Stable Group Project Assignment Algorithm

Lydia Peercy, Ryan Schuerkamp, James Heyne, Joe Rutkowski

1 Problem Definition

1.1 Variable Table

Variable Name	Significance
p	A project is a project being done by a group of m students
s	A student
S	Set of all students
L	List of projects based on the student's preferences
R	Total sum of preferences
P	Set of all projects

1.2 Constraints

- C_1: $3 \leq m \leq 4$
- C_2: A student *s* must provide a preference for every project *p* in their list of preferences *L*
- C_3: Each student *s* mapped to only 1 project *p*
- C_4: $|S| \geq 6$
- C_5: $|P| = \text{ceiling}(|S| \div m)$

1.3 Objective

Assign each student *s* to a project *p* based on their highest preference possible from *L* to minimize the total sum of preferences *R* and return the set of projects *P* where each student *s* has been assigned a project *p*.

1.4 Formal Problem Definition

Given constraints C_1, C_2, C_3, C_4, and C_5, a set of students *S*, a list of project preferences *L* for each student *s*, and a list of projects *P*, we want to assign each student *s* to a project *p* while minimizing the total sum of preferences of selected projects *R,* similar to a stable marriage problem.

2 Proposed Algorithm

2.1 Algorithm Design

This algorithm takes a list of projects *P* and a set of students *S* where each student *s* has a list of project preferences *L*. Then, it assigns each student *s* to a project *p* while minimizing the total sum of preferences *R*. The algorithm continues until every student *s* is assigned a project *p*, and each project *p* has 3 or 4 students. Finally, it returns the set of projects *P* where each project *p* has been assigned students.

The algorithm first calculates the number of projects with 3 students based on the cardinality of the set of students *S*. Then, it maps each student *s* to their most preferred project based on their preferences *L*. If a project *p* is overfilled (a project *p* has 5 students when it can only have 4), the algorithm then checks all the second preferences of the students currently assigned to it. If a student *s* has their second preference available, they are moved to their second preference. The algorithm continues checking students' preferences until project *p* has the required number of students. If none of the students' second preferences are available, the algorithm checks the students' third preferences, then fourth, and continues until enough students are moved to another project *p* so that each project *p* has the required number of students. Once all the projects have the required number of students and each student *s* is assigned to a project *p*, the algorithm returns the set of all projects *P*.

2.2 Pseudocode

```
Algorithm 1 AssignStudents
1:  if $S_S < 6 \vee P_S \neq \lceil(S_S \div 4)$ then
2:      return null
3:  end if
4:  comment: number of projects that will be assigned three students
5:  numProjectsOfThree = $(4 - S_S.length \,\%\, 4) \,\%\, 4$
6:  for s $\in S_S$ do
7:      s $\Rightarrow L_0$
8:  end for
9:  comment: Sort $P$ based on the number of students assigned to the projects in descending
    order
10: sort($P$, $key = |s \Rightarrow p|$,descending= $True$)
11: for i = 0 to i ¡ $|P_S|$ - numProjectsOfThree do
12:     comment: numPreference is the current preference being checked
13:     numPreference = 1
14:     while $|s \Rightarrow p_i| > 4$ do
15:         comment: For every student s assigned to project $p_i$
16:         for $s \Rightarrow p_i$ do
17:             comment: If less than 4 students assigned to S's $L_{numPreference}$ project
18:             if $|s \Rightarrow L_{numPreference}| < 4$ then
19:                 comment: Assign s to their $L_{numPreference}$ project
20:                 $L_{numPreference} \leftarrow s$
21:             end if
22:         end for
23:         numPreferences++
24:     end while
25: end for
26: for $i = |P|$ - numProjectsOfThree to i < $|P|$ do
27:     comment: numPreference is the current preference being checked
28:     numPreference = 1
29:     while $|s \rightarrow p_i| > 3$ do
30:         for $s \rightarrow p_i$ do
31:             if $|s \rightarrow L_{numPreference}| < 3$ then
32:                 $L_{numPreference} \leftarrow s$
33:             end if
34:         end for
35:         numPreferences++
36:     end while
37: end for
38: return P
```

Line 1 checks there are at least 6 students (constraint C_4) and that there are enough projects for the students (constraint C_5); line 2 returns null if these constraints are not met. Line 5 calculates the number of projects that will be assigned three students. Lines 6 and 7 assign each student *s* to their most preferred project based on their preferences *L.* Line 10 sorts the projects *P* in descending order based on the number of students with them as their first preference, so the most popular project is first. Line 11 iterates through the projects that will be assigned 4 students. Inside the for loop, the while loop line 14 executes while the project *p* has more than 4 students assigned to it. Lines 15-23 go through each student *s* assigned to the project *p* and their preferences *L*, checking if their second (then third, fourth, …) preferred project (L_1) has an opening and moving them if so. This process continues until the project *p* has 4 students assigned to it. Similar to line 11, line 26 iterates through the projects that will have 3 instead of 4 students assigned to them. On line 29, the while loop and code inside the while loop lines 30-35 execute similarly to lines 14-23 except terminate when the project *p* has 3 students assigned to it. Line 38 returns the set of projects *P* where each project *p* has been assigned either 3 or 4 students.

