Computational Social Networks
a SpringerOpen Journal

RESEARCH Open Access

# A partner-matching framework for social activity communities

Chunyu Ai[1*], Wei Zhong[1], Mingyuan Yan[2] and Feng Gu[3]

*Correspondence:
aic@uscupstate.edu
[1]Division of Mathematics and Computer Science, University of South Carolina Upstate, 800 University Way, 29303 Spartanburg, SC, USA
Full list of author information is available at the end of the article

**Abstract**

A lot of daily activities require more than one person to participate and collaborate with each other; however, for many people, it is not easy to find good partners to engage in activities with one another. With the rapid growth of social network applications, more and more people get used to creating connections with people on the social network. Therefore, designing social network framework for partner-matching is significant in helping people to easily find good partners. In this paper, we proposed a framework which can match partners for an active community. In order to improve the matching performance, all users are divided into groups based on a specific classification tree that is built for a specific activity. The optimization goal of the partner-matching is to maintain as many stable partnerships as possible in the community. To achieve the goal, various factors are considered to design matching functions. The simulation results show that the proposed framework can help most people find stable partners quickly.

**Keywords:** Social networks; Partner-matching; Stable partnership

## Background

A lot of daily activities require two or more people to collaborate. For instance, playing tennis, squash, rock climbing, and ballroom dancing. However, a lot of people liking these activities suffer from finding good partners. Some people have partners; however, it is still impossible to engage in these activities at the time they preferred since their partners might not be available at that time. Also, partners with different levels usually do not enjoy playing with each other. Everyone wants to have good partners since good partners can help each other improve their performance and skills efficiently. Lacking of matched partners really can ice people's enthusiasm for these activities. An application which can find good partners for people has high practical utility. Obviously, it can encourage more people to engage in these activities.

Nowadays, social networks have integrated into our daily life [1-5]. More and more people use social network applications everyday. A survey shows that 74% of online adults use social network sites as of January 2014 [6]. Therefore, for most of people, it is easy to accept using a social network application to find partners. However, randomly finding partners through social networks is not so efficient. First of all, it is difficult and takes time for users to find partners who meet their requirements via social networks. Most importantly, people have to take various risks to be partners with strangers

found on social network sites. Therefore, people rarely try to find partners via social networks. This motivates us to design a partner-matching framework for users to find good partners.

Usually, people engage in the activities which need partners at certain places such as a gym, fitness center, and classroom. Obviously, people usually do not want to change their activity locations for a new partner. Therefore, the searching pool should be limited to the people who engage in the same activity at the same place. Partner-matching applications can be used by these places to help their members to have better experiences and also attract more new members. The partner-matching application can be implemented as a social network website and/or mobile application.

The challenges of designing a partner-matching framework are as follows:

1. How to find a good partner for a user? A good partner for one may be bad for another since users have different definitions of good partners. Therefore, we cannot use the same criteria for every user. Only the user can rate the selected partner is good one or not according to their collaboration experience.
2. How to make the utmost effort to benefit all users? It is very difficult to satisfy every user. Then, how to match partners to benefit as many users as possible is a complicated optimization issue.
3. How to protect every user's privacy? In order to match partners very well, users need to provide their personal information such as age, gender, years of experience, and contact information. However, users do not want anyone else to access their private information. Thus, protecting the users' privacy is a big challenge.

In this paper, we proposed a partner-matching framework to address this issue. Our proposed framework designs to use a classification tree to distribute users to leaf nodes of the tree; thus, for a user's request, the searching pool is way smaller compared to finding the best match among all candidates. Apparently, the matching performance can be improved significantly by using the classification tree. Moreover, in order to achieve the optimization goal of maintaing as many stable partnerships as possible, we proposed matching functions which consider various factors. The simulation results show that our framework can find good partners for most of the users quickly.

The rest of the paper is organized as follows: 'Related work' section reviews the related work of this study. The proposed mechanism is introduced in the 'Partner-matching framework' section. 'Simulation' section shows the simulation results. The 'Conclusion' section concludes our work.

## Related work

Gale and Shapley proposed an algorithm to solve the college admissions and the stability of marriage problem [7]. The stable marriage problem is the problem of finding a stable match between two sets of elements given a set of preferences for each element. A matching is a mapping from the elements of one set to the elements of the other set. The Gale-Shapley algorithm solves the stable marriage problem. It executes in rounds. In the first round, each man proposes to the woman he prefers most. A woman faces one of the following three situations: 1) no one proposes to her, 2) only one man proposes to her, and 3) there are more than one man who propose to her. For the first situation, the woman does not need to do anything and just wait. For the second situation, the woman accepts

the proposal. For the third situation, the woman accepts the suitor she prefers most and rejects the rest of suitors. After the first round, each single man proposes to the woman he prefers most among all women who did not reject him no matter if she is single or not. A woman still picks the one she prefers most among all suitors and rejects the rest suitors. If a woman already accepted a man in the previous rounds, when she has a better suitor, she must reject the current match and accept the new suitor. Therefore, some men might become single again, then they can propose to another women in the next round. After rounds and rounds, till no one is single, there will be no new proposal; thus, the whole process ends. The stable marriages are achieved. David Gale and Lloyd Shapley proved that for any equal number of men and women, it is always possible to solve the stable marriage problem and make all marriages stable. The time complexity is $O(n^2)$. The Gale-Shapley algorithm is not suitable for the partner-matching problem since unlike the marriage relationship, we cannot divide people into two sets.

