Algorithm Design Document

Team Tech Support
Derek Felson
Margarita Otochkina
Guelor Emanuel

Submitted to
Dr. Christine Laurendeau
for the course COMP 3004

Due date
November 4, 2015

Introduction

This document describes the algorithm used by the cuPID project to take a list of students in a class and match them into optimal teams based on the project settings and the information in each student profile. This document assumes the reader is already familiar with cuPID and the context in which the algorithm will run.

This document has three main parts: qualifications, rules, and algorithm. Each student has to fill complete as profile consisting of 28 questions, detailed in the qualifications section. These qualifications are used as the basis of many rules, which together determine how “good” a team is. The final part of this document explains the algorithm that combines uses those rules in combination to determine the optimal teams for a project.

Qualifications

Every student profile consists of 28 qualifications, and each qualification includes an *answer*, a *minimum preferred value*, and a *maximum preferred value*. The preferred values describe the range of values that the student is looking for in their ideal team members. To be concise, this document may refer to the minimum preferred value and maximum preferred values as the *min* and *max* values of a qualification, or together as *preferences* or *preferred range*.

The qualifications make more sense when understood in terms of 7 categories.

Part 1: Goals

A good team must share the same goals, work habits, and quality standards. Teams that do not tend to have conflict, do poorly, and break up before the project is finished. The rules for this section will be detailed later, but they tend to both group similar students together and pay strong attention to the students' preferred ranges.

Question 1: What grade are you aiming to get on the project? Range is 0-12: (0 = F, 1 = D-, 2 = D, 3 = D+, 4 = C-, 5 = C, 6 = C+, 7 = B-, 8 = B, 9 = B+, 10 = A-, 11 = A, 12 = A+)

Question 1 is about their goals: what grade they want to get on the project. We will try to group people with similar goals together, and we will also pay careful attention to each student's preferred range. This may be our most important question, since if students cannot agree on what grade they are aiming for, they will likely be in constant conflict.

Question 2: Do you tend to start work at the last minute or early? Range is 1-5: 1 (very last minute), 2 (last minute), 3 (no tendency), 4 (early), 5 (very early)

Question 2 is about their work habits. There are many types of work habits we could have asked about, but we chose to ask only about whether they start their work early or late; we have a limited number of questions to ask, and this one is important and general enough to almost always make a big difference.

People who answer this question differently tend to have difficulty working with each other, so to avoid conflict we prefer to group students who give similar answers here. A student's preferred range indicates what types of people they are willing to work with, and how flexible they are about accepting people who work differently.

Question 3: What kind of quality standards do you hold your own work to?

Range is 1-5: 1 (just get it done) - 5 (everything should be perfect). Values 2, 3, and 4 are points between the two extremes.

Question 4 is about their quality standards. Some students just want to get things done, and some students want everything to be perfect. The answer to this question should correlate well, but not perfectly, with Question 1 (grades). It gives us a perspective that is more independent from student cynicism; just because someone likes everything to be perfect doesn't mean they will say they are realistically aiming for an A+. Students who have different quality standards will tend to be in constant conflict with each other, and we would like to avoid that.

Part 2: Grades

Teams work best when the members all have roughly the same level of ability in the course the project is for. When some students understand everything and others are doing very poorly, the team will tend to have conflict, whether it is over how long it takes to do something, the amount of fixing needed, or something else. If we want to predict how well a student will perform in the current course, it helps to look at how well they did in previous courses, especially similar ones.

While it is true that grades are not the most important criteria for matching team members (that would be shared goals, work habits, and quality standards), and they are not what defines the ultimate worth of a student (that would be an oversimplification), grades are still important.

We considered the possibility of grouping students with bad grades along with students with good grades, and hoping that the overall project grades are raised that way, but we decided that the likely increase in conflict was not worth it. A group composed entirely of C students may still work together so well that they get a B, for example, but a group that mixes A students with C students will likely fight and break up.

Question 4: What is your GPA?

Range is 0-12: (0 = F average, 12 = A+ average)

Question 6 asks for the student's overall GPA. There are many types of courses in the computer science curriculum, and some courses will be more relevant to certain projects than others. Since we do not know in advance anything about what projects cuPID will be used for, other than that they are for 3rd or 4th year computer science courses, we have to be as general as possible while still being useful. Much of a 3rd or 4th year computer science student's GPA will reflect how they performed in a variety of courses relevant to just about any project the system could be used for.

Question 5: What was your grade in COMP 2401? (Intro to systems programming)

Range is 0-12: (0 = F, 1 = D-, 2 = D, 3 = D+, 4 = C-, 5 = C, 6 = C+, 7 = B-, 8 = B, 9 = B+, 10 = A-, 11 = A, 12 = A+)

Question 7 asks for their grade in COMP 2401, Introduction to Systems Programming. This course uses the C programming language and is a good overall indicator of a student's programming skill. We considered asking about grades in a first year course, but we decided that such a qualification may place too strong an emphasis on how much a student knew about programming *before* entering university, rather than how well they applied themselves during it. Also, first year courses may be too easy for computer science majors. By second year, the courses are better predictors of how students will perform in their 3rd and 4th years.