2.3 Example Cases

2.3.1 Best Case Example

Assume that *S* contains 9 students, numbered s_1 through s_9 and that *P* contains the projects p_1, p_2, and p_3. The students' project preferences are as follows:

Student	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
First	p_3	p_2	p_1	p_3	p_2	p_1	p_3	p_2	p_1
Second	p_2	p_1	p_2	p_2	p_3	p_3	p_2	p_3	p_3
Third	p_1	p_3	p_3	p_1	p_1	p_2	p_1	p_1	p_2

First, the algorithm checks if the amount of total students |*S*| is greater than or equal to 6 and if the total amount of projects |*P*| is equal to ceiling($\frac{S}{4}$) (line 1). Since |*S*| = 9 and $9 \geq 6$, the first assumption is true. Additionally, since |*P*| = 3 = ceiling($\frac{9}{4}$), the second assumption is also true. Next, it calculates the number of projects that will have three members instead of four, which is found via $(4 - |S|\ \%\ 4)\ \%\ 4$ (line 5). Plugging *S* into this equation produces $(4 - |9|\ \%\ 4)\ \%\ 4$, which solves to $3\ \%\ 4$ and simplifies to 3. This means all three of the projects will be assigned three students instead of four.

Now that it is known how many students will be assigned to each project *p*, the algorithm can begin assigning students to projects based on their preferences *L*. It iterates through *S* and assigns each student *s* to their first choice project L_0 (lines 6 and 7). Line 10 sorts the projects *P* in descending order based on the number of students with them as their first preference, so the most popular project is first. In this case, there is a spot for each student *s* in their first choice project L_0, so all projects have 3 students assigned to them. The end result looks as follows.

p_1	p_2	p_3
s_3	s_2	s_1
s_6	s_5	s_4
s_9	s_8	s_7

Lines 11-25 don't execute since every project is assigned 3 students. Line 26 iterates through all 3 projects, but the while loop (lines 29-36) never executes since none of the projects are assigned more than 3 students. Then, line 38 returns *P* where each project *p* has been assigned 3 students. In this example, every student *s* got their first choice L_0, and it is the best case possibility of this algorithm.

2.3.2 Worst Case Example

Assume that *S* contains 8 students, numbered s_1 through s_8, and that *P* contains the projects p_1 and p_2. The students project preferences are as follows:

Student	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
First	p_2	p_2	p_2	p_2	p_2	p_2	p_2	p_2
Second	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1

First, the algorithm checks if the amount of total students |*S*| is greater than or equal to 6 and if the total amount of projects |*P*| is equal to ceiling($\frac{S}{4}$) (line 1). Since |*S*| = 8 and $8 \geq 6$, the first assumption is true. Additionally, since |*P*| = 2 = ceiling($\frac{8}{4}$), the second assumption is also true. Next, it calculates the number of projects that will have three members instead of four, which is found via $(4 - |S| \% 4) \% 4$ (line 5). Plugging

S into this equation produces $(4 - |8| \% 4) \% 4$, which solves to $4 \% 4$ and simplifies to 0. This means none of the projects will be assigned three students instead of four.

Now that the algorithm knows how many students will be assigned to each project *p*, it can begin assigning students to projects based on their choices. It iterates through *S* and assigns each student *s* to their first choice project L_0 (lines 6 and 7). In this case, every single student has the same first preference of p_2, so the first iteration assigns each student *s* to p_2.

p_1	p_2
	s_1
	s_2
	s_3
	s_4
	s_5
	s_6
	s_7
	s_8

Line 10 sorts the projects *P* in descending order based on the number of students with them as their first preference, so the most popular project p_2 is first. Since each project *p* has a max size of four, we need to iterate through the popular project p_2 and assign the 4 students to their next highest ranked project. This is done from lines 11-25 and looks as such.

p_1	p_2
s_5	s_1
s_6	s_2

s_7	s_3
s_8	s_4

As every student has the same preferences, we need to iterate through each project in *P* until all projects are filled. Since this scales poorly, this scenario of every student *s* having the same preferences *L* is the worst-case scenario. Lines 26-37 don't execute since there are no groups of 3. Line 38 returns *P* where each project *p* has been assigned 4 students.

2.3.3 Average Case Example

Assume that *S* contains 9 students, numbered s_1 through s_9 and that *P* contains the projects p_1, p_2, and p_3. The students project preferences are as follows:

Student	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
First	p_3	p_3	p_1	p_3	p_3	p_1	p_3	p_2	p_1
Second	p_2	p_1	p_3	p_2	p_2	p_2	p_1	p_3	p_3
Third	p_1	p_2	p_2	p_1	p_1	p_3	p_2	p_1	p_2

First, the algorithm checks if the amount of total students $|S|$ is greater than or equal to 6 and if the total amount of projects $|P|$ is equal to ceiling($\frac{S}{4}$) (line 1). Since $|S|$ = 9 and $9 \geq 6$, the first assumption is true. Additionally, since $|P|$ = 3 = ceiling($\frac{9}{4}$), the second assumption is also true. Next, it calculates the number of projects that will have three members instead of four, which is found via $(4 - |S|\ \%\ 4)\ \%\ 4$ (line 5). Plugging *S* into this equation produces $(4 - |9|\ \%\ 4)\ \%\ 4$, which solves to $3\ \%\ 4$ and simplifies to 3. This means all three of the projects will be assigned three students instead of four.