Dubins and Freedman [8] proved that for student admission problem, a student cannot improve his/her fate via lying about his/her preferences. Gale and Sotomayor [9] and Roth [10] proposed further analysis of the college admission and stable marriage problem. Roth [11] addressed that the college admission problem is not equivalent to the marriage problem. There does not exist a stable matching procedure to make a dominate strategy for colleges to reveal their preferences. Roth [12] studied the stability and polarization of interests in job matching. Roth [13] studied the common and conflicting interests in two-sided markets. In [14], the preferential partner selection in an evolutionary study of a prisoner's dilemma was studied. Marriage and employment relationship matching problem is addressed in [15]. Beckman et al. [16] studied the network partner selection.

A variant of stable marriage problem is the stable roommate problem which is a non-bipartite extension of stable marriage problem. Given $2n$ persons, each has a preference list over the other $2n-1$ people. The stable roommate problem is to find a stable matching [17]. For a roommate matching problem instance, there might not exist a stable matching. In [18], an algorithm with $O(n^2)$ time complexity is proposed to determine whether a stable matching exists for any instance of the problem and find one if it exists. The problem of finding a matching with the minimum number of blocking pairs is NP-hard, and it is hard to approximate [19]. Unfortunately, algorithms of stable roommate problem cannot be applied to partner-matching applications since giving a preference list will violate the rule of protecting privacy of users.

To the best of our knowledge, existing algorithms are not suitable for solving the partner-matching problem efficiently. Obviously, randomly matching partners or round robin will not find stable partnerships quickly since people will keep requesting a new partner when they are not satisfied with the current one. In this paper, we proposed a mechanism to match partners for a social activity community.

## Partner-matching framework

In this section, we introduce our proposed partner-matching framework. People who need to find partners can send a request. The partner searching pool includes people who request partners and all members of a certain place where the activity is held. Even though some members have no intention to send a partner-matching request, they might accept a partner request from others. Therefore, we also add them to the searching pool.

A profile is created for every member in the searching pool. An example profile for an indoor rock climbing gym is shown in Table 1 [20]. For different activities, the profile is designed specifically according to the features of the activity. When a climber uses a harness and rope as protection from falling, he/she needs a belayer to operate these belaying devices to ensure a falling climber does not fall very far. It is important for the belayer to closely monitor the climber's situation as the belayer's role is crucial to the climber's safety. Therefore, a rock climber usually carefully picks a partner. Usually, an indoor rock climbing gym requires climbers to pass a test to get a belay certification. A climber without a certification is not allowed to belay other climbers. There are two belay certification levels, top rope and lead climbing. A climber who can do lead climbing also can do top rope but not vice versa. The climbing route difficulty level is from 5.6 to 5.13 usually for an indoor gym. Climbers like to be partners with someone who has similar or more advanced climbing skills than themselves, thus they will improve more quickly.

For request senders, we can obtain their profiles easily when they send the requests. For the regular members of the place, we use their membership information to fill the profile, thus some information might be missing such as years of experience, level, and time schedule. A time schedule can be summarized according to check-in records of the past few months. If someone does not have a regular time schedule, we use the term *random* to fill it. Also, we can use their years of membership as their estimated years of experience, then use the average values of other members with the same years of experience to fill in other missing items. For these members with estimated items in their profiles, we put an *estimated* mark on them.

When a user signs in to use the partner-matching application, he/she needs to complete a form to describe their requirements to potential partners. An example request form is shown in Table 2. The request form is designed according to the features of the activities. Requirements are divided into two categories, strong requirements and weak requirements. Strong requirements are more important for users; thus, strong requirements must be totally or partially met. Weak requirements do not affect collaboration experience of partners obviously, so these are used for ranking search results when multiple candidates are found. For the climbing gym example, belay certification level, climbing level, and scheduling are strong requirements. Age and gender are weak requirements.

For the strong requirement items, users must specify a value or choose a range. For each requirement item, there are a few options from which to choose. In Table 2, we

**Table 1 Member profile for a rock climbing gym**

| Parameter | Value |
|---|---|
| Name | Rachel |
| Phone | 555-555-5555 |
| Email address | rachel@partnerfinding.com |
| Gender | Female |
| Age | 35 |
| Years of experience | 2 |
| Belay certification level | Top rope |
| Climbing level | 5.9 to 5.10 |
| Climbing time | Weekday evenings |

**Table 2 Partner request**

| Requirement | Options |
|---|---|
| Belay certification level | Top rope; lead climb |
| Climbing level | 5.6 to 5.7; 5.8 to 5.9; 5.10 to 15.11; 5.12 or above |
| Schedule | Weekday mornings; weekday afternoons; weekday evenings; weekends |
| Preferred gender | Male; female; none |
| Preferred age | Under 30; 31 to 40; 40 or above; none |

use semicolons to separate options. For some requirement items, users can choose more than one option. For instance, in Table 2, users can choose more than one *schedule* and *climbing level* option. But users can only choose one option at *belay certification level*. The domain of a requirement item which has values within a certain range are divided into several continuous intervals such as preferred age.