Question 6: What was your grade in COMP 2404? (Intro to software engineering)

Range is 0-12: (0 = F, 1 = D-, 2 = D, 3 = D+, 4 = C-, 5 = C, 6 = C+, 7 = B-, 8 = B, 9 = B+, 10 = A-, 11 = A, 12 = A+)

Question 8 asks for their grade in COMP 2404, Introduction to Software Engineering. This course uses the C++ programming language. We decided to ask about both courses because programming skill is likely an important part of any computer science project, and we want to estimate it as accurately as possible.

Part 3: When to work

Beyond just whether to start on an assignment early or late, teams typically need to be able to find time to meet, talk about the project, and sometimes do work together. Students who cannot agree on when to meet or do work will tend to have conflict and perform less well. For example, if you try to put one morning person with three night owls, it makes communication problems that much more likely.

Question 7: Are you willing to work on the project over reading week / spring break?

Range is 1-5: 1 (not at all), 2 (reduced hours), 3 (flexible / no preference), 4 (work extra over breaks), 5 (work a lot extra over breaks)

Question 3 addresses whether the team will work over the weekly break. In Carleton, 3rd and 4th year computer science courses are not offered in the summer, and the fall and winter terms both have a week-long break. Teams that include people who want to work during the break and people who do not are more likely to have conflict and do poorly on the coinciding parts of the project.

The preferred range a student enters for this question defines how flexible they are about what their other group members do, and the rules will place a high value on this.

Question 8: Do you tend to work early in the mornings? Range is 1-5: 1 (never), 2 (rarely), 3 (flexible / no preference), 4 (often), 5 (always)

Question 9 is about whether they are a morning person. Obviously morning people are going to have synergy with other synergy, and it may be harder for them to work with people who like to do all their work between midnight and 3 in the morning. The preferred range here shows how flexible they are about working with other people who do (or do not) like working in the mornings.

Question 9: Do you tend to work late at night? Range is 1-5: 1 (never), 2 (rarely), 3 (flexible / no preference), 4 (often), 5 (always)

Question 10 is about whether they are a night person. The logic is similar to Question 9 (working in the morning), but we chose to ask this as a separate question because we wanted them to be able to specify how flexible they are about working with both morning people and night people. Someone could be fine with mornings and nights, or perhaps be okay with mornings and not okay with nights.

Question 10: How flexible are you about finding time to work on the project or coordinate with your team? Range is 1-5: 1 (very inflexible), 2 (inflexible), 3 (average), 4 (flexible), 5 (very flexible)

Question 11 is about scheduling flexibility in general, rather than a specific time of day when they like to work. Some people have very rigid schedules, perhaps with a part-time job as well as school, or they live far away from campus, and they cannot easily make time for their team outside of certain hours. If we placed multiple such people together on a team, conflict is likely. For this reason we prefer a complementary mix of people with flexible and inflexible schedules (where it isn't possible just to have everyone be flexible).

Part 4: Finding a balance of group roles

In general, you don't want a team that's all one type of person. Different people contribute in different ways, and the most successful teams will have a complementary set of personalities.

These questions come in pairs. We examine students' roles in terms of four complementary relationships: leader/follower, detail-oriented/big picture, likes challenge/likes repetition, and generating ideas/evaluating ideas.

The internet is full of articles on different types of complementary group roles and types of contributors, and we could only pick a few for our survey without making it unwieldy. We ended up not using anything specific we found, and instead we generalized and made our own set of complementary roles.

We chose these four because they seemed to be general enough that they would apply to any 3rd or 4th year computer science project while not being so general that they become useless; many of the roles we found online would be more suited to a business environment or a marketing team than a team of computer science students.

Question 11: Do you like to take charge in a group? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)
Question 12: Do you like to follow other people's lead in a group? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)

Questions 12 and 13 are about how well the student would fit in the roles of leader and follower. We will try to avoid teams where everyone is a leader or everyone is a follower. We considered asking this as a single question, where the student picks a value somewhere along a spectrum of leader-follower, but we wanted to allow flexibility for students that are okay being either a leader or a follower.

The preferred range for question 12 (taking charge) shows how important it is to the student that their team members have initiative. The preferred range for question 13 (following the lead) shows how important it is to the student that their team members can follow instructions. Some people may only want to work with others who have high initiative, or who are good at following instructions. Others may be okay with people who don't take initiative on their own or who are highly independent and hard to manage; both types of people may be challenging, so it is important to identify them and place them with a team that can handle them.

The preferred ranges for questions 12 and 13 may work differently from the other questions in this category.

Question 13: Do you like to focus on getting the little details just right? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)
Question 14: Are you good at focusing on things that affect the bottom line (grades) the most? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)

Questions 14 and 15 are about how well the student would fit into the roles of handling detail work or big picture work. Some people like details, and some people are bored by them. Some people can see the big picture, whereas others may lose sight of what is important and spend too much time on things that do not matter. A big-picture person may, for example, be better at outlining and starting the work that needs to get done, and a detail person may be better at finishing it. The different types of people balance each other out and support each other.

As with questions 12 and 13 (leader or follower), this is a pair of questions so that the user can specify answers for both detail and big picture work. Some people are only good at one, and some people are good at both.

The preferred range for question 14 (detail-oriented) shows what kind of people the student is looking for on their team. While the algorithm will use this value and try to avoid placing students in a group whose average detail-orientedness is outside their preferred range, the penalty for such teams is not large. This is because if we place someone on a team where not everyone is as detail-oriented as they'd like, the team will still manage; on the other hand if we place an A student with a D student, the team will have a lot of conflict. Not all preferences are equally important.