Now that it is known how many students will be assigned to each project *p*, the algorithm can begin assigning students to projects based on their preferences *L*. It iterates through *S* and assigns each student *s* to their first choice project L_0 (lines 6 and 7).

p_1	p_2	p_3
s_3	s_8	s_1
s_6		s_2
s_9		s_4
		s_5
		s_7

Line 10 sorts the projects P in descending order based on the number of students with them as their first preference, so the most popular project p_3 is first.

p_3	p_1	p_2
s_1	s_3	s_8
s_2	s_6	
s_4	s_9	
s_5		
s_7		

Lines 11-25 don’t execute since every project is assigned 3 students. On the first iteration of the for loop on line 26, the while loop (lines 29-36) moves two students from p_3 to their second choice.

p_3	p_1	p_2
s_1	s_3	s_8
s_2	s_6	s_5
s_4	s_9	

	s_7	

On the second iteration of the for loop on line 26, the while loop (lines 29-36) moves one student from p_1 to p_2.

p_3	p_1	p_2
s_1	s_3	s_8
s_2	s_6	s_5
s_4	s_9	s_7

On the third iteration of the for loop on line 26, the while loop (lines 29-36) doesn't execute since the last project p_2 doesn't have more than 3 students, so none need to be moved. Line 38 returns *P* where each project *p* has been assigned 3 students.

2.4 Time Complexity Analysis

Lines 1-5 are constant time O(1) operations since they perform simple computations. Lines 6-8 are O(*S)* since it assigns every student *s* in *S* to their most preferred project based on their preferences *L*, and accessing their preferences is a constant time operation. Line 10 is O(*P*log*P*) since sorting is an O(NlogN) operation in most standard programming languages.

Line 11 is an O(*P)* operation since it'll execute for all projects that need to be assigned 4 students which could be all the projects. Line 13 is a constant time, O(1), operation since we are declaring and initializing a variable. Line 14 is an O(*L*) operation since the worst-case scenario is there are no openings until someone's last preference, so the loop would run |*L*| - 1 times since the first preference was already iterated through. Line 16 is an O(*S*) operation since all students could have the same first preference. Note if all students have the first preference, the other projects will be open at least initially, so the worst case of line 14 can't happen when the worst case of line 16 does. Lines 18-21 are constant time O(1) operations. Overall, the time complexity of lines 11-25 is at worst O(*PLS*)*.* This would occur when all projects will be assigned 4 students, there are no open projects until the students' last preference, and every student has the same first preference; if all students share the same first preference, projects will be open before their last preference, so the absolute worst case can never happen. It is possible all students have the exact same preferences which would be the practical worst case.

Lines 26-37 perform the same function as lines 11-25 above but for projects that will be assigned 3 students instead of 4. Line 26 is a constant time O(1) operation since it will run at most 3 times because we can only have 3 groups of 3. If we had 4 groups of 3, we would have 3 groups of 4 instead, which would then be processed by lines 7-13. Line 28 is a constant time O(1) operation. Line 29 is an O(*L*) operation like line 14 above. Line 30 is an O(*S*) operation like line 16 above; however, all groups of 4 will have been filled, so there will be fewer students to iterate through. Lines 31-33 are constant time O(1) operations. Overall, the time complexity of this section of code is O(*PS)* but will likely stay below that since the projects with 4 students will have been filled.

Line 38 is a constant time O(1) operation. Overall, the time complexity of our algorithm is O(*S* + *P*log*P* + *PLS* + *LS)*, which can be simplified to O(*PLS)*. Given constraints C_1, C_2, and C_3, the overall time complexity can be rewritten as O(S^3) because the number of projects |*P*| and the number of preferences |*L*| are both dependent on the number of students |*S*|.

3 Experimental Evaluation

3.1 Performance Metrics

The algorithm will be evaluated using two metrics:

- *Run time* will be measured by getting the difference between the time immediately before and after the algorithm is called. The run time and its growth as the input grows will be compared to the theoretical time complexity.
- *Sum of preferences R* will help quantify how well the algorithm assigns students given their preferences. For example, if all students get their first preference, *R* equals 1 times the number of students (*R* = |*S*| * 1). If all students get their second preference, *R* equals 2 times the number of students (*R* = |*S*| * 2). A lower sum of preferences *R* means more students got their higher preferences. The sum of preferences *R* produced by the algorithm will be compared to the sum of preferences *R* produced by an algorithm that randomly assigns students to projects.

Run time and sum of preferences *R* will help evaluate both the performance and accuracy of the algorithm.

