other restrictions aside, this should not be made available to any other students with a lower GPA, even if they meet the requirements based on the standards in place.

©IJRASET: All Rights are Reserved

B. Understanding the Rules of Branch Change

1) Sanctioned Strength

The sanctioned strength is the number of seats allowed to a specific branch. The Institute which we have taken for working now has four branches, each with its own set of sanctioned strengths: Computer Science and Engineering (CSE) - 50,
Electrical Engineering (EE) - 50,
Mechanical Engineering (ME) - 40,
Civil Engineering (CE) - 40

2) Existing Strength

Existing Strength refers to the number of students who have joined a particular branch at the start of the first semester. Existing strength may differ from sanctioned strength since some seats may be unoccupied.

a) Rule 3 says that a class's strength should not drop below its existing strength by more than 10% of the sanctioned strength. Let's assume the existing strength of CSE is 47. As sanctioned strength as mentioned above is 50, 10% of 50 = 5. This means that the existing strength may only drop by 5 i.e, 42. This is referred to as being "Floored out" or "Bottomed out."

b) Rule 3 says that a class's strength should not exceed the sanctioned strength by more than 10%. Let's assume the sanctioned strength for CSE is 50. 110% of 50 equals 55. As a result, the strength can reach a maximum of 55. This is referred to as being "Roofed out" or "Maxed out".

c) Rule 4 is a sub-case of Rule To deal with situations like this, the term irrespective of rule 3 is used. For example, if a branch's sanctioned strength is just 8 (any natural number less than 10 may be used as an example) then that branch's existing strength will definitely be 8 or below, now 10% of the sanctioned strength is less than one, So branch can't lose a student as the strength of a class will fall below the existing strength by more than ten percent of the sanctioned strength. As a result, Rule 4 assures that only one student can leave that branch under such circumstances.

3) Origin of Rule-5

The origin of Rule - 5 would have most likely been to guarantee that this reallocation was done fairly. Though this regulation appears to be fair for a student with a higher GPA, it may also be detrimental to a student with a higher GPA.

4) Supremum Criteria for Allocation

The student with the highest GPA is prioritized under this criterion. Assume there are two allocations, A_1 and A_2, that are sorted by GPAs in decreasing order. If student S gets his better preference in A_1 over A_2, we say A_1 is optimal/better. Where S is the first student from the top whose allocation has been changed.

III. ALLOCATION VERIFIER APPROACH

After reviewing the prior works we began to see the problem in a new way. We wanted to verify if a given allocation is optimal or not. If the allocation is not optimal, we'll need to figure out why (The thought behind this is if we figure out why our allocation isn't optimal and then figure out how to solve it). By solving it we can get into a better/optimal allocation. Now, this problem can be seen as an Allocation Verifier.

A. Problem Statement

We will be given a particular allocation and we need to verify whether it is the optimal allocation (following the supremum criteria) for the given branch change requests.

1) We assume the given allocation is not optimal and we try to check whether our assumption is right or wrong .
2) In the given allocation the students will be in the order of decreasing GPA.
3) We will take the first student who did not get his first preference and we try to find a way that will improve his preference.
4) If we find a way to improve the preference for this student we can say that the given allocation is not optimal. If not we repeat this procedure for all the students below him.
5) If none of the students got into their better preferences then we can conclude that the given allocation is optimal.

There are just three reasons for a student for not getting the better preferences in the given allocation.

- The current branch of the student is floored out
- The preferred branch of the student is roofed out
- Or the allocation does not obey rule-5.

But for the first person who misses out on his first preference, let say S_1, only the first 2 reasons (from above mentioned) will be applicable. If he misses out on any of the preferences then those particular branches cannot be allocated to any of the students below him by rule-5(As he is the person(S_1) with the highest CGPA who got rejected for a branch that means that all the other students below him should also be rejected). Now we need not verify the branch request by other students for the branch the S_1 got rejected. So we can decline/remove all the requests to that branch for all the students below him(this will eliminate all the rejections to that branch by rule-5) . The student immediately below S_1, let's say S_2, there won't be rule 5 into consideration and has only the possible floored out and roofed out reasons for rejection . So we can treat all the students below S_1 the same way we treated S_1.

B. Conclusion

We have to find out the different structures for both the scenarios (floored out and roofed out) in which we can tackle these cases.
We decided to put this Allocation Verifier approach on hold because we discovered a better analogy for the Branch Change Problem with Hospital-Residence Problem. So we went on to the research papers on it, and the literature review for those can be found in the following chapter.

IV. LITERATURE REVIEW

In Computer Science, the stable marriage problem (also known as the stable matching problem) is the problem of finding a stable matching equal-sized sets of elements given a preference ordering for each element.
The Hospitals/Residents problem is a many-to-one extension of the stable marriage problem. In this case, each hospital imposes a quota, or an upper limit on the number of seats it can provide. For this problem, there exists at least one stable matching, and finding one can be done in polynomial time. The papers which we will be looking below will study an extension of this problem in which each hospital sets not only an upper limit but also a lower limit on the number of seats. This problem is known as Hospital/Residence problem with Lower Quotas(HR-LQ). We'll study some papers on HR-LQ because the problem is quite similar to our branch change problem. We can find a good analogy between these two problems: hospitals are branches, residents are students and as we are not allowed to go below the 10% of the existing strength this will be the lower quota. But the major difference is that we are already allocated into a branch and we need to relocate them whereas HR-LQ problem is for a new allocation (i.e, if we directly remodel this we'll be allocating them from scratch so they might lose their current branch).
We'll study the below mentioned research papers and try to adapt them to our problem or we will gain the knowledge which might be helpful to tackle the branch change problem.

A. Envy-Free Matchings with Lower Quotas by Yu Yokoi[1]

As mentioned above every HR instance has a stable matching but with the lower quotas we won't have any stable matchings guaranteed. For such instances, by weakening the fairness properties we can expect an existence of an envy free matching.
They characterized the existence of an envy-free matching in the following way: Let I be a given HR-LQ instance, and I' be the HR instance generated by eliminating lower quotas and replacing upper quotas with the original lower quotas from I. They showed that I has an envy-free matching if and only if every hospital in a stable matching of I' is full. Combining this characterisation with the Rural Hospital Theorem results in an efficient algorithm for determining the existence of an envy-free matching for an HR-LQ instance.

