Technological Forecasting & Social Change 176 (2022) 121461

Contents lists available at ScienceDirect

Technological Forecasting & Social Change

journal homepage: www.elsevier.com/locate/techfore

Dynamic group formation in an online social network

Reshawn Ramjattan [a,*], Nicholas Hosein [b], Patrick Hosein [a], Andre Knoesen [b]

[a] *The University of the West Indies, St. Augustine, Trinidad*
[b] *University of California, Davis, CA 95616, USA*

ARTICLE INFO

Keywords:
Online groups
Recruitment
Matchmaking
Network flow

ABSTRACT

For those seeking to recruit teammates for a specific purpose, like a project or study group, challenges quickly arise once they have exhausted their social circle. In the wake of the current pandemic, meeting new people that are right for a specific team is even more difficult than before due to the lack of in-person events. On social media platforms, users often have large networks of connections but have very few close personal relationships within them. This makes it difficult to find compatible people that share the same goal, and are interested in niche groups on those platforms. We present a scalable framework for establishing small online groups that balance two objectives, making the best group recommendations to users and guiding group hosts to the best users for their group. We illustrate this framework using three use cases. Lastly, we evaluate a serverless implementation using a large social media dataset to simulate a production environment and compare our framework to a network flow approach to solving the problem.

1. Introduction

The ongoing pandemic has forced many events and gatherings to be cancelled or moved online. However, many social aspects of in-person gatherings, such as networking, cannot be easily facilitated by digital media. Attending relevant events, classes or conferences is a frequent approach to building a network of compatible and like-minded people. So without real chances to network, assembling teams with a common purpose becomes a challenging social task. For example, finding suitable co-founders for start-up ventures or potential employees based on a casual conversation at mixers or academic events, or establishing rapport with classmates during practical exercises to form study groups. These are situations where physical gatherings allow for personal connections as a by-product of its main purpose and can greatly help with meeting new people if you need a group.

While online platforms can create an acceptable stage for the main event, they do not provide the same level of interpersonal possibility. We consider the problem of a user seeking to form a small purposeful group of the most suitable and compatible people. On existing social networks, users often have a large number of friends but very few of them are close personal relationships. The large network of loose and estranged relationships is not very useful for finding group members that share the same interest, purpose and timing. Research done by Ramjattan et al. (2020) shows prior work in this area as well.

Consider a student forming a study group. Studies by Dolmans and Schmidt (2006) and Springer et al. (1999) show that there are several cognitive and motivational benefits to small group learning. There are also benefits to presentation, communication and team-building skills. Study groups allow students to take responsibility and benefit from small group learning outside the classroom. Without preexisting relationships, forming a study group in a class is close to forming a random assortment.

This randomness results in a variety of personality types, learning styles and interests among the members. This variety can negatively affect not only the group's compatibility, and therefore comfort in engaging discussion, but also the effectiveness of the small group learning since its members learn best in different ways. Even if the students do not know their best learning method and style, work by Chamorro-Premuzic et al. (2007) shows the correlation between personality and preference for learning methods.

By creating an online means to form these groups we can allow students to find a group with high compatibility both as people and as learners. It allows us to consider a wider set of potential group members and some key variables such as learning style, personality traits McCrae and John (1992), subject comfort and general common interests. We can then form the ideal online study group efficiently. For example, a

* Corresponding author.
E-mail addresses: reshawn.ramjattan@my.uwi.edu (R. Ramjattan), nhosein@ucdavis.edu (N. Hosein), patrick.hosein@sta.uwi.edu (P. Hosein), aknoesen@ucdavis.edu (A. Knoesen).

https://doi.org/10.1016/j.techfore.2021.121461
Received 10 February 2021; Received in revised form 24 December 2021; Accepted 27 December 2021
Available online 31 December 2021

German student in an English speaking University can find a study group with similar interests and the same native language.

While we just used educational groups for illustration, similar arguments can be made for many other group formation scenarios like recruitment, gaming teams and social clubs. The issue that needs to be addressed is the conflicting needs of the person creating the group and those being presented with groups to join. The seekers want to see groups that are closest to their preferences. However, if they apply to those, there might be users even closer to the host's target attributes. The outcome is that the group host has a large number of applications to review and the seeker is likely to be rejected from the group in comparison to better fits. Therefore the problem is finding the right combination of load balancing and matchmaking to meet both needs. We must balance the happiness of group hosts and group seekers, by showing seekers groups they are likely to get into and enjoy while helping hosts find the most suitable users for their group in the least applications.

Furthermore, as a live application, the set of users and groups would be constantly changing as people join, create new groups and close full groups. The initial solution to finding the best distribution of group recommendations would soon be invalid. If a new solution takes too long to produce, the end-user experience would vary depending on when the last solution was found. Hence, considering scalability and delay in solving the problem is also important.

We present a framework for establishing small groups that solve the problem described. We focus on scalability through an event-driven architecture and a fault-correction style approach to maintaining an optimal system state. Illustrative scenarios are used to describe the usefulness of the framework, which was implemented using serverless cloud infrastructure. We evaluate the implementation by using social media data to simulate a production environment with a large number of users. Finally, we formulate the challenge as a network flow problem and carry out a method to calculate the optimal result given a set of users and groups. By doing this, we observed our framework's sacrifice in quality for its timeliness by comparing its results to the optimal one.

2. Literature review

The group recommending problem described can be viewed as an extension to the matchmaking algorithm of a dating application. Work done in Hitsch et al. (2010), Brozovsky and Petricek (2007), Li et al. (2019), Tu et al. (2014), Mackinnon (2015), mal (2020) describe attempts to find solutions to the one-to-one matchmaking problem. The methods include using the Gale-Shapley algorithm Dubins and Freedman (1981) as well as recommender systems and a combination of techniques for producing additional user attributes. These methods accomplish a means of determining high-quality two-sided matching. However, the use case does not need to consider the load of matches or applications faced by a user as much as it has to focus on producing the most likely matches. Nor do they explicitly address the over-time scalability of the approaches used.