The preferred range for question 15 (big-picture) follows the same logic.

Question 15: Would you be excited to tackle the most challenging parts of the project? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)
Question 16: Are you reliable at handling the repetitive, less challenging parts of a project? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)

Questions 16 and 17 are about how well the student handles challenging work or repetitive work. There is a bit of overlap between this pair and questions 14 (detail-oriented) and 15 (big picture), but those were about broadness of focus, and this is about difficulty. There are challenging details and tedious details; just as there are big-picture things both challenging and tedious that have a large impact on the final grade.

Projects tend to consist of both tedious work and challenging work. Some people are better at one than the other, and a good team will have both kinds. Some people may also be good at both (or neither), which is why we decided to ask this as a pair of questions rather than as a single scale from "likes repetitive" to "likes challenge."

The preferred range for questions 16 (likes challenge) and 17 (likes repetitive) show what kind of people the student is looking for in a team. They follow the same logic as question 14 (detail-oriented).

Question 17: Are you good at generating ideas for an assignment or project? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)
Question 18: Are you good at evaluating and criticising ideas? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)

Questions 18 and 19 are about how well the student fits in the complementary roles of idea generator and idea evaluator. A good team needs a balance of both types of people. You do not want a team where everyone is afraid of criticising anyone, and you do not want a team where nobody comes up with any ideas.

There may be a bit of overlap with questions 12 (leader) and 13 (follower), since leaders need to come up with ideas and evaluate them, but those questions are more about initiative, leadership, and following instructions rather than how they handle ideas.

The preferred ranges for questions 18 and 19 work similarly to what was described in question 14 (detail-oriented).

Part 5: Personality

Some people look as though they would make great team members--on paper--and when it comes time to put them together, they fight anyways. This is often because of some kind of personality mismatch.

Personality is hard to define, and there has been a lot written about what personalities make for a good or a bad team. Unfortunately, we can only ask so many questions about personality before the profile creation process becomes a burden and the algorithm becomes too complicated, and it usually takes a dozen or more questions to get a good idea of someone's personality. Furthermore, many of the professional personality models such as DISC or Myers-Briggs do not readily lend themselves to the generation of team matching rules. DISC, for example, might say that a high D and a high C would be constantly at each other's throats, but they could just as easily complement each other's strengths and weaknesses nicely.

We decided that the best way to approach this was to read a bit, speak to some people, and consider our own experiences, then come up with a few questions of our own that aim to identify a few of the more common and destructive personality conflicts, and make some qualifications and rules to avoid pairing those people together.

In the end we identified three personality traits we thought were important: bluntness, amount of communication, and tolerance of structure.

Question 19: You prefer to be blunt and direct.

Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)

Question 21 is about bluntness. Some people are direct, forceful, and aggressive. Others are indirect, gentle, and passive. Generally it is safer to place blunt people with other blunt people. The student's preferred range for this question is very important, as different people have different levels of tolerance for different styles of communication.

Question 20: You prefer lots of communication, as opposed to being left alone to do your part.

Range is 1-5: Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)

Question 22 is about how much communication someone prefers. Some people like to be left alone to do their part and would resent their teammates checking up on them every day. Others thrive on constant discussion and have a hard time doing anything without talking it over first; it's not that they are helpless, it's just that talking to people is an important part of their creative process. Both kinds of people may annoy the other, so it is important to group them with people who have a similar style. As with question 21, the student's preferred range for this question is important and shows how tolerant they are of different kinds of people.

Question 21: You prefer tasks with clearly defined structure, rules, and deadlines, as opposed to figuring out what you have to do on your own.

Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always)

Question 23 is about how much structure someone prefers. Some people like their work to be highly structured and rule-driven, and others are more independent and prefer to figure things out on their own. There are group members who would love having a team leader send everything a list of specific tasks with associated deadlines twice a week, while other group members would resent that kind of imposition. If we put the two different kinds of people together, conflict becomes more likely. The student's preferred range shows how tolerant they are of different levels of structure.

Part 6: Skillsets

Even in computer science, group projects involve a lot of different kinds of work. Almost all of them involve programming, but often they involve other tasks such as documentation, presentations, organizing, and research. Different people are good at different things, and the team members' skillsets should complement each other. We should also try to avoid putting the people who are great at everything with the people who rate themselves low on everything; conflict is more likely in such groups, and it is unfair to the more skilled students.

Rather than consider these qualifications individually, it makes more sense to consider them as a single set.

Question 22: Rank the statement “I am a strong programmer” Range is: 1 (strongly disagree), 2 (disagree), 3 (neutral), 4 (agree), 5 (strongly agree)
Question 23: Rank the statement “I am good at writing or documentation” Range is: 1 (strongly disagree), 2 (disagree), 3 (neutral), 4 (agree), 5 (strongly agree)
Question 24: Rank the statement “I am good at presentations” Range is: 1 (strongly disagree), 2 (disagree), 3 (neutral), 4 (agree), 5 (strongly agree)
Question 25: Rank the statement “I am good at organizing” Range is: 1 (strongly disagree), 2 (disagree), 3 (neutral), 4 (agree), 5 (strongly agree)
Question 26: Rank the statement “I am good at research” Range is: 1 (strongly disagree), 2 (disagree), 3 (neutral), 4 (agree), 5 (strongly agree)

Questions 24-28 are about what different skills the student has. Their preferred range for these qualifications show what skills they are looking for in their team members.