1) Comments on this study

This study does not help much with our problem. This paper does not talk about the non wastefulness and also weakens fairness. It only determines the existence of an envy-free matching, but because we already have a given allocation in our problem and the given allocation itself turns out to be an envy-free matching. So, in any given instance, we'll have an envy-free matching, and we won't have to look for it again.

B. Strategy proof Matching with Minimum Quotas[2]

1) Model and Notations

A market consists of $(S, C, p, q, >_S, >_C)$ where $S = s_1, s_2, ..., s_n$ is a set of n students, $C = c_1, c_2, ..., c_m$ a set of m colleges. We use $p = (p_{c1}, ..., p_{cm})$ and $q = (q_{c1}, ..., q_{cm})$ to denote lists of minimum and maximum quotas, respectively, for each college, where $p_c \geq 0$, $q_c > 0$, $p_c \leq q_c$ and $n \geq q_c + \Sigma_{c' \neq c} (p_{c'})$ for all $c \in C$ and $\Sigma_{c \in C} p_c < n < \Sigma_{c \in C} q_c$. We can have '$e$' as all excess students after the minimum quota is filled for all the branches, i.e, $e = n - \Sigma_{c \in C} p_c$.

Each student s and school c have strict preference relations >s and >c respectively, denoted as >S= (>s)s∈S for students and >C= (>c)c∈C for schools. The set P represents the possible preference relations over C, while P|S| encompasses all preference profiles for all students.

A matching μ is a mapping μ: S ∪ C → 2S ∪ C that adheres to the following conditions:
(1) For all s ∈ S, μ(s) ∈ C.
(2) For all c ∈ C, μ(c) ⊆ S.
(3) For any s ∈ S and c ∈ C, μ(s) = c if and only if s ∈ μ(c).

A pair (s, c) within a matching is deemed a blocking pair if both s and c prefer each other over their current assignments. In other words, if c >s μ(s) and s >c s′ for some s′ ∈ μ(c). A matching is considered fair if no blocking pairs (s, c) exist.
Additionally, a matching μ is non-wasteful if, whenever a student claims an empty seat at school c, the number of students assigned to μ(s) is equal to the quota pμ(s) for that school.
However, it is important to note that simultaneous fair and non-wasteful matchings are not always achievable. In the presence of minimum quotas, it may not be possible to identify matchings that satisfy both fairness and non-wastefulness criteria. This is exemplified in the following scenario involving two students (s1 and s2) and three colleges (c1, c2, and c3) with their respective priorities and quotas.

$>_{c1} >_{c2} >_{c3} s_2\, s_2\, s_1$

$s_1\, s_1\, s_2$

$p_c\ 1\ 0\ 0\ q_c\ 1\ 1\ 1$

$>_{s1} >_{s2} c_2\, c_3\, c_3\, c_2\, c_1\, c_1$

As c_1 has its minimum quota 1, either s_1 or s_2 has to be assigned to c_1 and non-wasteful then requires the other student who didn't assigned to c_1 to take his first preference. If s_1 is assigned to c_1 then s_2 can go to c_3 here, s_1 prefers c_3 and c_3 gives higher priority to s_1 than s_2, and thus forms a blocking pair. It's same with the case where s_2 is assigned c_1. This shows that the fair and non-wasteful is not always possible.

2) Extended-seat deferred Acceptance (ESDA)

We enlarge the original market $(S, C, p, q, >_S, >_C)$ to meet our demands.When the market is expanded, the student population remains constant.
To this we add new extended schools $C*$ such that $q_{c*} = q_c - p_c$ and $p_c = 0$. For the regular school, we make $q_c = p_c$. Both the school and the extended school have the same $>_c$ which is the original $>_c$. For the students preferences if it was $c_1 >_s c_2$ it will become $c_1 >_s c_1* >_s c_2 >_s c_2*$

Algorithm
Let the set *X* represent the students who are not in the waiting list of any other college. We select the first student in set *X* and check his college preference order. According to his college preference order we check whether the student can be placed in his first preference.Let us apply to the most desired school c ∈ C based on s, assuming that college c has not yet rejected him or her. If c is a regular school, let school c select up to the highest ranking students based on >c from among those students who have applied to c but have not yet been rejected by c. Any remaining students are rejected by School C. We add the rejected student to the set *X*.
If *c* is an extended school then we use the above method until the total number of students in all the extended schools is equal to *e*. When the number of students in extended schools is e then from that point onwards we don't add any new students to extended schools but only replace a student of lower priority with a student of higher priority. We repeat this process until set *X* is empty.
Note : All the students who are removed will be added to the end of set *X*.
Significance of e
As we have discussed in the model before $e = |S| - \Sigma p_i$ which is the same as saying that e is the number of excess students who will be left out after all the minimum quotas of the colleges are filled.
So the maximum number of students in the extended schools cannot exceed *e* as if it exceeds *e then* there will be a regular school whose quota is not fulfilled. So by using *e* we can make sure that all the schools fulfill their minimum quota.

Example

The preference order of the colleges and their upper and lower limits are given below.

$>_{c1} >_{c2} >_{c3}$
$s_5\ s_3\ s_1$
$s_4\ s_2\ s_5$
$s_3\ s_5\ s_4$
$s_1\ s_4\ s_3$
$s_2\ s_1\ s_2$

p_c 1 1 1
q_c 2 2 2

The preference order of the students is given below.

$>_{s1} >_{s2} >_{s3} >_{s4} >_{s5}$
$c_1\ c_1\ c_3\ c_1\ c_3$
$c_2\ c_2\ c_1\ c_3\ c_2$
$c_3\ c_3\ c_2\ c_2\ c_1$

Now we will discuss the allocation process of the above students step wise . Note : Here priority of C* =C

We can use *m* to represent the number of students in the extended schools at a particular point .