Online gaming lobby creation is a popular group focused matchmaking use case, so their approaches are worth considering as well. In Boroń et al. (2020), Agarwal and Lorch (2009), Myślak and Deja (2014), Manweiler et al. (2011), techniques used to form balanced and well-matched groups, while addressing the time constraints of their use case, are described. The methods place users in groups that best match their case-specific attributes but these applications are shaped by the time-sensitive nature of their purpose. To ensure users spend little time waiting to play, lobby creation is done automatically so solutions are not as concerned with the suitability constraints of our described problem. In trying to emulate the social activity of forming a small meaningful group, granting both hosts and group seekers control and choice is paramount. When managing the load of applications bared by group hosts, the problem expands out of only matchmaking and into the area of load balancing applications for the host's review.

Load balancing algorithms receive user requests as input and distribute those requests among available resources based on their capacity. Variations of such algorithms including FIFO, fair scheduling and capacity scheduling are presented by Ghomi et al. (2017) and Zaharia (2009). These are strong solutions to load balancing problems in general, however, they fail to account for two of the main concerns of our described problem. Once an allocation is made, these methods do not address maintaining a steady-state system while the sets of groups, users and their attributes change over time. Nor do they concentrate on matching attributes that can shape the way the load is distributed.

When considering our set of users, groups and the constraints that limit the recommendations of groups to users, the problem can also be viewed as a network flow assignment problem. Work in Bertsekas (1992), Kuhn (1955), Gelareh et al. (2016), Bertsekas (1985) describe solutions to this type of problem using auction, Hungarian and branch and cut algorithms. These methods can find the best resulting recommendations from a given network. But, in our described problem the given set of users and groups are almost constantly changing. Running such an algorithm on a large network so frequently would be expensive and infeasible. Hence we must explore solutions that can maintain a steady solution as the network changes while producing good quality recommendations.

Maintaining a steady state can be perceived as consistently moving towards an optimal solution in the face of sub-optimal changes. Simulated Annealing (SA) Van Laarhoven and Aarts (1987), uses the analogy of the annealing of solids to solve optimization problems. At each stage of the SA algorithm, with some probability, one stays in the present state or moves to a new state. This results in eventually moving to lower states of energy, gradually approaching an approximate global optimum Delahaye et al. (2019).

For a system with many users and many groups, finding the optimal solution for the best recommendation of users to small groups is a computationally intensive task. One possible approach is overnight batch processing that re-stabilizes the system. However, this is still a huge computation and not the most timely solution. Works by Attiya et al. (2020) and Saadatpour et al. (2019) show cases of SA's use in load allocation problems. While none of these is a complete solution to our problem, they present evidence that a scalable approach to solving the problem described and maintaining a near-ideal state over time can be achieved.

3. Methodology

Our network comprises of users seeking groups and groups created by users/hosts, both with a set of preferences or attributes. Users can see a small number of recommended groups and choose to apply to any of those. Once they apply, the group host can accept or deny them. Whether their group application is accepted or denied, users can continue to apply to other groups since a new group will be suggested to them. Our goal through these recommendations is to help users find the best fitting group for their needs and help group hosts find the best users for theirs.

This network would be constantly changing as users join and leave while groups get created and closed. Therefore our problem is not only finding the optimal network of users and recommended groups but finding and maintaining such a system over time. Processing the entire network continuously is expensive and infeasible, so we use a set of algorithms to break down the network and focus on points of the most significant change. By doing this, we lose some matchmaking quality but can maintain a steady system in real-time. We define the goals of our framework as follows.

- **Balancing Happiness** - Users and group hosts have conflicting needs. Users want the best group for them but they may not be the best fit, while the group hosts also want the best fitting members. So we must find a balance that allows users to see the best fitting groups they are likely to get accepted to.

- **Ease of Use** - This solution can be applied to several different areas where quality-focused matchmaking is needed. Considerations must be made for the framework to be re-applied with little new effort.
- **Real-time Stability** - One of its key attributes is maintaining a close-to-best state at all times rather than periodically optimizing to the best possible state.

3.1. Attributes

The attributes our framework uses are Preference Difference and Users per group. Preference Difference is an aggregation of suitability features that are decided per application. We chose to aggregate and abstract these features to a single attribute to enhance the reusability of the framework. Users per group is a measure of how many people are recommended a group, relative to that group's capacity. This is to help ensure a fair spread of users across groups. This is also called the alpha value, where u is the number of users shown a group, c is its capacity and $\alpha = \frac{u}{c}$.

3.1.1. Preference difference

Consider the example of companies recruiting new employees for a specific team. The preference difference would comprise of the differences between the user and team for several different recruitment specific features. So we must also define how we compare those features through cost functions. For illustration, suppose that our application considers years of experience and the number of matching skills as its features. We would define cost functions for measuring the difference between a user's experience level and the experience level the team manager is looking for. Similarly for the difference in skill sets, resulting in values for experience cost and skills cost respectively. The preference difference d_p our framework is concerned with would then be the average of experience cost (c_e) and skills costs (c_s) and so $d_p = \frac{c_e+c_s}{2}$.

Next, we derive the Experience Cost Function. We would want team members with experience closest to what the manager has defined as the target to have the lowest difference cost. An example function that could result in this would be c_e where the difference between the user's years of experience and the target number is e and so $c_e = \frac{e}{e+1}$. So a user with 1-year experience applying for a team with a target of 5 years would have a cost difference of $c_e(4) = 0.8$ while someone with 4 years experience would have a cost of $c_e(1) = 0.5$

The Skills Cost Function is derived as follows. In comparing skill sets, we may want to reward matching skills rather than punish mismatches in our difference cost function. So where c_s is skill cost, n is the total number of skills and s is the number of matching skills between the user and target team, our cost is $c_s = \frac{n-s}{s}$.

3.1.2. Users per group

The number of users recommended a group is important to consider because while we are only presenting options to users and not controlling who applies or gets accepted, we do not want to cause a small group to be overloaded with unnecessary applications. For example, imagine we have ten users and two created teams in our system where a user is shown only one recommendation. We can show five users the first team and the other five the second team. However, if one team has a capacity of ten while the other has a capacity of three, the number of recommendations per team is not fair. Hence the need for measuring users per group through α. Ideally, we would want all groups to have similar α values, making it an important parameter in the framework.