3.2 Description of Data Set

The data sets used in this report were produced by a python script developed by one of the students in the group (script available on the GitLab as data_generator.py). The script generated random names for students using python’s faker library and random preferences using the random library until the specified number of students were created, each with a name and list of preferences. The students produced were then written to text files labeled by the number of students students.txt (e.g. 1000students.txt). Text files were produced containing data for 31, 62, 125, 250, 500, and 1000 students (all files available on the GitLab inside the data folder). In each file, each line contains the data for one student. The student’s first and last name is listed and is followed by their list of preferences (e.g. 'Patricia James' 2 3 4 8 6 7 5 1). Projects are simply a list of unique numbers in this data set since project names have a negligible effect on the algorithm. Having 6 unique data sets of different sizes enables the comparison of run times and sums of preferences *R* across input size as described in section 3.1 Performance Metrics.

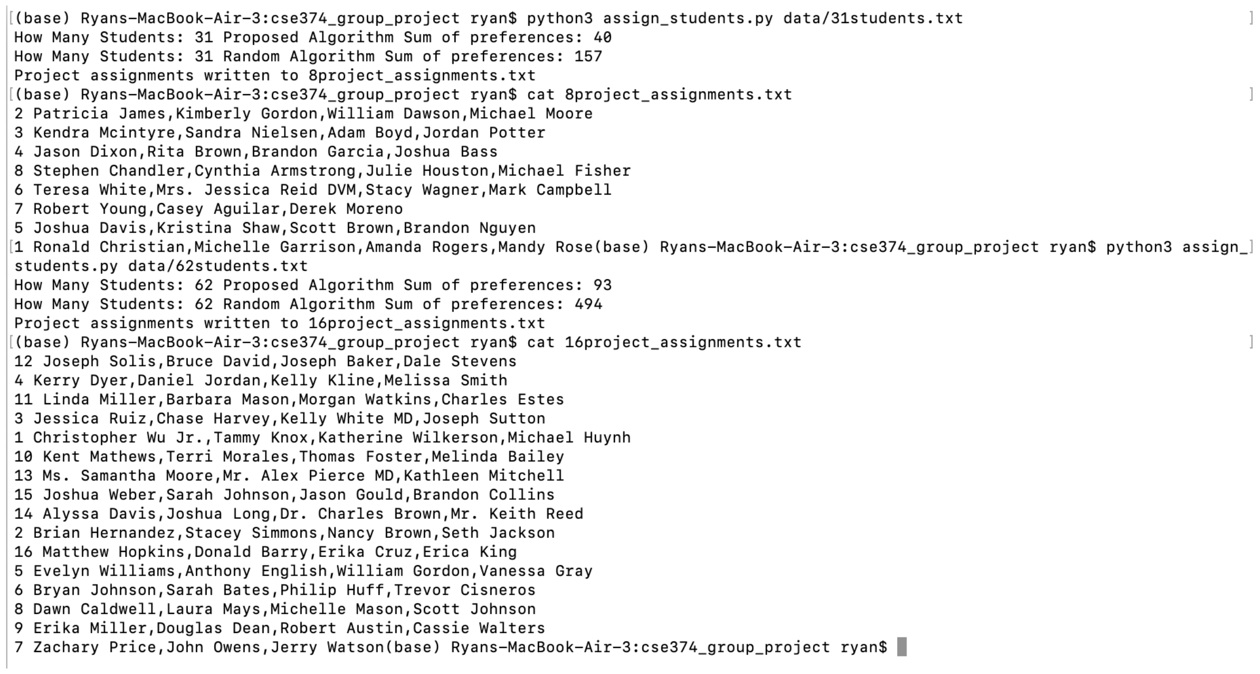
3.3 Experimental Setting

The code produced, proposed algorithm, and random algorithm were run on one of the student's laptops with the following specifications:

Laptop	MacBook Air (Retina, 13-inch, 2019)
Operating System	macOS Mojave Version 10.14.6
Central Processing Unit	1.6 GHz Intel Core i5
Main Memory Size (RAM)	8 GB
Graphics Card	Intel UHD Graphics 617 1536 MB