- Initial condition

The table represents the sets of all the students whose requests are currently accepted by the college (students in the waiting list)

$>_{c1}$					
$>_{c1}*$					
$>_{c2}$					
$>_{c2}*$					
$>_{c3}$					
$>_{c3}*$					

The row below represents the students who are not currently in the waiting list of any college

s_1	s_2	s_3	s_4	s_5		

- First iteration

The table represents the sets of all the students whose requests are currently accepted by the college (students in the waiting list)

$>_{c1}$	s_1				
$>_{c1}*$					
$>_{c2}$					
$>_{c2}*$					
$>_{c3}$					
$>_{c3}*$					

©IJRASET: All Rights are Reserved

The row below represents the students who are not currently in the waiting list of any college

s_2	s_3	s_4	s_5			

- Next iteration

The table represents the sets of all the students whose requests are currently accepted by the college (students in the waiting list)

$>_{c1}$	s_1				
$>_{c1}*$	s_2				
$>_{c2}$					
$>_{c2}*$					
$>_{c3}$					
$>_{c3}*$					

The row represents the students who are not currently in the waiting list of any college

s_3	s_4	s_5			$m = 1$

- Next iteration

The table represents the sets of all the students whose requests are currently accepted by the college (students in the waiting list)

$>_{c1}$	s_1					
$>_{c1}*$	s_2					
$>_{c2}$						
$>_{c2}*$						
$>_{c3}$	s_3					
$>_{c3}*$						

The row represents the students who are not currently in the waiting list of any college

	s_4	s_5				$m = 1$

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VII Jul 2023- Available at www.ijraset.com

Now there will be a clash in $>_{c1}$ between $>_{s1}$ and $>_{s4}$ and $>_{c1}$ gives higher priority for $>_{s4}$. $>_{s1}$ will then be removed from $>_{c1}$'s waiting list and placed in *X*.

$>_{c1}$	s_4				
$>_{c1}*$	s_2				
$>_{c2}$					
$>_{c2}*$					
$>_{c3}$	s_3				
$>_{c3}*$					

The row represents the students who are not currently in the waiting list of any college

	s_5	s_1				$m = 1$

$>_{c1}$	s_4				
$>_{c1}*$	s_2				
$>_{c2}$					
$>_{c2}*$					
$>_{c3}$	s_3				
$>_{c3}*$	s_5				

The row represents the students who are not currently in the waiting list of any college. Now m = e so from here on no extra students can be added to the extended students .(It is only possible to replace students in extended students from now on)

	s_1					$m = 2$

©IJRASET: All Rights are Reserved

Now there will be a clash in $>_{c1*}$ between $>_{s1}$ and $>_{s2}$ and $>_{c1*}$ gives higher priority for $>_{s1}$. $>_{s2}$ will be removed from its waiting list and placed in X.

$>_{c1}$	s_4				
$>_{c1}*$	s_1				
$>_{c2}$					
$>_{c2}*$					
$>_{c3}$	s_3				
$>_{c3}*$	s_5				

The row represents the students who are not currently in the waiting list of any college

	s_2					$m = 2$

21

$>_{c1}$	s_4				
$>_{c1}*$	s_1				
$>_{c2}$	s_2				
$>_{c2}*$					
$>_{c3}$	s_3				
$>_{c3}*$	s_5				

The row represents the students who are not currently in the waiting list of any college

						$m = 2$

From the above steps using ESDA approach we can say that c_1 takes s_1 and s_4, c_2 takes s_2 and c_3 takes s_3 and s_5

- *Comments on ESDA*

ESDA satisfies the fairness criteria but the non-wastefulness may not be satisfied. The order in which students are picked in the usual ESDA algorithm is random, but we may modify the selection process and send them in the order of CGPA. The algorithm, however, does not discriminate between students depending on the sequence in which they are assigned. This might result in rule-5 violations. We also need to check whether ESDA has already handled common situations like swapping, cycles, and chains.

3) Multistage Deferred Acceptance(MSDA)

MSDA satisfies non-wastefulness but weakens fairness. In MSDA we will maintain a universal precedence list $>_{PL}$ that ranks all the students i.e, $s_1 >_{PL} s_2 >_{PL} \cdots >_{PL} s_n$. Here the precedence list is to weaken the fairness, we weaken the blocking pair and we define PL blocking pair. A pair (s, c) is called a PL-blocking pair, if $c >_s \mu(s)$, $s >_c s'$ and $s >_{PL} s'$. A matching is PL-fair if no (s, c) pairs can form a PL-blocking pair.

MSDA is performed in many stages. We reserve certain students at the start of each stage and run the standard DA on the remaining students. Reserve students are used to guarantee that the lower quotas are filled, i.e. the number of students to be reserved at each stage is equal to the total of the lower quotas at that stage. After the students who took part in the assignment have been allocated in that stage and their assignments will be finalized, we will update the upper and lower quotas accordingly. We'll now move on to the sub-problem and continue the process until all students are allocated.

Algorithm

The market is $(S, C, p, q, >_C, >_S)$, the precedence list ranks students as $s_1 >_{PL} s_2 >_{PL} \cdots >_{PL} s_n$.

a) We will start the instance with $R^0 = S$, $p^1{}_c = p_c$ and $q^1{}_c = q_c$ for all $c \in C$. Let the number of students that will be reserved in R^1 be $r^1 = \Sigma_{c \in C} p^1{}_c$.

b) For stage $k \geq 1$, Set $R^k = s_{n-r^k+1}, s_{n-r^k+2}, \cdots, s_n$ i.e, R^k is the set of r^k students with the lowest priority according to $>_{PL}$.

- If R^{k-1} and R^k are not the same, then run the standard DA on the students in $R^{k-1} \setminus R^k$ with upper quotas for the schools equal to $(q^k{}_c)_{c \in C}$.
- If R^{k-1} and R^k are same, then run the standard DA on the students in R^k with upper quotas for the schools equal to $(p^k{}_c)_{c \in C}$

c) Let μ^k be the matching from step $k - 1$, and eliminate all assigned students from S. End the process after all students have been assigned to a college. If not, move on to the next step.

d) New quotas for college are computed as follows:

(a) $q^{k+1}{}_c = q^k{}_c - |\mu^k{}_c|$

(b) $p^{k+1}{}_c = \max 0, p^k{}_c - |\mu^k{}_c|$

(c) $r^{k+1} = \Sigma_{c \in C} p^{k+1}{}_c$

e) Move on to stage $k + 1$.

Following the completion of the algorithm, we are left with a collection of stage-wise matchings $\mu^1, \cdots, \mu^k$. The algorithm's final matching output is the matching, which is the union of all stage-wise matchings. MSDA mechanism is non-wastefulness and PL-fair.

- *Comments on MSDA*

MSDA satisfies the non-wastefulness but it weakens the fairness.

When using this algorithm to solve our problem, we can make the following observations: As this will be allocating them in stages, if a student didn't get his preferred branch there will be only an issue with the upper quota.