3.2. Framework

Our framework takes an event-driven approach to defining its algorithms. It facilitates a serverless architecture for implementation, easing the infrastructural demands and simplifying maintenance through relevant cloud platforms. We define our metrics, method for evaluating the state of a system and our objective function below. Then the 4 use cases that follow describe our means of satisfying that objective function. The use cases define the major sub-optimal events that can occur, as well as the rectifying action to be taken by its corresponding algorithm. A single function, called the Bubble Function, is used after these cases for optimizing the quality of recommendations.

3.2.1. Objective function

To measure the state of the entire system we defined a metric called system entropy. System entropy evaluates both the number of users seeing a recommended group and the preference difference across that system of matches. In its best possible state, this value would be zero. Since we strive for equal α values across all groups, we measure the spread of users per group by calculating the variance of α values. Where u is the number of users in a group, c is its capacity, α_i is the value of the ith group, $\widetilde{\alpha}$ is the mean of α, n is the number of groups and the variance of α, σ^2, is given by

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(\alpha_i - \widetilde{\alpha})^2 \tag{1}$$

For preference difference, we calculate the average difference in features, using their cost functions, between all users and their recommended groups. System entropy is then the sum of these two metrics. Resulting in the objective function of minimizing the system entropy, e, where A is the set of α values for all groups, d_i is the preference difference value of the ith user in the system and n is the total number of users.

$$\min e = \sigma^2(A) + \frac{1}{n}\sum_{i=1}^{n} d_i \tag{2}$$

3.2.2. Use cases

1. **New User:** As a new user joins they would need to be given groups they can choose to apply to based on their defined preferences. Initially, the user's recommendations are populated with the lowest α groups to help balance the spread of users per group. A function is then run to improve the quality of the user's recommendations without making changes to the number of recommendations or number of users seeing a group.
2. **New Group:**A new group takes the worst matching users from the groups with the highest α values. This rectifies the number of users per group. The bubble function is then run on each moved user to increase the quality of matches.
3. **Remove User:** When a user chooses to delete their account, they are removed from any groups they belong to. The α of groups they were shown will decrease, and be addressed when new users join.
4. **Close Group:** If a group is full or a host decides to close or delete their group, all users who were recommended this group now have an empty slot to be filled. Firstly, to address balancing users per group, these users are spread across the lowest α groups. Then the bubble function is run for each user to replace the once low α group with a better quality match.

3.2.3. Bubble function

The previous cases make considerations for balancing the number of users recommended each group, the group α values. This function is concerned with improving the quality of those recommendations. The bubble function seeks to swap out recommended groups with better quality matches. This is done by iterating through a user's recommendations and searching for swaps with users in other groups that would decrease the overall preference difference. We swap users in groups rather than simply changing groups to maintain the distribution of users across groups.

We take a greedy approach to finding the best swaps. For each group a user has, we search the entire set of other groups' members for the best swap. This process is simple and allows for the possibility of stopping after finding some good swaps. A swap is considered good if the total preference difference after the swap is lower than it was before. As the computation of this can become too heavy, one can filter the other groups that checked based on some important feature, for example, distance.

3.3. *Applications*

The events of the framework architecture remain generally the same across applications. Adapting to an application focuses on changing the definition of preference differences that measure the matching quality. The following examples illustrate this as well as the usefulness of the framework through different applications.

3.3.1. *Employee/Team recruitment*

Recruiting employees or members for a specific team, project or department is a time-consuming process. Our framework, in a case like this, could consider primary roles, skill sets and personal interest tags as cost functions or features. Some common personal interests can lead to the easy establishment of rapport and therefore cultural fit. An example of a cost function c_r, where diversity of roles is preferred, is as follows. $c_r = 1 - \frac{r_u}{n}$, where r_u is the number of unique preferred role values among users in a team and n is the total number of users in the group. This systematic approach can improve the quality of matching for small projects as well as improve the efficiency of pre-processing applications for larger businesses.

3.3.2. *Study groups*

For a university student, classmates might not be the best candidates to form a study group with. Learning styles and personality traits can vary greatly and can negatively affect the compatibility needed for a well-functioning study group. Online study group matchmaking increases the efficiency and likelihood of forming an ideal study group by widening the scope of candidates and considering key attributes. These key attributes also define our preference difference cost functions and include learning styles and personalities.

3.3.3. *Online gaming teams*

Depending on the nature of the online video game, users may play alone or need to form larger teams beyond their friend group. Ideal gaming teams can be created by devising cost functions around ensuring similar player skills, a variety of preferred roles, similar intentions, whether it's for competitive improvement or casual fun, and general interest tags for building team rapport.

3.4. *Implementation*

We implemented the framework using a set of cloud services and datasets that could simulate a network of users and groups. An android application was used to drive simulated user actions.

3.4.1. *Architecture*

The driving android app was developed using Flutter and supported events for our major use cases as well as over-time simulation of those events. Google Cloud Platform services were used for data storage and hosting function logic. Specifically, Cloud Functions were used for hosting serverless functions for each of the event use cases and functions described in the previous section. Firestore was used for storing user data sent from the application. Figure 2 illustrates the solution's system architecture.

3.4.2. *Datasets*

The Facebook100 Traud et al. (2011) dataset was used as a source of feature values for both users and groups. For example, the "Student Major" feature had the most normal distribution and was used to represent a simulated application feature in the later experiments. The dataset did not contain location data for users which were needed for measuring distance costs as an additional feature. We used the tweet-geolocation-5m Zubiaga et al. (2017) dataset and assigned location information to each pseudo-user. This resulted in a dataset of approximately 1.2 million users with subject comfort level and location attributes. The listings in Fig. 1 show summaries of the resulting data models for users and groups.