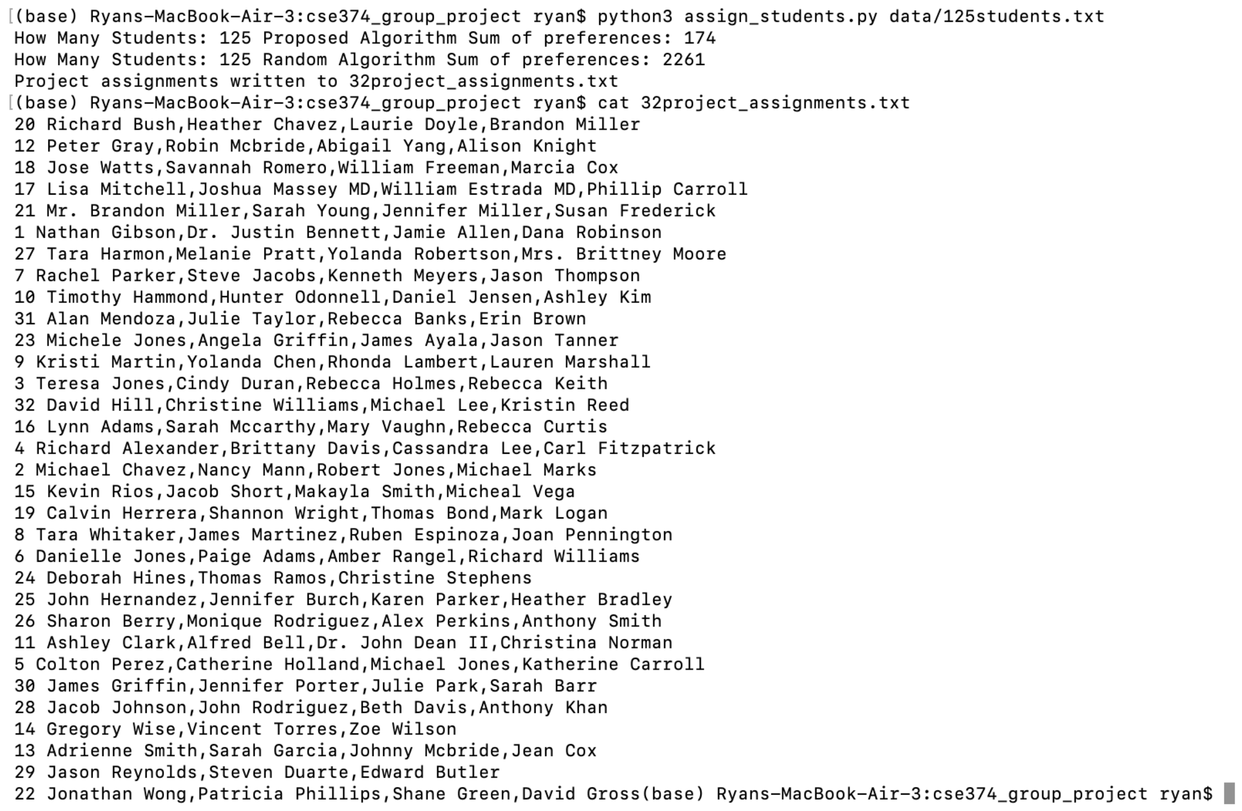
4 Results

4.1 Screenshots of Test Runs



```
[(base) Ryans-MacBook-Air-3:cse374_group_project ryan$ python3 assign_students.py data/31students.txt     ]
How Many Students: 31 Proposed Algorithm Sum of preferences: 40
How Many Students: 31 Random Algorithm Sum of preferences: 157
Project assignments written to 8project_assignments.txt
[(base) Ryans-MacBook-Air-3:cse374_group_project ryan$ cat 8project_assignments.txt     ]
2 Patricia James,Kimberly Gordon,William Dawson,Michael Moore
3 Kendra Mcintyre,Sandra Nielsen,Adam Boyd,Jordan Potter
4 Jason Dixon,Rita Brown,Brandon Garcia,Joshua Bass
8 Stephen Chandler,Cynthia Armstrong,Julie Houston,Michael Fisher
6 Teresa White,Mrs. Jessica Reid DVM,Stacy Wagner,Mark Campbell
7 Robert Young,Casey Aguilar,Derek Moreno
5 Joshua Davis,Kristina Shaw,Scott Brown,Brandon Nguyen
[1 Ronald Christian,Michelle Garrison,Amanda Rogers,Mandy Rose(base) Ryans-MacBook-Air-3:cse374_group_project ryan$ python3 assign_]
students.py data/62students.txt
How Many Students: 62 Proposed Algorithm Sum of preferences: 93
How Many Students: 62 Random Algorithm Sum of preferences: 494
Project assignments written to 16project_assignments.txt
[(base) Ryans-MacBook-Air-3:cse374_group_project ryan$ cat 16project_assignments.txt     ]
12 Joseph Solis,Bruce David,Joseph Baker,Dale Stevens
4 Kerry Dyer,Daniel Jordan,Kelly Kline,Melissa Smith
11 Linda Miller,Barbara Mason,Morgan Watkins,Charles Estes
3 Jessica Ruiz,Chase Harvey,Kelly White MD,Joseph Sutton
1 Christopher Wu Jr.,Tammy Knox,Katherine Wilkerson,Michael Huynh
10 Kent Mathews,Terri Morales,Thomas Foster,Melinda Bailey
13 Ms. Samantha Moore,Mr. Alex Pierce MD,Kathleen Mitchell
15 Joshua Weber,Sarah Johnson,Jason Gould,Brandon Collins
14 Alyssa Davis,Joshua Long,Dr. Charles Brown,Mr. Keith Reed
2 Brian Hernandez,Stacey Simmons,Nancy Brown,Seth Jackson
16 Matthew Hopkins,Donald Barry,Erika Cruz,Erica King
5 Evelyn Williams,Anthony English,William Gordon,Vanessa Gray
6 Bryan Johnson,Sarah Bates,Philip Huff,Trevor Cisneros
8 Dawn Caldwell,Laura Mays,Michelle Mason,Scott Johnson
9 Erika Miller,Douglas Dean,Robert Austin,Cassie Walters
7 Zachary Price,John Owens,Jerry Watson(base) Ryans-MacBook-Air-3:cse374_group_project ryan$
```