In relation to our branch change problem, we can't directly have the universal precedence list, which may result in someone not obtaining his/her preferred allocations and being kicked off of his/her home branch. Consider the following scenario: all EE students who applied for a branch change are in the bottom spots in the precedence order. Then, other branch students who are higher than these EE students can get into EE based on their preferences, and these guys may lose their own spot if they do not get into their better preferences. We also can't have different precedence lists for each branch because it would violate rule 5. As a result, we need to work on a better precedence order if we try to adapt this.

V. ESDA APPROACH

As Our Branch Change Problem is mostly about fairness rather than Non-wastefulness. So, we try to adapt the ESDA into our Branch Change Problem.

A. Model and Notations

In our problem, we require the $(S, B, p, q, >_S, >_B)$ where $S = \{s_1, s_2, ..., s_n\}$ is a set of n students who applied, $B = \{b_1, b_2, ..., b_m\}$ a set of m branches. We use $p = (p_{b1}, ..., p_{bm})$ and $q = (q_{b1}, ..., q_{bm})$ to denote lists of minimum and maximum quotas, respectively, for each branch, where $p_b \geq 0$, $q_b > 0$, $p_b \leq q_b$ and $n \geq q_b + \Sigma_{b' \neq b} (p_{b'})$ for all $b \in B$ and $\Sigma_{b \in B} p_b < n < \Sigma_{b \in B} q_b$. Each student has a strict preference relation $>_s$ over B, while each branch b has a strict priority relation $>_b$ over S, i.e, branches prefer their home branch students at the top then after the students who requested branch change to that particular branch in CGPA order.

To this we add new extended branches B^*such that $q_b* = q_b - p_b$ and $p_b = 0$. For the regular branch, we make $q_b = p_b$. Both the branch and the extended branch have the same $>_b$ which is the original $>_b$. For the students' preferences, if it was $b_1 >_s b_2$ it will become $b_1 >_s b^*_1 >_s b_2 >_s b^*_2$.

We can have 'e' as all excess students after the minimum quota is filled for all the branches, i.e, $e = n - \Sigma_{b \in B} p_b$.

B. Algorithm

Calculate the quotas. Let X be the set of students in CGPA order.

1) Begin with an empty matching

2) We'll choose the top CGPA students from X. If no student exists, end the algorithm

3) Let s apply to the most preferred branch b according to his/her preference over branch that has not yet rejected him/her. If b is a regular branch, let branch b choose the students up to q_b highest priority students according to branch b among those who thus far applied to b but have not yet been rejected by b. Branch b rejects any remaining students, and the algorithm returns to step 2. If b is an extended branch, move to the next step.

4) In this step, As b is an extended branch then we use the above method until the total number of students in all the extended branches is equal to e or the upper quota of that extended branch has hit i.e, number of students in branch $b^*$is equal to q_b* . When the number of students in extended branches is e or when the particular extended branch reaches it's upper quota then

a) If students s's home branch is b then

- If there are people present in extended school b who are originally not from branch b then remove the student with the lowest CGPA among them and add s in the newly created vacancy.
- If this is not possible then remove the lowest CGPA non-home branch student among all the extended branches and add the students in extended branch b.

b) If students s's home branch is not b then

- Remove the lowest CGPA non-home student from the extended branch b and add students in the new slot created.
- If there is no non-home student in extended branch b then reject s.

C. Observations and Relevant Proofs

1) Observations

a) All excess branch seats being filled by home branch students will happen only if everybody is in their home branch (no branch change scenario).

b) Once we hit excess = e, it remains so forever.

Reason - after hitting e, if a student in the waiting list wants to be allocated a slot then there are 2 possibilities

(i) Student enters a normal branch slot where there is no change in e

(ii) Student enters an extended branch where to accommodate him we will kick out someone else from an extended branch.

So, e does not change once it hits.

c) Type-1 deterioration for a branch b : s replaces s'in b where b prefers s' over s. This case is not possible because according to our algorithm a lower preference student will never be able to replace a higher preference student. From this point we can say that, No Type-1 deterioration will happen till the first reapplication is granted. No reapplication will be granted if no Type-1 deterioration occurs. This chicken and egg argument proves that Type-1 deterioration won't happen and reapplications are useless.

d) Type-2 deterioration for a branch b : b losing one student

This will happen during 4.a.ii in the algorithm above after an excess = e is attained. But this will not help any reapplication since type-2 deterioration only happens to excess-branches and no free additions are possible to excess branches once the excess hits e.

2) Relevant Proofs

Algorithm Terminates

Lemma : If we prove that our algorithm finishes in finite time (i.e, have to show the total number of applications is bounded and reapplications are unnecessary) then we can say that it terminates.

Proof: A student will be able to apply for a branch only once, we can say that because if a student is rejected by a branch b once the only chance that student can enter b again if b deteriorates. But from observation 3 and 4 we can say that the student will not have another chance to join b. With this we can say that any student can make a maximum number of applications which is equivalent to $2|B|$ ($|B|$ normal branches and $|B|$ extended branches) and the total number of students is $|S|$. So, maximum possible number of steps in our algorithm is $\leq 2|B||S|$ which is finite.

Allocation is fair

Lemma : Rejection of a student will be either direct reject or acceptance and push-out later. Proof : When a student s applies to a branch, if that branch was filled with students better than him up to the quota then it rejects directly. If the branch was not up to quota then it can accept the students and might push out when students better than him as preferred by branch applies to that particular branch.

Lemma : If a rejection is fair then the allocation is fair.

Proof : For direct rejection, accept and push-out later rejection if we show these rejections are fair then the allocation is fair.

a) Direct Rejection: If a student (let's call them "Student S") is denied from a regular branch (referred to as "Branch B"), it means that Branch B has achieved its full capacity with students who have higher rankings or preferences from Branch B's perspective. By the end of the process, Student S must have a lower CGPA than every non-home student assigned to Branch B because the ranking of students assigned to a branch improves as the algorithm runs.

If, on the other hand, s is rejected from an extended branch b^*, it must be because either

- Higher ranked students fill that extended branch to capacity (original max - original min), in which case the previous justification will still hold true, or
- e other students are tentatively assigned to extended branches and everyone currently in $b^*$is better than s (in the preference list of b^*). In that case no one lower than s is in b^* at that time. No student weaker than s will later get b^*since either b^*swaps a weaker student for a better student, loses its weakest student or it accepts a home branch student.

b) s was once accepted by b^* and later rejected : If s was pushed-out because someone better asked for b^* and s was the then lowest in b^*, then the earlier argument holds.