Question 27, about organizational skill, may relate to someone’s ability to lead a team, but it could just as easily tell us about their ability to remember deadlines and details from class, arrange meetings, and facilitate the group working and communicating as a team. The other leadership-relevant qualifications we have are more about whether someone prefers the role of leader.

Question 24, about programming skill, may have some overlap with the previous questions about grades in programming courses, but this one tells us about their programming skill in general, as opposed to just how well they did in a course involving C or C++. This question, however, has the disadvantage of being self-rated. Perhaps an average could be taken over question 24 and the programming grades questions, to give greater accuracy.

Part 7: General questions

After we completed the initial 6 parts of the profile, we realized that there were a few general questions that did not fit in anywhere, but which we still thought were important. These are efficiency and workload.

Some people spend a lot of time on things that shouldn't take too long, often because they want to do things in the "best" or most interesting way possible. We should stick them with more practical and efficient teammates who can rein them in. This is especially important in computer science, which tends to be full of strongly detail-oriented people.

Question 27: How often do you finish something only to realize that some of what you did was unnecessary, or you could have saved a lot of time by doing things a simpler way? Range is 1-5: 1 (never), 2 (rarely), 3 (sometimes), 4 (often), 5 (always) Note that a 1 means they are efficient, and a 5 means they are inefficient.

Question 20 is about efficiency. Sometimes two people will get the same grade on an assignment, but one took 1 hour and the other took 5. The difference is that some people are better at finding the fastest, most efficient way to get things done, and others may either not immediately see how to approach a problem, or they take their time to master the material and get everything perfect. Either way, in a project it is important to balance out the people who take longer with the people who are good at finding efficient ways to get things done. The preferred range shows what a person is looking for in their teammates.

Question 28: How heavy a workload do you have this semester? Range is 1-5: 1 (very light; less than 3 courses, no part time job), 2 (light), 3 (average), 4 (heavy), 5 (very heavy; 5 or more courses and a part time job)

Question 5 is about workload. Some people are only taking a few courses, and others have not just 5 courses, but also a full time job. While it is true that even very busy people may be good enough at managing their time to get everything done, as a general rule we should be careful not to put too many students togethers who have a heavy workload; those teams are less likely to do well. On the other hand, we need to be careful not to always pair the busy students with the ones who have a lot of free time. It is good for a team to have people who can afford to pick up the slack once in awhile, but when that happens too often, it results in unfair expectations on the less busy students to do an unequal share of the work.

The preferred range shows how tolerant the student is of busy (or less busy) project partners. Perhaps someone will say say are very busy and only want students who have the most free time, and in that case the algorithm will have to just recognize that it is difficult to place them in a team that meets their requirements while being fair to everyone else, and they will end up being placed in a group that if the closest fit to what they want while still satisfying all the other criteria. A lot of the rules will be like that, and they should behave well in edge cases.

Rules

With every qualification, there are some common-sense and not-so common-sense ways of using them to inform the team selection algorithm. These are the *rules*. A rule is the algorithm's way of telling how well two students would work together, or how well a team would work together, or how well a student would work with a team.
The output of a rule is a compatibility rating, which is a number. The algorithm will try to choose the overall selection of teams that maximizes the compatibility score across all teams. A positive compatibility score for a team is good, and a negative compatibility score is bad. The algorithm may include optimizations to avoid evaluating possibilities that include a team with a negative score on one or more rules, so long as it still has other possibilities to evaluate where all the rules return a positive compatibility score.

The 28 qualifications have different rules, though there is a lot of overlap between them. Many of the qualifications are evaluated in the same way, and so we reuse the same rule.

Finally, some rules are more important than others, and all rules need some sort of normalization, since they may output a vastly different range of values that are unrelated to their overall importance to the algorithm. For this reason later parts of the algorithm will normalize and weight the output of all the different rules before deciding on the total compatibility score for a team, or a set of teams. The weights, however, may be better off decided during the implementation stage, once it is possible to test and refine the algorithm based on large sets of test data. Until then, our best guess for sample weights and normalization values will be given.

We have six main rules:

- Basic similarity rule
- Basic complement rule
- Custom time flexibility rule
- Custom efficiency rule
- Custom workload rule

These rules will be explained in detail later on. In the meantime, the following table shows which qualification is handled by which rule. Note that the basic complement rule works on pairs of qualifications, the table indicates which ones go together.

Table 1: Mapping between qualifications and rules

Qualification	Rule
1: Desired grade	Basic similarity rule
2: Start late or early	Basic similarity rule

3: Quality standards	Basic similarity rule
4: GPA	Basic similarity rule
5: Grade in 2401	Basic similarity rule
6: Grade in 2404	Basic similarity rule
7: Work over break	Basic similarity rule
8: Morning person	Basic similarity rule
9: Night owl	Basic similarity rule
10: Schedule flexibility	Custom time flexibility rule
11: Leader role	Basic complement rule (Pair A)
12: Follower role	Basic complement rule (Pair A)
13: Detail-oriented role	Basic complement rule (Pair B)
14: Big picture role	Basic complement rule (Pair B)
15: Challenge role	Basic complement rule (Pair C)
16: Repetitive role	Basic complement rule (Pair C)
17: Idea generation role	Basic complement rule (Pair D)
18: Idea evaluation role	Basic complement rule (Pair D)
19: Bluntness	Basic similarity rule
20: Amount of communication	Basic similarity rule
21: Amount of structure	Basic similarity rule
22: Programming skill	Skill complement rule
23: Documentation skill	Skill complement rule
24: Presentation skill	Skill complement rule
25: Organizing skill	Skill complement rule
26: Research skill	Skill complement rule
27: Efficiency	Custom efficiency rule
28: Workload	Custom workload rule

Types of rules

Basic similarity rule

The basic similarity rule handles qualifications where we want to group people with similar answers together, and we want to avoid pairing a student with someone whose answer falls outside their preferred range.