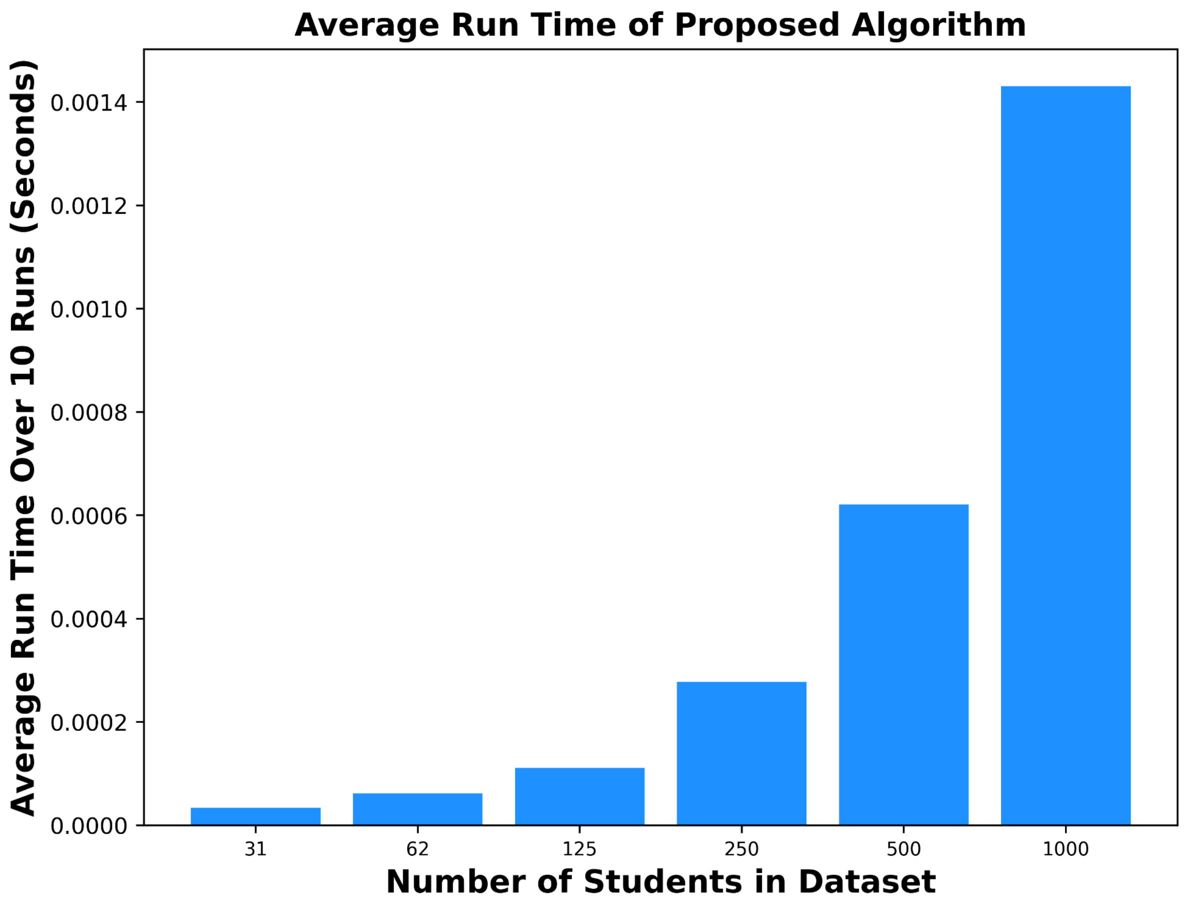
The above screenshot displays the test runs and results of the algorithm on data sets containing 31 and 62 students.



```
[(base) Ryans-MacBook-Air-3:cse374_group_project ryan$ python3 assign_students.py data/125students.txt
How Many Students: 125 Proposed Algorithm Sum of preferences: 174
How Many Students: 125 Random Algorithm Sum of preferences: 2261
Project assignments written to 32project_assignments.txt
[(base) Ryans-MacBook-Air-3:cse374_group_project ryan$ cat 32project_assignments.txt
20 Richard Bush,Heather Chavez,Laurie Doyle,Brandon Miller
12 Peter Gray,Robin Mcbride,Abigail Yang,Alison Knight
18 Jose Watts,Savannah Romero,William Freeman,Marcia Cox
17 Lisa Mitchell,Joshua Massey MD,William Estrada MD,Phillip Carroll
21 Mr. Brandon Miller,Sarah Young,Jennifer Miller,Susan Frederick
1 Nathan Gibson,Dr. Justin Bennett,Jamie Allen,Dana Robinson
27 Tara Harmon,Melanie Pratt,Yolanda Robertson,Mrs. Brittney Moore
7 Rachel Parker,Steve Jacobs,Kenneth Meyers,Jason Thompson
10 Timothy Hammond,Hunter Odonnell,Daniel Jensen,Ashley Kim
31 Alan Mendoza,Julie Taylor,Rebecca Banks,Erin Brown
23 Michele Jones,Angela Griffin,James Ayala,Jason Tanner
9 Kristi Martin,Yolanda Chen,Rhonda Lambert,Lauren Marshall
3 Teresa Jones,Cindy Duran,Rebecca Holmes,Rebecca Keith
32 David Hill,Christine Williams,Michael Lee,Kristin Reed
16 Lynn Adams,Sarah Mccarthy,Mary Vaughn,Rebecca Curtis
4 Richard Alexander,Brittany Davis,Cassandra Lee,Carl Fitzpatrick
2 Michael Chavez,Nancy Mann,Robert Jones,Michael Marks
15 Kevin Rios,Jacob Short,Makayla Smith,Micheal Vega
19 Calvin Herrera,Shannon Wright,Thomas Bond,Mark Logan
8 Tara Whitaker,James Martinez,Ruben Espinoza,Joan Pennington
6 Danielle Jones,Paige Adams,Amber Rangel,Richard Williams
24 Deborah Hines,Thomas Ramos,Christine Stephens
25 John Hernandez,Jennifer Burch,Karen Parker,Heather Bradley
26 Sharon Berry,Monique Rodriguez,Alex Perkins,Anthony Smith
11 Ashley Clark,Alfred Bell,Dr. John Dean II,Christina Norman
5 Colton Perez,Catherine Holland,Michael Jones,Katherine Carroll
30 James Griffin,Jennifer Porter,Julie Park,Sarah Barr
28 Jacob Johnson,John Rodriguez,Beth Davis,Anthony Khan
14 Gregory Wise,Vincent Torres,Zoe Wilson
13 Adrienne Smith,Sarah Garcia,Johnny Mcbride,Jean Cox
29 Jason Reynolds,Steven Duarte,Edward Butler
22 Jonathan Wong,Patricia Phillips,Shane Green,David Gross(base) Ryans-MacBook-Air-3:cse374_group_project ryan$ 
```