If s was pushed-out of b^* because of some home-branch student getting into another excess branch b^*. In this case, s was not only the weakest guy in b^*, he's the weakest among all the extended branches and hence this rejection is fair.

Once the eth student is approved, the branch that rejects student s may have a group of students with a higher CGPA than s, which may be empty. No student ranked lower than those already holding at this branch can be admitted due to the specified sequence in which students are admitted (based on CGPA).

Every student gets his/her home branch or better

By the construction of the 4.a in the algorithm proposed in the section 6.2. A student applies the branches in a preferred order and if he doesn't get any of his/her preferred branches then his last preference his home branch will definitely get to him.(as his home branch prefers him on top of other branch students)

D. Implementation

1) Terminology

- Sanctioned Strength: The sanctioned strength is the number of seats allowed to a specific branch.
- Existing Strength: Existing Strength refers to the number of students who have joined a particular branch at the start of the first semester. Existing strength may differ from sanctioned strength since some seats may be unoccupied.

- Minimum Possible Strength: As per our Branch Change Rules, the strength of a class should not fall below the existing strength by more than ten percent of the sanctioned strength and as per Rule-4 of our branch change rules, a minimum of one student will be eligible irrespective of rule-3. So the Minimum possible Strength is, minimum of (Existing Strength - 10% of the Sanctioned Strength) and (Existing Strength - 1)
- Maximum Possible Strength: As per our Branch Change Rules, the strength of a class should not go above the sanctioned strength by more than ten percent of the sanctioned strength, i.e, Maximum Possible Strength = 110% of the Sanctioned Strength
- No.of Students Applied: This is the number of students applied/requested for the branch change.
- Non-Applied Students: This is number of students who did not applied/requested for the branch change, i.e, Non-Applied Students = Existing Strength - No.of Applied Students

2) Algorithm Implementation

The programming language used for the implementation is python, complete code can be found in the link given in Appendix A. Here we will see the main snippets of code. In our implementation we will exclude all the Non-Applied Students from the existing strengths of all the branches as they are not involved and they are fixed with their branches. Now, our existing strengths of all the branches will change accordingly. So, our new existing strengths modify our Minimum and Maximum possible strengths as Lower Quota and Upper Quota respectively,

- Lower Quota: Lower Quota is the minimum number of students the branch has to be filled with. Lower Quota = Minimum Possible Strength - Non-Applied Students
- Upper Quota: Maximum number of students a branch can take into it. Upper Quota = Maximum Possible Strength - Non-Applied Students.

The Global Excess (*e*) is the maximum number of students who can be filled above the lower quotas i.e, *e* = No.of Applied Students - (Sum of all Lower Quota's) and Students in Extended is the number of students in the extended branches at a instance in the allocation. It goes on increasing and the maximum it can go up to is *e*. Once it hits me, it never gets down.

a) Input

- The inputs for the file will be 2 CSV files, one will contain the details of all the students who have requested for branch change and other will contain the details of all the branches.
- The exact format of the studentInfo.csv file is, one line will consist of one student details in the exact order of <student name, student rollNo, GPA, JEE rank, home branch, preference branch1, preference branch 2, preference branch 3, preference branch 4>
- So, the total number of lines - 1(excluding header) in the file will be the total number of students who have applied for branch change.
- The exact format of the branchInfo.csv file is one line will consist of one branch details in the exact order of <branch Name, existing strength, sanctioned strength>
- A student may/may not give 4 preference branches(As our college has only 4 branches). We have to give his least preference as his home branch. The rest all preferences corresponding value in his record will be a null string ''. For Example, if a students gives only his first preference, then his second preference will be taken as his home branch and Preference 3 and 4 as Null.

Branch	ExistingStrength	SanctionedStrength
CS	50	50
EE	44	50
ME	36	37
CE	35	37

Fig. 6.1 Sample Branch Input

	Name	RollNumber	GPA	JEERank	Home	Pref1	Pref2	Pref3	Pref4`
1	132001026	9.3	2000	ME	CS	ME			
2	102001033	9.3	4000	CE	CS	CE			
3	122001044	9.23	5885	EE	CS	EE			
4	132001031	9.13	7065	ME	EE	ME			
5	122001019	9.07	9462	EE	CS	EE			
6	102001028	9	4099	CE	CS	EE	ME	CE	

Fig. 6.2 Sample Student Input

b) Output

The Output will be a csv file, studentFinalFile.csv, it contains all the fields from the student input file additionally it adds a column named Final Allocated Branch which contains the final allocation of all the applied students.

	Name	Roll Number	GPA	Home Branc	Pref1	Pref2	Pref3	Pref4	Final Allocated
1		132001026	9.3	ME	CS	ME			CS
2		102001033	9.3	CE	CS	CE			CS
3		122001044	9.23	EE	CS	EE			CS
4		132001031	9.13	ME	EE	ME	CS		EE
5		122001019	9.07	EE	CS	EE			CS
6		102001028	9	CE	CS	EE	ME	CE	CS

Fig. 6.3 Sample Output

3) Code Snippets for Different Scenarios that Can Occur During an Allocation

a) Case 1: When the applicant is allocated in to regular branch (refer figure 6.4)

```
if(len(b.alloc) < int(b.lowerquota)):
    b.alloc = self.allocating(b, s)
    break
```

Fig. 6.4 Case-1 scenario during an allocation

b) Case 2: When students in extended has not hit *e* and student is being added to extended branch (refer figure 6.5)

```
if(len(b.alloc) < int(b.upperquota) and stu_in_ext < e):
    b.alloc = self.allocating(b, s)
    break
```

Fig. 6.5 Case-2 scenario during an allocation

c) Case 3: When students in extended has hit *e* or the branch in which the applicant is currently applying reaches its upper quota and the current applying branch is the home branch of the applicant and the student least preferred by the branch is not originally from that branch (refer figure 6.6)

```
if(stu_in_ext == e or len(b.alloc) == int(b.upperquota)):
    if(s.home_branch == b.branch_name):
        least_s = b.alloc[-1]
        if(least_s.home_branch != b.branch_name):
            self.pq.put((-1*float(least_s.GPAwithRank), least_s))
            b.alloc.pop()
            b.alloc = self.allocating(b, s)
            break
```