3.4.3. *Race conditions*

Within our described cases and algorithms, race conditions are common. For example, if two sub-optimal changes, like groups being created, occur at the same time and result in two bubble functions being run concurrently on two different users. This creates the possibility of those two users attempting to swap with the same person to get into that group, as illustrated in Fig. 3. To resolve this, our implementation uses Firestore Transactions to ensure atomic operations on the most up to date data whenever state-changing actions, such as swapping groups, are to be made.

3.4.4. *Bubble function subsets*

As mentioned previously, the bubble function uses a subset of the groups in the system to reduce the computational load. One of the design decisions made for our implementation was to use proximity as the basis for deciding the subset. Proximity was chosen because the similarity in culture and timezone as well as the potential for some future in-person communication was deemed to be important for many different group-forming scenarios. Further to this point, one of the features our implementation considered was the physical distance between a user and a group. This distance was calculated using the Haversine formula, also known as great circle distance, which is a method of measuring the distance between two pairs of coordinates. The distance cost function where c_d is cost, d is the distance between the user and group location, is

Listing 1: User Model

```
{
    String id,
    Array<float> features,
    Array<group> recommended groups,
    String longitude,
    String latitude

}
```

Listing 2: Group Model

```
{
    String id,
    Array<float> features,
    String status,
    Float alpha,
    Array<user> users seen by,
    Integer capacity,
    String longitude,
    String latitude

}
```

Fig. 1. Data types.

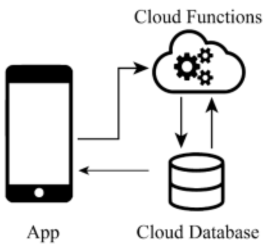

Fig. 2. System architecture.

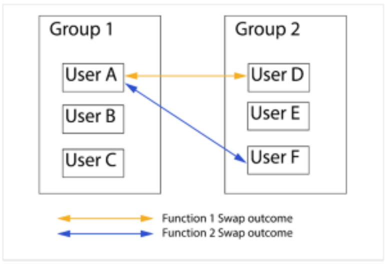

Fig. 3. Bubble function race condition.

given by $c_d = \frac{d}{d+1}$

The Haversine formula for producing d is given below where R is the radius of the Earth and lon_1, lon_2, lat_1, lat_2 are the coordinates of two points.

$$\Delta lon = lon_1 - lon_2$$

$$\Delta lat = lat_1 - lat_2$$

$$a = \sin^2\frac{\Delta lat}{2} + \cos\ lat_1\ \cos\ lat_2\ \sin^2\ \frac{\Delta lon}{2}$$

$$c = 2atan2\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = Rc$$

To produce the proximity subset, we fetch the x closest groups to the primary user of the function, where x is the size of the subset. Sorting by coordinates in queries is not supported by Firestore so an alternative process was used. The longitude and latitude float values are converted to 32-bit floating-point binary representations. Then the bits are interleaved in the order of most significant to the least. This 64-bit value is stored in the user or group's document. Consider three user documents sorted by this 64-bit value. The order is such that the first user is farthest from the third. That is, the documents closest on either side to any given document are the closest users to it. Therefore to fetch the x closest groups, we sort by the 64-bit location value and query for $\frac{x}{2}$ documents less than and greater than the primary user's location value.

3.4.5. *Evaluation*

We ran two main experiments to evaluate our approach. The first was to ensure the framework successfully improved our definition of the system state. While the second sought to show that improving our measures of system quality corresponds to forming better group recommendations.

The first experiment measured the system entropy throughout a simulation of events. Entropy represents the state of the system concerning our goal. As mentioned in Section 3.2.1, it's the combined value of the differences in preferences across all recommended groups to users and the number of users recommended each group concerning group capacity, also known as the α value. A sample of 100 users and 20 groups were created in a way that all groups had the same α value. Users and groups only held a single feature of interest, student major, for the sake of simpler calculation verification and simpler visualization of results. This feature was chosen because it had the most normal distribution among those in the dataset. Next, the bubble function was triggered on different users 350 times. For this setup only, the function was triggered manually since no sub-optimal changes will occur to trigger it automatically. This was tracked on every update for any group or user by an additional cloud function.

All traced entropy values were accompanied by a timestamp and the entropy over time was plotted, resulting in the graph shown in Fig. 6. During this period we also measured the average difference cost of users in a group for all groups and produced the plot shown in Fig. 7. The average difference cost is the average difference in preferences between users and each of the groups they were recommended. Figure 7 shows how this average changed over the duration of the experiment. This visualizes the use cases' impact of optimizing for the entire system on individual users.

Next, we adjusted the framework to run as intended for a real system, with the cases and functions set to run on triggering events. We also added new groups and users over time. This was done to test the method's ability to maintain a steady-state. The aforementioned entropy trace continued in the same way. The entropy over time is provided in Fig. 4. To show the impact of the framework's use case functions, we also tracked the α values of all groups during this period, the results are shown in Fig. 5. Since α is the ratio of users being recommended a group to that group's capacity, Fig. 5 shows the impact of our framework on the average visibility of groups over time as recommendations to users change.

The second experiment focused on testing whether or not our framework and definition of metrics resulted in improved group quality and recommendations. We repeated the simulations of experiment 1 but with 1000 users and 20 groups holding 50 users each. Before and after it was run, we plotted each user's value of the feature of interest, simulated using the "Student Major" feature, against their recommended group's corresponding value. As more points overlap they merge, and their shade of copper darkens as density increases. Figure 8 shows the initial state of users' simulated feature value against the value of the group they were being recommended. This state shows a random assortment of group recommendations to users and represents a poor system state. Figure 9 shows the final state after the framework's functions were run and optimized for the objective function.

3.5. *Network flow approach*

The major disadvantage of our approach versus a network flow approach is the possibility of falling into local optima. When the network flow solution is run, though infrequently, it would result in the optimal state of the system. Therefore, it was important to determine how much worse the solutions from our method were, on average. To that end, we defined our scenario as a network flow problem with the constraint below where d_{ij} is the flow cost from user to a group, the number of groups a user can belong to is predetermined and constant and the number of users per group is calculated beforehand such that the α variance is minimized. The network model is also illustrated in Fig. 10.