This rule compares one student's answer against another to come up with a compatibility metric for those two students. The compatibility metric for a potential team will be the sum of this rule's output for all pairs of students in a team, with some normalization.

Applicability

The basic similarity rule applies to 12 of the 28 qualifications, listed in the table below.

Table 2: Qualifications that use the basic similarity rule

Qualification	Importance
1: Desired grade	Very high
2: Start late or early	Very high
3: Quality standards	Very high
4: GPA	High
5: Grade in 2401	High
6: Grade in 2404	High
7: Work over break	High
8: Morning person	Moderate
9: Night owl	Moderate
19: Bluntness	Moderate
20: Amount of communication	Moderate
21: Amount of structure	Moderate

From the table it should be clear why the basic similarity is so important to the algorithm. Perhaps an altered version could be designed for the questions of only moderate importance, since the basic similarity rule was meant to be applied to cases where a good match is critical. However, in addition to the properties listed, the basic similarity rule has a few subtler properties that were determined through careful analysis and verified through a

simple test implementation. Because of those subtler properties we have reason to believe that the function will be appropriate for only moderately important qualifications, provided the overall algorithm applies a lower weight to the corresponding rules.

Algorithm

The following table gives pseudocode for computing the basic similarity metric between two students (for a given qualification). The Question object contains information about the range of possible values the qualification answers can take.

Table 3: Pseudocode for basic similarity metric

```
// Gives you the score for how well b is a match for a.
// Not necessarily the same as the score for how well
// a matches b.
double similarityMetric(Question q, Qualification a,
                        Qualification b) {
    double range = q.max - q.min + 1;
    double aInflexibility = range - (a.max - a.min);
    int distance = a.value < b.value ? b.value - a.value :
a.value - b.value;
    if (b.value < a.min) {
        int pdistance = a.min - b.value;
        return (1 + aInflexibility/range)*-(distance +
pdistance)/range;
    } else if (b.value > a.max) {
        int pdistance = b.value - a.max;
        return (1 + aInflexibility/range)*-(distance +
pdistance)/range;
    } else {
        return (1 + aInflexibility/range)*(range -
distance)/range;
    }
}
```

You can see that there are three output cases: other student's value is below the preferred range, other student's value is above the preferred range, and other student's value is within the preferred range.

Properties

The compatibility rating output by the similarity function behaves according to four properties: similarity, inflexibility, preference, and negativity bias. This section will define each of those properties and describe how they are implemented in the function and why they are desirable properties for the function to have.

Similarity: The closer their values are to each other, the better the match.

If the other student's value is above or below the preferred range, we see how far it is below the first range, and how far it is from the first student's value, and we take the sum of those as the similarity score. This will be negative, indicating a bad match.

If the other student's value is within the preferred range, we see how close it is to the first student's value. The closer it is the better the match.

Inflexibility: The seriousness of a bad match increases the more inflexible someone is. This is because if someone is flexible enough to accept a wide variety of teammates, they are more likely to get along with someone who falls outside their preferred range than someone who is intolerant of difference.

The benefit of a good match increases the more inflexible someone is. This is because all else being equal, if we have a person who could match two people based on a qualification, we would prefer to match him with the one who has narrower preferences, since that person is less likely to find as good a match again.

We achieve the inflexibility property by computing `aInflexibility = range - (a.max - a.min)` and working it into the results in the three possible output cases. The inflexibility factor is calculated as `(1 + aInflexibility/range)`, which means that this has a maximum of 2 when the student is as inflexible as possible, and a minimum of close to 1. So it's twice as important to find a match for the most inflexible student as it is for the most flexible student.

Preference: The algorithm encourages matches within the user's preferences and discourages matches outside of them. Any match within the range is better than any match outside the range.

We achieve this by separating the output into cases where the student's value falls outside the range (this will be a negative number) and cases where the student's value falls within the range (this will be a positive number).

The reason we want to separate the output into positive and negative values is because the user's preference is of the utmost importance when evaluating the qualifications to which this rule applies. We don't want to say, "Well, we know you didn't want to work with someone who likes to start work at the last minute, but it's still a decent match because the obscure logic of our utility function says so." If it isn't possible to match the student with someone who fits within the preferences, the algorithm will still be able to tell the difference between a bad match and a very bad match, but the important thing is seeing if something is a bad match at all is as simple as checking the sign on the rule's return value.

Negativity bias: Avoiding bad matches is more important than getting the perfect good match.

The qualifications to which the basic similarity rule is applied tend to be some of the most important. For this reason we strongly prefer to avoid matching students with people outside their preferred ranges, since such incompatibilities are likely to destroy a team. Though really, this is more of an adjustment to the previously described Preference property of the function.