Fig. 6.6 Case-3 scenario during an allocation

d) Case 4: When students in extended has hit *e* or the branch in which the applicant is currently applying reaches its upper quota and the current applying branch is the home branch of the applicant and the student least preferred by the branch is originally from that branch (refer figure 6.7)

```
if(stu_in_ext == e or len(b.alloc) == int(b.upperquota)):
    if(s.home_branch == b.branch_name):
        least_s = b.alloc[-1]
        if(least_s.home_branch == b.branch_name):
            b_ind = list(bdict.values())[0]
            for i in bdict.values():
                if(len(i.alloc) > int(i.lowerquota)):
                    least_s_gpa = i.alloc[-1].GPAwithRank
                    if(least_s.home_branch != i.branch_name and least_s_gpa < b_ind.alloc[-1].GPAwithRank):
                        b_ind = i
            least_s = b_ind.alloc[-1]
            self.pq.put((-1*float(least_s.GPAwithRank), least_s))
            b_ind.alloc.pop()
            b.alloc = self.allocating(b, s)
            break
```

Fig. 6.7 Case-4 scenario during an allocation

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VII Jul 2023- Available at www.ijraset.com

e) Case 5: When students in extended has hit *e* or the branch in which the applicant is currently applying reaches its upper quota and the current applying branch is not the home branch of the applicant(refer figure 6.8)

```
if(stu_in_ext == e or len(b.alloc) == int(b.upperquota)):
    if(s.home_branch != b.branch_name):
        least_s = b.alloc[-1]
        if(least_s.home_branch != b.branch_name):
            if((least_s.GPAwithRank) < (s.GPAwithRank)):
                self.pq.put((-1*float(least_s.GPAwithRank), least_s))
                b.alloc.pop()
                b.alloc = self.allocating(b, s)
                break
```

Fig. 6.8 Case-5 scenario during an allocation

4) Test Cases

We will analyze a test case manually and compare to the generated output from code where all the 5 different scenarios will occur while computing the allocation. Testcase 1:

a) Branch Info:

Branch Info is given with the already computed Lower and Upper Quota's.

Branch Name	Lower Quota	Upper Quota
CS	0	2
EE	2	4
ME	2	4
CE	3	5

b) Student Info

In the Student Info below, students are given in the order of highest Priority i.e, in the order of CGPA (for CGPA clashes the student with least JEE Rank is given higher priority)

35

Name	Home Branch	Pref 1	Pref 2	Pref 3	Pref 4
1	EE	CS	EE		
2	ME	CS	EE	ME	
3	CE	CS	EE	ME	CE
4	EE	CS	EE		
5	ME	CS	EE	ME	
6	EE	CS	EE		
7	ME	CS	EE	ME	
8	ME	CS	EE	ME	
9	CE	CS	EE	ME	CE
10	CS	EE	ME	CS	
11	CE	CS	CE		

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

c) Branch Allocation Table

All the branches were made empty and their lower and upper quotas were calculated and given in the above branch info table. All the students will be placed in the waiting list. Now we'll take the student with the highest priority in the waiting list ant start the allocation process for that student. After every step of allocation we will see through which cases of scenario(refer section 6.4.3) he is being allocated to that particular branch.

5) Initialisation

In the table the cell color white represents students below lower quota and cell color silver represents students above lower quota

CS					
EE					
ME					
CE					

The below row is the students waiting list. It also has the Global Excess (*e*) and Students in Extended (*stu in ext*)

1	2	3	4	5	6	7	8	9	10	11	*stu in ext* = 0	*e* = 4

- Step - 1

Student 1 will request CS and will get it. (case-2)

CS	1				
EE					
ME					
CE					

2	3	4	5	6	7	8	9	10	11		*stu in ext* = 1	*e* = 4

- Step - 2

Student 2 will request CS and will get it.(case-2)

CS	1	2	-	-	-
EE					-
ME					-
CE					

3	4	5	6	7	8	9	10	11			*stu in ext* = 2	*e* = 4

- Step - 3

Student 3 will request CS and will get rejected as CS reached it's upper quota and then he will apply for EE and get it.(case-1)

CS	1	2	-	-	-
EE	3				-
ME					-
CE					

4	5	6	7	8	9	10	11				*stu in ext* = 2	*e* = 4

- Step - 4

Student 4 will request CS and will get rejected as CS reached it's upper quota and then he will apply for EE and get it.(case-1)

CS	1	2	-	-	-
EE	3	4			-
ME					-
CE					

5	6	7	8	9	10	11					*stu in ext* = 2	*e* = 4

Step - 5

Student 5 will request CS and will get rejected as CS reached it's upper quota and then he will apply for EE and get it. (case-2)

CS	1	2	-	-	-
EE	3	4	5		-
ME					-
CE					

6	7	8	9	10	11						*stu in ext* = 3	*e* = 4

- Step - 6

Student 5 will request CS and will get rejected as CS reached its upper quota and then he will apply for EE and get it. (case-2)

CS	1	2	-	-	-
EE	3	4	5	6	-
ME					-
CE					

7	8	9	10	11							*stu in ext* = 4 limit reached	*e* = 4

- Step - 7

Student 7 will request CS and will get rejected then request EE and will get rejected and then he will apply for ME and get it. (case-1)

CS	1	2	-	-	-
EE	3	4	5	6	-
ME	7				-
CE					

8	9	10	11								*stu in ext* = 4 limit reached	*e* = 4

- Step - 8

Student 8 will request CS and will get rejected then request EE and will get rejected and then he will apply for ME and get it. (case-1)

CS	1	2	-	-	-
EE	3	4	5	6	-
ME	7	8			-
CE					

9	10	11									*stu in ext* = 4 limit reached	*e* = 4

- Step - 9

Student 9 will request CS and will get rejected then request EE and will get rejected and then request ME and will get rejected as students in extended has hit *e* we can't directly add into it's extended quota and As students who are already in the branches are better than him and any of the branch he requested earlier is not his home branch then finally he will apply for CE and get it. (case-1)

CS	1	2	-	-	-
EE	3	4	5	6	-
ME	7	8			-
CE	9				

10	11										*stu in ext* = 4 limit reached	*e* = 4

- Step - 10

Student 10 will request EE and will get rejected then request ME and will get rejected as students in extended has hit *e* we can't directly add into it's extended quota and then it will apply for CS which is his home branch and get it. As this is his home branch if home branch of least preferred student in CS is not CS then he gets removed here it is Student 2, who is originally from ME will be sent back to the waiting list (case-3)