The network flow solution was developed in JULIA and modelled using JuMP, a modelling language for mathematical optimization embedded in Julia, and the Coin-or Branch and Cut (Cbc) optimizer. We then ran a series of experiments where a sequence of actions was simulated using our approach, while the network flow solution was run on the same system of users and groups. The average system entropy, as defined in the aforementioned evaluation section, was calculated and recorded for both solutions. Results are shown in Table 1.

4. Discussion

Figure 6 shows the result of the experiment to test the ability of the

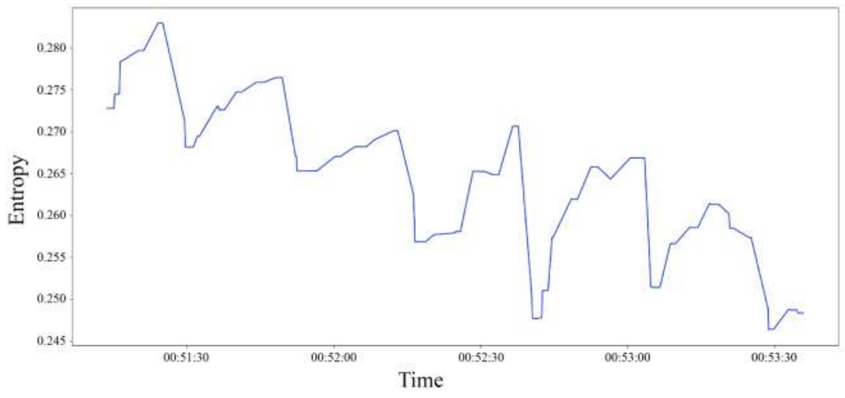

Fig. 4. System entropy over steadying state evaluation.

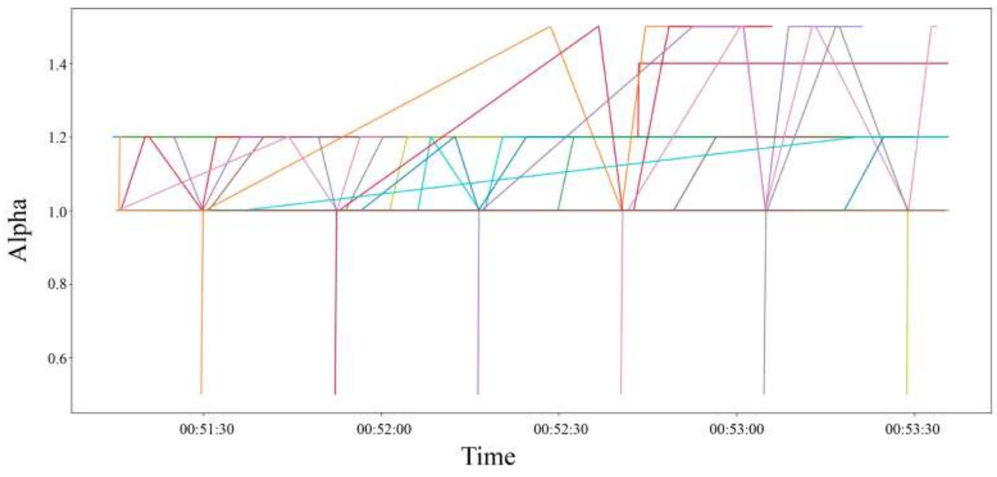

Fig. 5. Alpha per group.

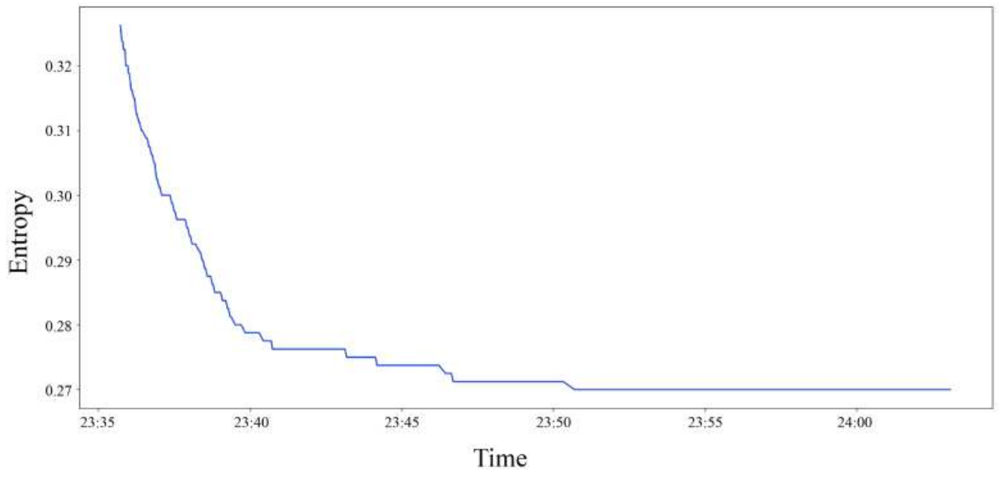

Fig. 6. System entropy over bubble evaluation.

bubble function to move a static set of users and groups towards an ideal state of low entropy. The plot indicates that our results are as expected, the system strictly decreases in entropy until it plateaus. This shows that the framework successfully improves the state of recommendations on a still set.

The average preference difference cost between each group and their respective set of users is shown in Fig. 7. From the plot, it can be seen that the costs tend to oscillate throughout the process before steadying. This is because a group can only be shown to a reasonable number of users. So when the Bubble function makes changes that improve the overall preference difference cost of the entire system, swaps occur where users are given worse matches for the sake of giving another user an improved match that outweighs their peer's loss. This results in the oscillation of the group averages.