We implement this by, when looking at cases where a student's value falls outside the preferred range, adding the distance from the student's value to the closest preferred value plus the distance from the student's value to the other student's value. This means that, when you look at the function, the similarity metric has a worst case value of `-2*(range-1)` when the user is outside the preferred range, as opposed to a best case of `range` when the student is inside the preferred range. Note that range refers to the scale of the whole question, whereas preferred range is the set of answers the student will accept.

Basic complement rule

As with the basic similarity rule, the basic complement rule is applied to several different qualifications, but with slightly different weights and normalization values that will be determined during implementation. The difference is that the similarity rule operates on a single qualification, and the complement rule operates on pairs of qualifications. The similarity rule tries to group similar people together, and takes preferences strongly into account, while the complement rule groups dissimilar people together and takes preferences less strongly into account.

Applicability

The basic complement rule applies to 8 of the 28 qualifications, listed in the table below.

Table 4: Qualifications using the basic complement rule

Qualifications	Importance
11: Leader role 12: Follower role	High
13: Detail-oriented role 14: Big picture role	Moderate
15: Challenge role 16: Repetitive role	Moderate
17: Idea generation role 18: Idea evaluation role	Moderate

Of the four complementary qualification pairs the algorithm considers, the leader/follower one is the most important. This will be reflected in the overall evaluation function, where a higher weight is assigned to the output of that rule.

Algorithm

The algorithm output is based on two factors: how dissimilar the student's value for one question is from the team's average (and maximum) values for the other question. We decided to do it this way because the more dissimilar two students are, the more complementary their roles will be. But if we stick a person on a team where on average the team members are not what the student wants, there will be conflict. There will also be conflict between the student and the team member who is most different from them, if the person who is most different falls outside their preferred range. The math takes all of this into account.

The following table shows the pseudocode for one half of the comparison. We have labelled the general qualification pairs as the *left* and the *right*. First you would call the function to compare a student's left to the team's right, then your would call the function to compare the student's right to the team's left. Take the sum of those, possibly adjust with weights and normalization, and that says how well the student fits into the team. Do the same computations for every student in the team, adjust with more weights and normalization, and that is the compatibility score for the rule given the team.

Table 5: Pseudocode for basic complement rule

```
// Have to repeat with left and right swapped, since this
// is definitely not a symmetric function.
double complementRule(Question q, Qualification aLeftAnswer,
                      Qualification aRightAnswer,
                      vector<Qualification> teamRightAnswers) {
    int dissimilaritySum = 0;

    // Calculate team average and max answers for right
    double averageRight = 0;
    int maxLead = 0;
    for (int i = 0; i < teamRightAnswers.size(); i++) {
        dissimilaritySum += min(aLeftAnswer.value,
teamRightAnswers[i].value);
        if (teamRightAnswers[i].value > maxRight) maxRight =
teamRightAnswers[i].value;
        averageRight += teamRightAnswers[i].value;
    }
    averageRight = averageRight / teamRightAnswers.size();

    // Result will be sum of dissimilarity value plus
    // some kind of preference value.

    int range = q.max - q.min + 1;
    int inflexibility = range - (aRightAnswer.max -
aLeaderAnswer.min + 1); // # values they won't accept
```

```
    double inflexibilityFactor = 1 + inflexibility/range;
    double averageRightFactor = 0;
    double maxRightFactor = 0;

    // Is the team average inside or outside the preferred range
    if (averageRight < aRightAnswer.min) {
        averageRightFactor = aRightAnswer.min - averageRight;
    } else if (averageRight > aRightAnswer.max) {
        averageRightFactor = averageRight - aRightAnswer.max;
    }
    if (maxRight < aRightAnswer.min) {
        maxRightFactor = aRightAnswer.min - maxRight;
    } else if (maxRight > aRightAnswer.max) {
        maxRightFactor = maxRight - aRightAnswer.max;
    }

    // Preference penalty depends on how inflexible the person is
    double preferencePenalty = inflexibilityFactor *
maxRightFactor * averageRightFactor;

    // Rule output depends on how dissimilar they are
    // minus some adjustment based on whether the team falls
    // within the person's preferred range.
    return dissimilaritySum - preferencePenalty;
}
```

Custom time flexibility rule

Question 10 asks about how flexible the user is with their time. As explained previously, we want to avoid putting too many inflexible people together, and we'd like to balance out inflexible people with flexible people. However, there is no reason not to place several flexible people together, so simply calculating the difference between two students' values would not make a good rule. Instead we designed a custom rule based on a table, which shows the output of the function for each pair of values. The columns (top labels) show the student's value, and the rows (right labels) show the group average.

Table 6: Custom time flexibility function

1 (most inflexible)	**2**	**3**	**4**	**5 (most flexible)**	
4	2	0	0	0	**5 average**
3	1	0	0	0	**4 average**
2	0	0	0	0	**3 average**

-1	-1	0	1	2	**2 average**
-1	-1	2	3	4	**1 average**

According to the output table, we penalize teams that put inflexible students in inflexible groups, and we prefer teams that put flexible students in inflexible groups. We have no preference regarding adding flexible students to flexible groups.

We decided to make the comparison based on a student and the group's average because this speeds up computation and still provides an accurate answer. Adding a flexible student to a group that is on average, inflexible, is probably a good thing. At least it's better than adding another inflexible student.