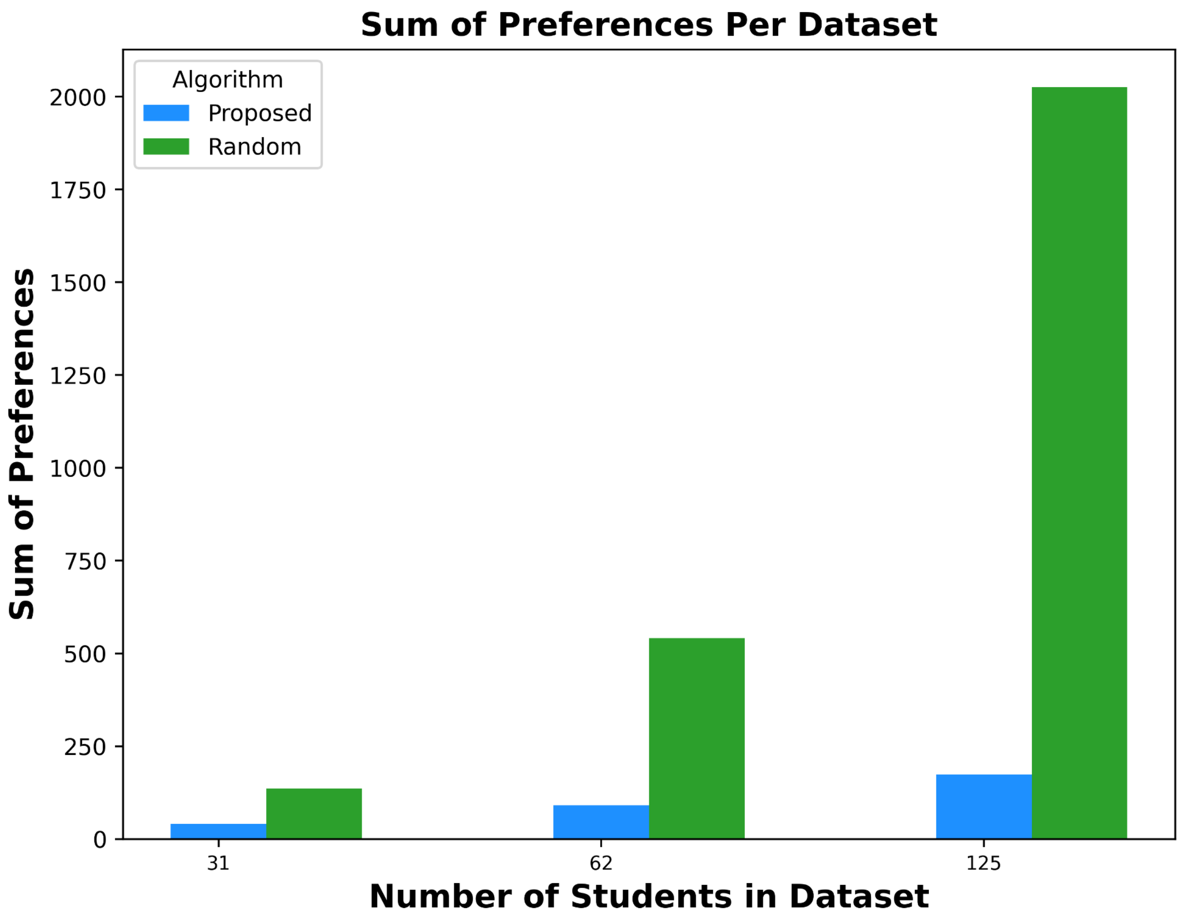
The above screenshot depicts the test run and results on a dataset containing 125 students. For brevity, screenshots of the other tests runs and outputs aren't included, but the outputs of the other test runs are available in the results folder in GitLab.