CS	1	10	-	-	-
EE	3	4	5	6	-
ME	7	8			-
CE	9				

2	11										*stu in ext* = 4 limit reached	*e* = 4

- Step - 11

Student 2 will request EE and get it. As, from the students who are allocated into EE student 5 is least preferred and Among Student 2 and 5 EE prefers Student 2. So, he will get it and Student 5 will be sent back to the waiting list (case-5)

CS	1	10	-	-	-
EE	3	4	2	6	-
ME	7	8			-
CE	9				

5	11										*stu in ext* = 4 limit reached	*e* = 4

- Step - 12

Student 5 will request ME and it is his home branch so he should get it. As ME has all of its home branch students so no one can be sent back. So, here we will look into non-home least preferred students from all the extended branches and remove the least among them so that we can put Student 5 into an extension of ME. So, Here it is Student 3 will be sent back to the waiting list (case-4)

CS	1	10	-	-	-
EE	4	2	6		-
ME	7	8	5		-
CE	9				

3	11										*stu in ext* = 4 limit reached	*e* = 4

- Step - 13

Student 3 will request CE and get it. (case-1)

CS	1	10	-	-	-
EE	4	2	6		-
ME	7	8	5		-
CE	9	3			

11											*stu in ext* = 4 limit reached	*e* = 4

- Step - 14

Student 11 will request CE and get it. (case-1)

CS	1	10	-	-	-
EE	4	2	6		-
ME	7	8	5		-
CE	9	3	11		

											stu in ext = 4 limit reached	*e* = 4

Here we can see that the allocation we get manually is the same as the the allocation we get after running the same test case in the code(Fig 6.9)

Name	Roll Number	GPA	Home Branch	Pref1	Pref2	Pref3	Pref4	Final Allocated Branch
1	1	9.3	EE	CS	EE			CS
2	2	9.3	ME	CS	EE	ME		EE
3	3	9.23	CE	CS	EE	ME	CE	CE
4	4	9.13	EE	CS	EE			EE
5	5	9.07	ME	CS	EE	ME		ME
6	6	9.0	EE	CS	EE			EE
7	7	8.97	ME	CS	EE	ME		ME
8	8	8.93	ME	CS	EE	ME		ME
9	9	8.87	CE	CS	EE	ME	CE	CE
10	10	8.87	CS	EE	ME	CS		CS
11	11	8.87	CE	CS	CE			CE

Fig. 6.9 Test Case Output run on Algorithm

E. Results

We ran this algorithm on the actual branch change requests of 2020 batch which has 35 students. The details are as follows:

Branch	ExistingStrength	SanctionedStrength
CS	50	50
EE	44	50
ME	36	37
CE	35	37

Fig. 6.10 BranchInfo of Batch 2020

	Name	RollNumber	GPA	JEERank	Home	Pref1	Pref2	Pref3	Pref4
1		132001026	9.3	2000	ME	CS	ME		
2		102001033	9.3	4000	CE	CS	CE		
3		122001044	9.23	5885	EE	CS	EE		
4		132001031	9.13	7065	ME	EE	ME	CS	
5		122001019	9.07	9462	EE	CS	EE		
6		102001028	9	4099	CE	CS	EE	ME	CE
7		122001025	8.97	3378	EE	CS	EE		
8		122001018	8.93	8789	EE	CS	EE		
9		132001014	8.87	2500	ME	CS	ME		
10		122001004	8.87	4500	EE	CS	EE		
11		102001036	8.87	6000	CE	EE	ME	CS	CE
12		122001027	8.8	5046	EE	CS	EE		
13		122001022	8.77	4586	EE	CS	EE		
14		132001035	8.7	5000	ME	CS	EE	ME	
15		102001031	8.7	6500	CE	CS	EE	ME	CE
16		122001021	8.57	5017	EE	CS	EE		
17		122001047	8.5	3716	EE	CS	EE		
18		122001034	8.43	4200	EE	CS	EE		
19		122001012	8.43	6200	EE	CS	EE		
20		132001008	8.37	7000	ME	EE	ME		
21		102001018	8.37	8000	CE	EE	ME	CS	CE
22		132001033	8.37	9000	ME	EE	ME		
23		102001020	8.3	8925	CE	CS	EE	ME	CE
24		122001039	8.27	9156	EE	CS	EE		
25		102001012	8.07	6060	CE	CS	EE	CE	
26		102001005	8	8109	CE	CS	EE	ME	CE
27		122001028	7.97	6343	EE	CS	EE		
28		102001007	7.83	4320	CE	EE	CE		
29		102001026	7.57	8591	CE	CS	EE	ME	CE
30		102001019	7.5	4062	CE	CS	CE		
31		102001024	7.3	9218	CE	CS	EE	ME	CE
32		102001039	7.1	6268	CE	ME	CE		
33		102001035	7.03	4433	CE	CS	ME	EE	CE
34		102001023	6.97	6490	CE	CS	CE		
35		102001027	6.53	5614	CE	ME	EE	CS	CE

Fig. 6.11 Student Info of Batch 2020

	Name	Roll Number	GPA	Home Branc	Pref1	Pref2	Pref3	Pref4	Final Allocated
1		132001026	9.3	ME	CS	ME			CS
2		102001033	9.3	CE	CS	CE			CS
3		122001044	9.23	EE	CS	EE			CS
4		132001031	9.13	ME	EE	ME	CS		EE
5		122001019	9.07	EE	CS	EE			CS
6		102001028	9	CE	CS	EE	ME	CE	CS
7		122001025	8.97	EE	CS	EE			EE
8		122001018	8.93	EE	CS	EE			EE
9		132001014	8.87	ME	CS	ME			ME
10		122001004	8.87	EE	CS	EE			EE
11		102001036	8.87	CE	EE	ME	CS	CE	EE
12		122001027	8.8	EE	CS	EE			EE
13		122001022	8.77	EE	CS	EE			EE
14		132001035	8.7	ME	CS	EE	ME		EE
15		102001031	8.7	CE	CS	EE	ME	CE	EE
16		122001021	8.57	EE	CS	EE			EE
17		122001047	8.5	EE	CS	EE			EE
18		122001034	8.43	EE	CS	EE			EE
19		122001012	8.43	EE	CS	EE			EE
20		132001008	8.37	ME	EE	ME			EE
21		102001018	8.37	CE	EE	ME	CS	CE	CE
22		132001033	8.37	ME	EE	ME			ME
23		102001020	8.3	CE	CS	EE	ME	CE	CE
24		122001039	8.27	EE	CS	EE			EE
25		102001012	8.07	CE	CS	EE	CE		CE
26		102001005	8	CE	CS	EE	ME	CE	CE
27		122001028	7.97	EE	CS	EE			EE
28		102001007	7.83	CE	EE	CE			CE
29		102001026	7.57	CE	CS	EE	ME	CE	CE
30		102001019	7.5	CE	CS	CE			CE
31		102001024	7.3	CE	CS	EE	ME	CE	CE
32		102001039	7.1	CE	ME	CE			CE
33		102001035	7.03	CE	CS	ME	EE	CE	CE
34		102001023	6.97	CE	CS	CE			CE
35		102001027	6.53	CE	ME	EE	CS	CE	CE