To compare two students as opposed to a student and a team, the rule can be applied with the other student's value instead of the team average. The overall compatibility score for a team is the sum of the rule's output for each student compared to the team average.

To take the user's preference into account, with each student we add some number to the compatibility score if the group average falls within the preferred range and subtract some number if the group average falls outside the preferred range. However, the user preference is not as important here as it is which some of the other qualifications, such as quality standards, so the adjustment serves more as a tiebreaker than anything else. The actual preference adjustment value depends on normalization factors that will be determined in implementation. A guess for the adjustment value is 1 if within the preferred range, else -1.

Custom efficiency rule

Question 27 is about efficiency. Even when you look at students who got the same grade on something, you have some who spent a lot more time on it than others. We think that it is a good idea to mix the less efficient students with the more efficient students.

On the one hand, it could annoy the more efficient students to have to help their slower teammates find faster ways to do things or avoid unnecessary work, but keep in mind that this rule will most often be used to distinguish between possible teams of students who tend to get the same grades. So they are unlikely to hate each other, and they both have the same general level of competence.

Additionally, when the efficient students help their inefficient team members find better ways of doing things and avoid unnecessary work, the group grades as a whole will tend to go up. We believe that the benefit to grades of mixing efficient and inefficient students will outweigh the minor increase in conflict.

The following table shows the output of the function based on any pair of inputs. It works the same way as the custom time flexibility table. The top shows the student's answer, and the right shows the group average.

Table 7: Custom efficiency function

5 (most inefficient)	**4**	**3**	**2**	**1 (most efficient)**	
3	2	0	0	0	**1 average**
2	1	0	0	0	**2 average**
1	1	0	0	0	**3 average**
-1	-1	1	1	2	**4 average**
-2	-1	1	2	3	**5 average**

We avoid adding inefficient people to what is on average an inefficient team, and we prefer adding efficient people to inefficient teams. There is no special benefit to adding an efficient person to an efficient team.

As with the custom time flexibility rule, for each person we add 1 if the group average falls within their preferred range and subtract 1 if the group average falls outside the preferred range.

We could have used the same rule for efficiency and time flexibility, but we wanted the option of providing different output values for different combinations.

Custom workload rule

Question 28 asks about workload. We want to avoid teams where everyone has a very heavy workload. It is good when a team has people who can pick up the slack when the others are busy, but we do not want to pair the busiest people with the least busy people, since such combinations may lead to disrespect, a sense of entitlement on the part of the most busy team members, and situations where the least busy people end up doing far more than their share of the work.

The following table shows the output of the function based on all possible pairs of inputs. This works more or less the same way as the other custom workload functions. The columns depend on the student's own workload, and the rows depend on the team average workload.

Table 8: Custom workload function

5 (very heavy)	**4**	**3**	**2**	**1 (very light)**	
-1	-1	0	0	0	**1 average**
0	1	0	0	0	**2 average**

2	1	0	0	0	**3 average**
-1	-1	1	1	-1	**4 average**
-2	-1	2	0	-1	**5 average**

We prefer matching busy people with slightly less busy teams, and we prefer matching slightly less busy people with slightly more busy teams. We avoid matching busy people with busy teams or very busy people with people who are not busy at all.

The student preferences are taken into account as with the other custom workload functions. For each person, add 1 if the group average falls within their preferred range, subtract 1 otherwise.

Skillset complement rule

Any computer science team project is likely to involve a number of different skills beyond just programming. For this reason we try to create teams where every required skill has someone who is good at it. We also try to group people in teams where everyone has more or less the same overall skill level, so nobody feels taken advantage of.

Applicability

This rule applies to questions 22-26 together. The following table gives some idea of the relative weighting.

Table 9: Skillset weighting

22: Programming skill	Moderate
23: Documentation skill	Low
24: Presentation skill	Low
25: Organizing skill	Low
26: Research skill	Low

Almost all computer science team projects will involve programming, so we place more important on that skill than any of the others. However, programming skill is already covered in some of the previous questions (grades in 2401 and 2404), so we only place a moderate emphasis on it here. We don't know which of the other skills will be involved in any given project in advance, so we place a low importance on each of the other individual skills.

Algorithm

The rule will tell you how good a match a person is for a team. To find the overall compatibility score for an entire team, you have to run the rule for every person in it and take

the sum. Comparisons are made between the individual and the team averages, so you don't have to compare every pair of students in the team.

There is no pseudocode this time, but the algorithm is roughly as follows.

The output of the rule is a function of three things, the *similarity adjustment*, the *complement adjustment*, and the *preference adjustment*.

Note that #students refers to the number of students in the team.

The *similarity adjustment* defines how far each student is from the team average, and it is based on overall scores. To get the similarity adjustment, first you calculate the *overall error*, as follows.

Overall error: The overall average for a student or a team is calculated as `2*programming skill + documentation skill + presentation skill + organizing skill + research skill`. Then you take the sum for each student of the absolute difference between the team overall average and the student's overall average. That's basically some kind of error function, calculating how far each student is from the team average. The maximum value is `2*#students`, which you would get with a team average of 3 and every person in the team either a 1 or a 5.

The *similarity adjustment*, then, is `2*#students - overall error`. The closer each student is to the team average, the lower the error will be (minimum 0), so the similarity adjustment ranges from `2*#students` for the best team and 0 for the worst.