4.2 Run Time



The figure above depicts the average run time of the algorithm across 10 runs on each of the datasets. As the number of students doubles, the average run time of the algorithm just over doubles.

4.3 Sum of Preferences



The figure above depicts the sum of preferences for both the proposed algorithm and an algorithm that randomly assigns students to projects. The proposed algorithm vastly outperforms the random assignment algorithm. As the input size increases, the proposed algorithm outperforms the random algorithm to a greater extent. This trend is further depicted in the table below which records the number of students and the sum of preferences for both the proposed algorithm and random algorithm for up to 1000 students.

Number of Students	Proposed Algorithm Sum of Preferences	Random Algorithm Sum of Preferences
31	40	147
62	91	509
125	174	1919

250	354	8069
500	762	30711
1000	1616	127705

5 Discussion

5.1 Comparison to Theoretical Time Complexity

The results depicted in section 4.2 display that the average run time of the algorithm aligns well with the theoretical time complexity detailed in section 2.4. The trend of the average run time just over doubling as the input size doubles is a linear trend. The theoretical time complexity of O(PLS) can be rewritten as O(S^3) given constraints C_1, C_2, and C_3. The trend observed in section 4.2 is well below O(S^3). The algorithm was run on data with random student preferences. Because of the random preferences, there wasn't a ton of overlap between students' preferences, mostly avoiding the expensive nested loops. This could be one factor explaining why the average run time of the proposed algorithm depicted in section 4.2 is well below the theoretical time complexity.

In reality, students' preferences are likely not randomly distributed; there are probably a few projects that are popular. Running the proposed algorithm on data that reflects this assumption would likely decrease the performance of the proposed algorithm and be closer to the theoretical time complexity. Further work is needed to generate or gather more realistic datasets that better represent the true distribution of students' preferences and measure the run time of the proposed algorithm on the new datasets.

5.2 Analysis of Sum of Preferences

Section 4.3 displays the proposed algorithm effectively minimizes the sum of preferences, especially when compared to the random algorithm. This causes us to conclude that the proposed algorithm is vastly better than randomly assigning students to projects and is a viable candidate for real-world applications; however, further work is needed to test the proposed algorithm and modify it to make it more suitable for real-world applications.

Notice how the sum of preferences for 1000 students is 1616 (from the table in section 4.3), implying that each student got on average their 1.6th preference. This performance is likely unrealistic when applied to real-world data because the distribution of students' preferences is unlikely to be completely random as in the data on which the algorithm was tested; however, this is a promising result regardless, as it displays that the algorithm can effectively accomplish its goal of assigning students to projects while maximizing collective student happiness by minimizing the sum of preferences. Further work is needed to gather or generate more realistic data and measure the sum of preferences produced by the algorithm.

5.3 Potential Applications to Real-World Data and Future Work

As discussed in the two prior sections, the data used in this study assumes students' preferences are randomly distributed. In reality, there are likely a few projects that are popular among students. Additionally, students likely have well-defined top preferences but may not care about the rest of their choices as much. The random preferences used in this study partially reflect this, but more work is needed to generate or obtain more representative datasets. Moreover, the algorithm could be modified or the data easily preprocessed to allow students to specify a set number of preferences and fill in the rest randomly. This would ease the burden on students and make the proposed algorithm more likely to be applied, especially as the number of students grows because students likely don't want to provide or care about more than 5 or 10 preferences.

In this paper, we analyzed the results of the proposed algorithm on up to 1000 students. Realistically, the current algorithm won't be applied to an input size this large, as class sizes are unlikely to reach this threshold. Even if they do, the professor is unlikely to assign and grade 250 unique group projects. In such a case, the proposed modifications allowing students to specify some preferences and leave the rest random would be essential. Additionally, the proposed algorithm could be modified to allow duplicate projects to ease the burden of grading.

6 Conclusion

In this paper, we formally defined the problem of group project assignment, proposed an algorithm, evaluated our implementation of the algorithm, and shared and analyzed the results. We found that the proposed algorithm is a viable solution to group project assignment. Further work is required to test the proposed algorithm on more realistic datasets and to modify the algorithm to allow students to specify a few preferences and leave the rest randomized. The completion of further work could help apply the proposed algorithm to group project assignments in the real world, reducing the burden of work on instructors and making students happier.