The remainder of the experiment was done in a setup that simulated expected behaviour and use. The overall goal was to determine the

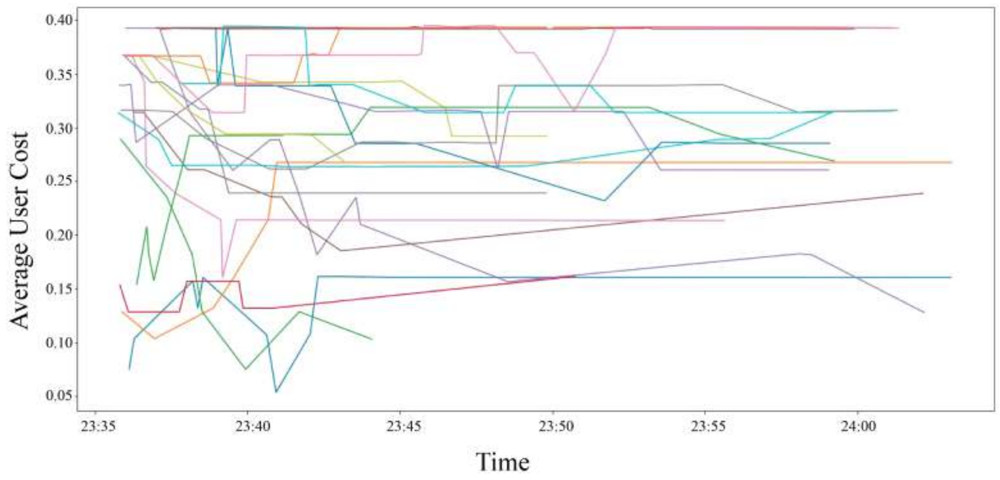

Fig. 7. Average cost of each group's users.

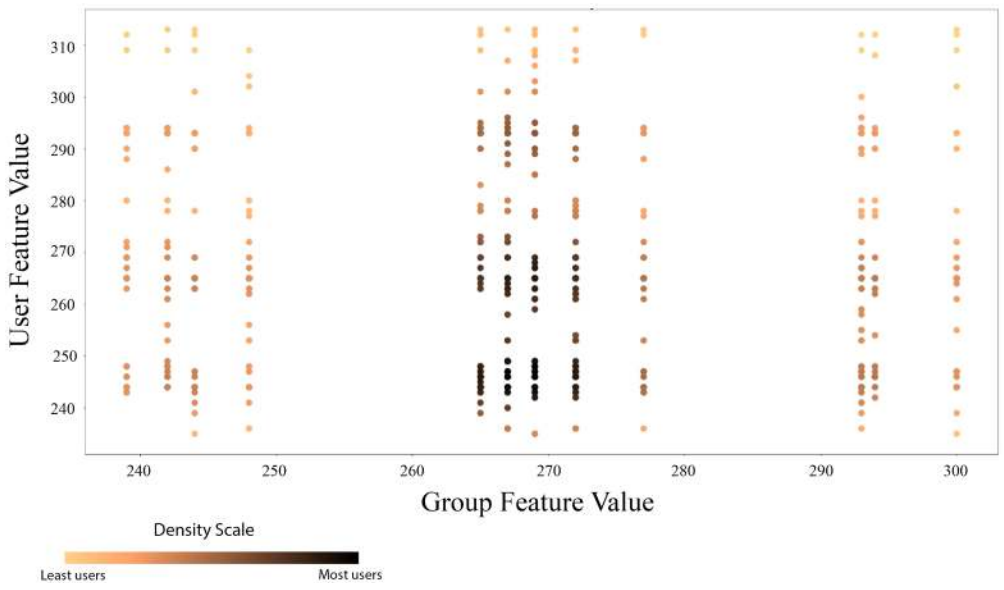

Fig. 8. Initial state of user feature values vs. group feature values.

ability of the system to maintain an ideal state following sub-optimal changes. In Fig. 4 instead of strictly decreasing, at the points of simulated actions, there were small spikes of entropy due to the sub-optimal changes. The bubble function that followed in each case immediately brought that entropy back down. It should also be noted that the plot is still trending downwards because of the further improved recommendations that were found during the changes.

As the simulation changed the set of groups in the system, the alpha values could not remain constant as before. In Fig. 5 we see the alpha values adopted an oscillating pattern close to the other plots. This demonstrates the way our use case functions balance group alpha by relocating users to new groups from high alpha ones and vice versa in response to appropriate system events.

Figure 8 shows the initial state of groupings with the initial entropy values. The arrangement is random and represents a poor allocation of users. Each group has a wide-spanning variety of user feature values shown by the spread of points on the plot. Figure 9 shows the final state of groupings after the framework functions were run. A positive linear correlation between user values and group values for the feature of interest can be seen. This shows that users move closer to one another and towards groups with comfort level values close to their own. These are the groups they would be happiest with and are most likely to get accepted into. Small outlying clumps of users are still present due to cases where the best matching group is already filled with better matches.

The results in Table 1comparing our solution to the optimum of the network flow approach show that the difference is negligible at the system size of our tests. As the system grows in size the impact of local minima may increase but we deem the trade-off of less than absolute results for near real-time solutions worth it in some applications.

5. Conclusion

In the case of forming small online groups, existing options are too impersonal because of their size and do not consider the happiness of both group seekers and creators. Some solutions such as network flow optimization and clustering algorithms may find the best possible solution, but in applications where the problem set is constantly changing, they are unable to efficiently maintain a steady solution. We provide a solution to this in the form of a framework that makes continuous steps towards the optimal in response to sub-optimal events. The framework helps group hosts get the best members from the least application