The *complement adjustment* gives the highest score to a team where every skill has someone in the team with a 5 in it, and the lowest score to a team where every skill only has people with 1s in it.

The *complement adjustment* is calculated by first for every skill, finding the person with the highest value in it, then adding up the max values (programming skill counts twice). The complement adjustment therefore ranges from 0 to 30, because the highest value in each skill is a 5, there are 5 skills, and the programming skill counts twice. It is extra important that each team has someone who is a 5 in programming skill.

The *preference adjustment* rewards teams where the team average is within a student's preferred range for each skill. It is calculated by first finding the team average for each skill, then: (if the average is within the preferred range) adding 1 or (if the average is outside the preferred range) subtracting one. The lowest value is -#students*5, and the highest value is #students*5.

The overall output of the function will be some linear combination of the similarity adjustment, the complement adjustment, and the preference adjustment. The exact weights will be determined in implementation once we can test the design.

Algorithm

This section of the document describes how the algorithm uses the rules to determine the best partitioning of students into teams.

Checking team size

First, the algorithm needs to see if the project settings for team size are appropriate given the number of students in a project. If you need teams of size 7-8, and there are only 10 people in the class, the algorithm will have to make a choice. It cannot fulfill the requirements, so it must adjust the parameter values. In this case it will decide that a team of 10 is the lesser evil, since that way the total error is minimized. It would calculate total error for a team of 10 as 2 because there would be one team 2 people over the limits, and it would calculate the total error for 2 teams of 5 as (-2 + -2), since there are two teams two under the limits.

The algorithm would then run with actual team size settings of 5-8, and presumably the interface's output would notify the user of this change. The actual project settings in the database would not be modified.

Running time to calculate all possible team compatibility scores

Note that we probably will not calculate all possible team compatibility scores as the first step in the algorithm. This is just an analysis to show how an individual team compatibility score is computed and how long it might take.

Just some quick math to appreciate the scope of the problem: if you have 100 students in a project and you want teams of 4, there are C(100,4) possibilities = 100*99*98*97 = about 100 million. And there is a fair amount of calculation to find the compatibility rating for a given team.

However, there are a few ways we can reduce that running time. First, the basic similarity rule, which covers 12 of the 28 qualifications, only depends on the answers of the two students being compared. So we can precompute that for every pair of students in only C(100,2) * 12 * 2 = 100*99 = about 240,000 steps. The 12 is because there are 12 qualifications, and the 2 is because the function is not symmetric.

The custom efficiency, custom time flexibility, and custom workload rules compare a student to an entire team, instead of a student to each person in the team. This means that in a team of size 4, these three rules only need to be run 4 times, as opposed to C(4,2). This grows much more slowly as team size increases.

Similarly to the 3 custom rules, the skillset complement rule is linear in the team size since it compares each student only to the team average and maximum. The 8 complementary rules

also are linear in the team size for the same reason; it only compares the student's answers to the team maximum and average.

So the overall running time to calculate the compatibility score of all possible teams would be on the order of O(C(#students,teamsize)*teamsize).

Note that we are ignoring in this analysis the weighting/normalization for each rule, since calculating some linear combination of a size 28 vector is not computationally difficult at all. And really, the compatibility score for a team will be some kind of linear combination of the 28 rules, since not every rule is as important overall. The exact weights will be determined during implementation, but generally they will follow the level of importance indicated in the applicability sections in the previous part of the document.

Compatibility score for a partition

Unfortunately, the job is not over once all the score for all possible teams has been computed. We still need to find the optimal partition. A partition is a division of a set (all the students in the project) completely into non-overlapping subsets (teams). We define the **compatibility score of a partition** as the sum of the compatibility score of all of its teams. With that, we now have a function to optimize.

There are many algorithms for finding a solution based on some optimization function in a combinatorially large search space. This is exactly what we are dealing with here.

We considered many different possibilities for the overall search algorithm, including k-nearest-neighbor clustering, attempts to reduce to known NP-complete or NP-hard graph problems, but none of them exactly fit our needs.

We could have chosen an exhaustive search of all possible arrangements of teams, but that is actually very large. Must larger than C(100,4). It's more like 100!/(4!)^25, the same as the number of ways to rearrange the letters of AAAABBBB….(for 25 letters).

Instead what we settled on was a **backtracking algorithm**.

Backtracking algorithm

Backtracking algorithms are a general solution to constraint satisfaction problems and are usually faster than doing an exhaustive search over all solutions. This is because a backtracking algorithm allows you to omit large portions of the search space by eliminating them when they fail some test.

The backtracking algorithm is built on six functions.

root(P): The initial partial partition returned by root(P) will just be the empty set, where nothing has been partitioned.

reject(P,c): We can reject a partial solution candidate (a team we are considering adding to our final partition) if the team compatibility rating is a negative number.

accept(P,c): We can reject a partial solution candidate (a team we are considering adding to our final partition) if the team compatibility rating is a positive number.

first(P,c): we prefer a best-first search as much as possible, so our quick heuristic for guessing the best match is to pick a team with a high compatibility score.

next(P,c): follows similar logic for first(P,c), only returns the next element. We may store the possible selections of candidate teams as some kind of priority queue, but that is an implementation detail.

output(P,c): once we have a complete partitioning of the students in the project into teams, we are done.