Fig. 6.12 Final Allocated Branches of Batch 2020 run on our algorithm

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

This allocation exactly matches with the allocation that the academic section has arrived at for this set of branch change requests. This algorithm takes less than a second for execution for this input, We also tested this for a random data with 4 branches and 1000 students it still takes less than a second for execution. Where as the brute force algorithm takes around 58 minutes for 4 branches and 20 students to arrive at the solution.

VI. CONCLUSION AND FUTURE WORK

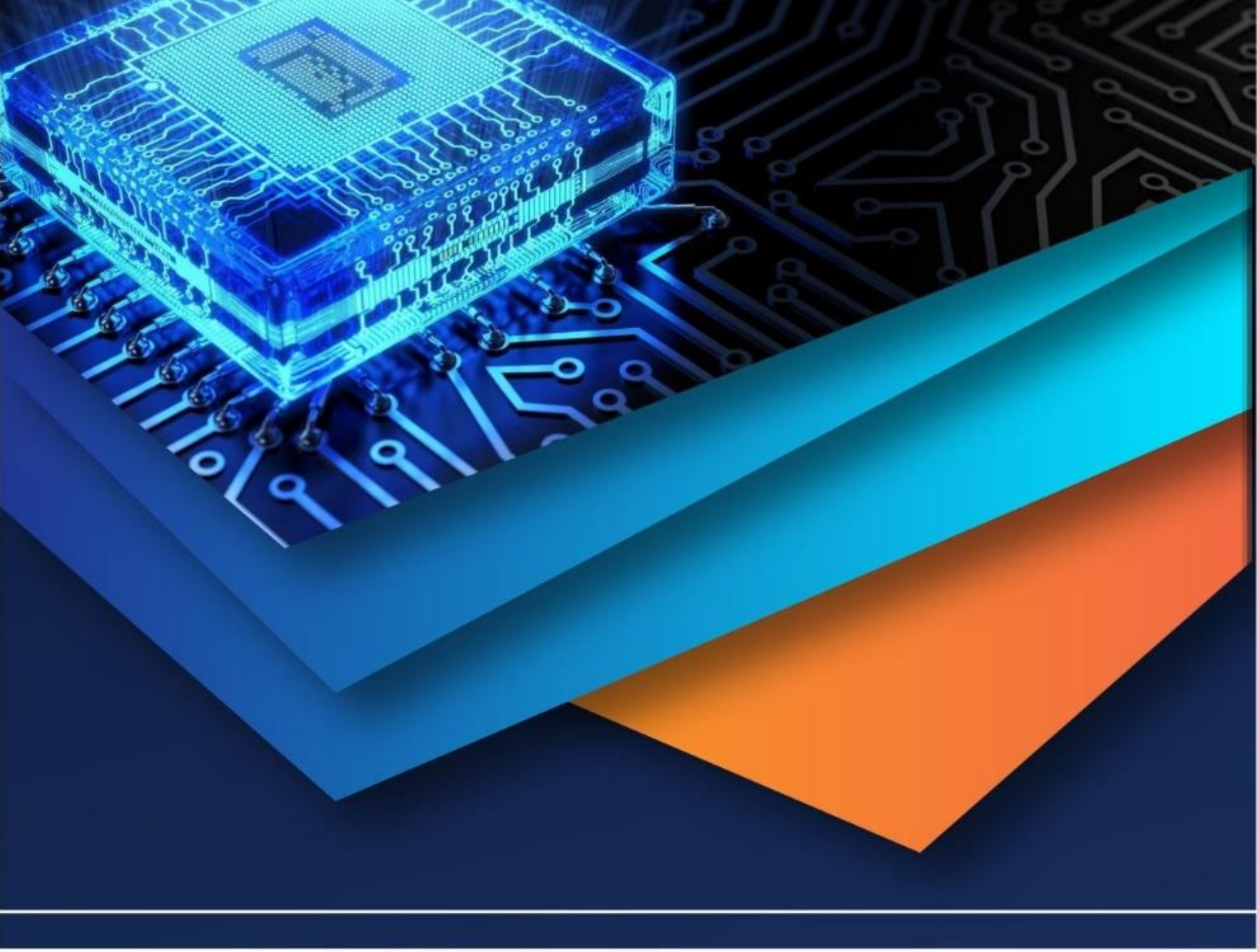
We have successfully designed a polynomial time algorithm for the Branch Change problem with supremum rule. Time complexity of our algorithm is $O(b^2s^3log(s)))$ compared to exponential time ($(O(b^s))$) of the Brute force solution for the problem with only supremum rule. Where s is the number of students and b is the number of branches.

In our code we are sorting the students of non home branches while allocating. By using priority queue instead of sorting there then we might improve our time complexity. We are yet to prove that the result we get using this algorithm is the optimal solution or not. If it isn't then how close it is to the optimal solution.

REFERENCES

[1] Y. Yokoi, “Envy-free matchings with lower quotas,” 2018.
[2] D. FRAGIADAKIS, A. IWASAKI, P. TROYAN, S. UEDA, and M. YOKOO, “Strate gyproof matching with minimum quotas,” 2015.
[3] Branch Change Rules, IIT Palakkad. [Online]. Available: https://drive.google.com/ file/d/1VulhJ0R7LsTVPBclxvd8gllMddTTl4Rk/view
[4] Kasireddy Durga Prasad Reddy, “Graph theoretic approach on branch reallocation,” B.Tech Project Report, Computer Science and Engineering, IIT Palakkad, 2019.
[5] Vishnu Teja, “Graph theoretic approach on branch change,” B.Tech Project Report, Computer Science and Engineering, IIT Palakkad, 2020.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)