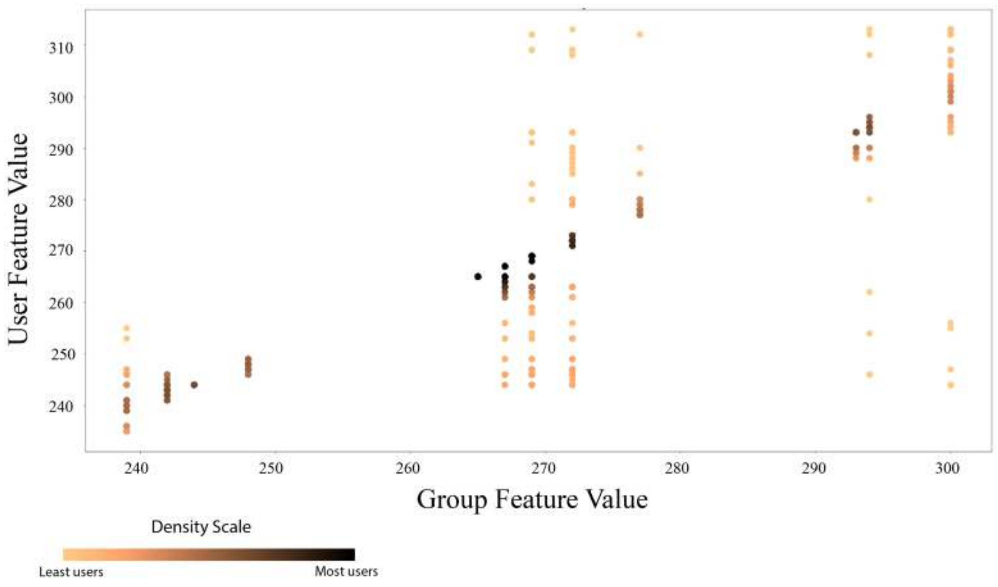

Fig. 9. Final state of user feature values vs. group feature values.

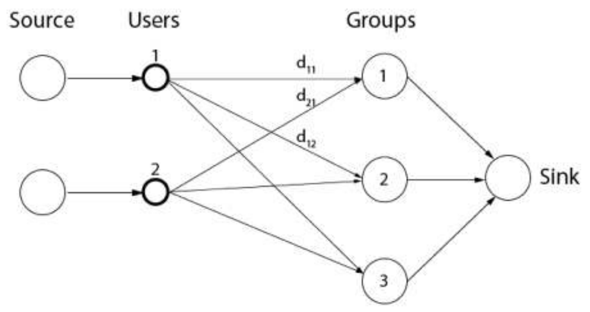

Fig. 10. Network flow model.

Table 1
Performance of proposed approach.

	Proposed approach value	Optimal value
1000 users, 100 groups	620.96	620.94
2000 users, 100 groups	1224.85	1224.82

reviews and shows users the best groups they are likely to be interested in and also accepted by since they are among the best users for it. We illustrated the framework's usefulness through practical examples and validated it through implementation and testing. While the proposed solution addresses the problem, there is still room for improvement.

Future work includes obtaining feedback on the usefulness of example applications. In each example use case the target audience is evident. We can validate their usefulness by designing and administering appropriate surveys to judge potential users' thoughts on the current state of the problem and how much they value a solution.

Furthermore, we can improve the depth of the implementation and simulations by focusing on a single scenario like recruitment. Our implementation featured two cost functions for the sake of simplicity and ease of understanding. Assessing a combination of more cost functions would be a closer representation of a real application. Moreover, the simulations by which we assess the implementation can be improved by modelling the experiments to closer resemble a real system. We can do this by emulating a realistic user growth curve and churn rate. Improving the quality and variety of how we evaluate the framework can create chances for beneficial discoveries that would lead to more successful production use.

The current approach describes cases and functions focused on similarity as an indicator for good match fit. However, existing work shows that trust among users is just as important as a measure of similarity De Meo et al. (2015, 2017a, 2017b). This can be addressed by exploring the use of trust frameworks within the network. For example, work by Ramoudith and Hosein (2020) demonstrates a framework that can produce trust values for users within a network. This value can then be included in our approach as a weighted feature. Thus, including trust, while keeping the ability to maintain ideal solutions in environments with frequently changing sets.

Finally, we plan to investigate the integration of algorithms used in the network flow formulation of the problem with our framework. Replacing the greedy approach of our bubble function with such an algorithm will allow us to take advantage of their efficiency while benefiting from our framework's event-driven focus on subsets of the network. The constraints used will be similar to the evaluation section above. The capacity of groups will be decided by the framework's alpha balancing cases as usual. The updated bubble function will continue to not affect how many users are recommended each group, only the quality of the recommendations.

CRediT authorship contribution statement

Reshawn Ramjattan: Methodology, Software, Writing – original draft, Visualization, Investigation, Validation. **Nicholas Hosein:** Conceptualization, Methodology, Formal analysis, Resources, Writing – review & editing. **Patrick Hosein:** Conceptualization, Formal analysis, Supervision, Writing – review & editing. **Andre Knoesen:** Conceptualization, Formal analysis, Supervision, Writing – review & editing.

References

OKCupid: The Math Behind Online, 2020. Dating.https://blogs.ams.org/mathgradblog/2016/06/08/okcupid-math-online-dating/
Agarwal, S., Lorch, J.R., 2009. Matchmaking for online games and other latency-sensitive P2P systems. Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, pp. 315–326.
Attiya, I., Abd Elaziz, M., Xiong, S., 2020. Job scheduling in cloud computing using a modified harris hawks optimization and simulated annealing algorithm. Comput. Intell. Neurosci. 2020.
Bertsekas, D., 1985. A distributed asynchronous relaxation algorithm for the assignment problem. 1985 24th IEEE Conference on Decision and Control, pp. 1703–1704.
Bertsekas, D.P., 1992. Auction algorithms for network flow problems: a tutorial introduction. Comput. Optim. Appl. 1 (1), 7–66.

Boroń, M., Brzeziński, J., Kobusińska, A., 2020. P2P matchmaking solution for online games. Peer-to-Peer Netw. Appl. 13 (1), 137–150.
Brozovsky, L., Petricek, V., 2007. Recommender system for online dating service. arXiv preprint arXiv:cs/0703042.
Chamorro-Premuzic, T., Furnham, A., Lewis, M., 2007. Personality and approaches to learning predict preference for different teaching methods. Learn. Individ. Differ. 17 (3), 241–250.
De Meo, P., Messina, F., Pappalardo, G., Rosaci, D., Sarnè, G.M., 2015. Similarity and trust to form groups in online social networks. OTM Confederated International Conferences "On the Move to Meaningful Internet Systems". Springer, pp. 57–75.
De Meo, P., Messina, F., Rosaci, D., Sarn, G.M., 2017. Combining trust and skills evaluation to form e-learning classes in online social networks. Inf. Sci. 405 (C), 107–122. https://doi.org/10.1016/j.ins.2017.04.002.
De Meo, P., Messina, F., Rosaci, D., Sarné, G.M., 2017. Forming time-stable homogeneous groups into online social networks. Inf. Sci. 414, 117–132.
Delahaye, D., Chaimatanan, S., Mongeau, M., 2019. Simulated annealing: from basics to applications. Handbook of Metaheuristics. Springer, pp. 1–35.
Dolmans, D.H., Schmidt, H.G., 2006. What do we know about cognitive and motivational effects of small group tutorials in problem-based learning? Adv. Health Sci. Educ. 11 (4), 321.
Dubins, L.E., Freedman, D.A., 1981. Machiavelli and the Gale-Shapley algorithm. Am. Math. Month. 88 (7), 485–494.
Gelareh, S., Monemi, R.N., Semet, F., Goncalves, G., 2016. A branch-and-cut algorithm for the truck dock assignment problem with operational time constraints. Eur. J. Oper. Res. 249 (3), 1144–1152.
Ghomi, E.J., Rahmani, A.M., Qader, N.N., 2017. Load-balancing algorithms in cloud computing: a survey. J. Netw. Comput. Appl. 88, 50–71.
Hitsch, G.J., Hortaçsu, A., Ariely, D., 2010. Matching and sorting in online dating. Am. Econ. Rev. 100 (1), 130–163.
Kuhn, H.W., 1955. The Hungarian method for the assignment problem. Nav. Res. Logist. Q. 2 (1–2), 83–97.
Li, X., Yuan, J., Li, E., Yao, W., Du, J., 2019. Trust-aware and fast resource matchmaking for personalized collaboration cloud service. IEEE Trans. Netw. Serv.Manage. 16 (3), 1240–1254.
Mackinnon, L., 2015. Love's Algorithm: Algorithmic Life: Calculative Devices in the Age of Big Data, 161.
Manweiler, J., Agarwal, S., Zhang, M., Roy Choudhury, R., Bahl, P., 2011. Switchboard: a matchmaking system for multiplayer mobile games. Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, pp. 71–84.
McCrae, R.R., John, O.P., 1992. An introduction to the five-factor model and its applications. J. Pers. 60 (2), 175–215.
Myślak, M., Deja, D., 2014. Developing game-structure sensitive matchmaking system for massive-multiplayer online games. International Conference on Social Informatics. Springer, pp. 200–208.
Ramjattan, R., Hosein, N., Hosein, P., Knoesen, A., 2020. Dynamic group formation with suitability constraints in large social networks. 2020 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD). IEEE, pp. 1–6.
Ramoudith, S., Hosein, P., 2020. A trust framework for the collection of reliable crowd-sourced data. Future of Information and Communication Conference. Springer, pp. 42–54.
Saadatpour, M., Afshar, A., Khoshkam, H., 2019. Multi-objective multi-pollutant waste load allocation model for rivers using coupled archived simulated annealing algorithm with qual2kw. J. Hydroinf. 21 (3), 397–410.
Springer, L., Stanne, M.E., Donovan, S.S., 1999. Effects of small-group learning on undergraduates in science, mathematics, engineering, and technology: a meta-analysis. Rev. Educ. Res. 69 (1), 21–51.
Traud, A.L., Kelsic, E.D., Mucha, P.J., Porter, M.A., 2011. Comparing community structure to characteristics in online collegiate social networks. SIAM Rev. 53 (3), 526–543.
Tu, K., Ribeiro, B., Jensen, D., Towsley, D., Liu, B., Jiang, H., Wang, X., 2014. Online dating recommendations: matching markets and learning preferences. Proceedings of the 23rd International Conference on World Wide Web, pp. 787–792.
Van Laarhoven, P.J., Aarts, E.H., 1987. Simulated annealing. Simulated annealing: Theory and applications. Springer, pp. 7–15.
Zaharia, M., 2009. Job scheduling with the fair and capacity schedulers. Hadoop Summit 9.
Zubiaga, A., Voss, A., Procter, R., Liakata, M., Wang, B., Tsakalidis, A., 2017. Towards real-time, country-level location classification of worldwide tweets. IEEE Trans. Knowl. Data Eng. 29 (9), 2053–2066.

Reshawn Ramjattan obtained B.Sc and MSc. Degrees in Computer Science from The University of the West Indies, St. Augustine. During his second year, he interned at First Citizens Internal Audit where he wrote data analysis scripts. Mr. Ramjattan went on to spend some time as a developer at a retail software company before becoming a part time tutor at the Department of Computing and Information Technology. He is currently a member of the TTLAB research group and a Data Science Intern at CIBC First International Caribbean Bank.

Nicholas Hosein rejoined the PhD program at UC Davis after taking three years off to start fashion wearable company, Kábrya, as well as to work full time as a life coach across Europe and the Americas. He has his MS from UC Davis in Ai and algorithms design for low power embedded systems and his BS from UC Berkeley in Electrical Engineering and Computer Science. His contributions to education include designing and launching a cross disciplinary course in IoT startups. Nick has an interest in using social dynamics to improve society and how we educate.

Patrick Hosein attended the Massachusetts Institute of Technology (MIT) where he obtained five degrees including a PhD in Electrical Engineering and Computer Science. He has worked at Bose Corporation, Bell Laboratories, AT&T Laboratories, Ericsson and Huawei. He has published extensively with over 100 refereed journal and conference publications. He holds 40 granted and 42 pending patents in the areas of telecommunications and wireless technologies. He was nominated for the Ericsson Inventor of the Year award in 2004 and was the Huawei US Wireless Research Employee of the year for 2007. Patrick is presently a Professor of Computer Science at the University of the West Indies. His present areas of research include radio resource management, QoS and pricing for 5G cellular networks.

Andre Knoesen received his Ph.D. degree from the Georgia Institute of Technology, Atlanta in 1987. He is a Professor and Department Chair of the Department of Electrical Engineering, University of California, Davis. He is the principal author of a successful interactive online textbook "Introduction to MATLAB" published by zyBooks. His current research is in cyber-physical systems and sensor networks. Dr. Knoesen is a fellow of the Optical Society of America.