### Problem definition

The optimization goal of the partner-matching application is to construct as many stable partner relationships as possible. A partner relationship is stable when each party thinks he/she found the best partner already or he/she cannot find a better partner. The rating system in Table 3 is used for rating partners.

A set $P$ is created including everyone who has either sent a partner request or accepted a partner request. For a person $p_i$ in $P$, $R_{p_i}$ is the rate he/she gave to the current partner, If he/she does not have a partner currently or if a person $p_i$ sent a request but never found a partner, $R_{p_i} = 0$. The goal of matching partners is as follows:

$$\text{Maximize} \sum_{i=1}^{|P|} R_{p_i} \tag{1}$$

In order to achieve this optimization goal, we propose a partner-matching strategy in the 'Partner-matching' section.

### Classification

If a user requests a partner, the system can recommend a list of members who meet or almost meet the requirements of the user. In order to efficiently find the members who meet the requirements, we use classification techniques to divide all members in the searching pool into groups. SP is the searching pool. $\text{SP} = \{m_1, m_2, \cdots, m_{|\text{SP}|}\}$ where $m_i$ is the profile of member $i$ in the searching pool.

**Table 3 Partnership rating**

| Score | Rate | Action | Possibility of accepting a new partner (%) |
|---|---|---|---|
| 4 | Excellent | None | 0 |
| 3 | Good | Little chance of accepting new partners | 25 |
| 2 | Fair | Big chance of accepting new partners | 50 |
| 1 | Poor | Request a new partner | 75 |

Assume there are $n$ requirements, $r_1, r_2, \cdots, r_n$, in the partner request form. For every requirement $r_i$, we assign a weight $w_i$ for it, and the total weight of all requirements is 1 as shown in Formula 2.

$$\sum_{i=1}^{n} w_i = 1 \tag{2}$$

The more important the requirement is the bigger weight is assigned. The opinions of the experts of this activity are necessary for weight assignment since it will affect a user's degree of satisfaction to recommended partners. Table 4 shows the weight assignment for the rock climbing gym example. Usually, the total weight of weak requirements is not more than 0.1 since they are not so important compared to strong requirements.

Assume there are $s$ strong requirements. We sort strong requirements in descending order by weights and store in the list $R_S = r_1, r_2, \cdots, r_s$, where $w_i > w_j$ if $i < j$. Each requirement $r_i$ in $R_S$ has $o_i$ options. $r_{i_k}$ indicates the $k$th option of requirement $r_i$. These strong requirements are used to divide all members into groups. The requirement with bigger weight is applied first. The purpose is to reduce the searching pool size when processing a user's request. A classification guide tree is used to help the group division process.

Initially, we create an empty classification tree $T$. A tree node $t$ has the structure $\{r_{i_k}, \text{member\_list}, \text{child\_list}\}$. $r_{i_k}$ is the classification attribute used for the current tree node. *member_list* includes all members who fall into the group that the current tree node represents. *child_list* contains links to child nodes of the current tree node. In the classification tree, all members are in the leaf nodes; in other words, the *member_list* of an internal node is empty. Therefore, for any tree node, either *member_list* or *child_list* is empty.

Algorithm 1 describes the basic process of building a classification tree. First, the root node is created. Its *member_list* initially includes all members in the searching pool. Then, all options of the requirement $r_1$ are used as grouping criteria to divide all members into separate groups. For each option, a new group is generated. We create a new tree node for each group to store the members in this group. Also, all these new created nodes are children of the current node. Then, for these new nodes, options of the next requirement in the list $R_S$ are used to partition their members to new groups. This process is repeated until the last strong requirement is applied. However, if members are not evenly distributed based on these options of requirements or there is no sufficient members, a lot of sparse nodes (groups without member or just with few members) will be generated. The tree with many sparse nodes is not desired since it leads to poor searching performance. In order to avoid generating too many sparse nodes, a *classifi-*

**Table 4 Partner request**

| Requirement | Weight |
|---|---|
| Belay certification level | 0.4 |
| Climbing level | 0.3 |
| Schedule | 0.2 |
| Preferred gender | 0.05 |
| Preferred age | 0.05 |

*cation threshold* is predefined. When the number of members of the current processing node is less than the *classification threshold*, it is not necessary to continue the partition process.

**Algorithm 1 Classification**

**Input:** $SP$, $R_S$, $OS$, and *classification_threshold*

**Output:** Classification tree $T$

create a queue $Q$, set $Q$ empty initially
create a tree $T$ with only the root node *root* where $root = \{null, SP, \emptyset\}$
add the root node *root* to $Q$
**while** Q is not empty **do**
  $t$ = dequeue($Q$) {return the first node in the queue}
  **if** |member_list| of $t$ is greater than *classification_threshold* **then**
    **if** the first element classification attribute $r_{i_k}$ of $t$ is *null* **then**
      *classification_requirement* = $r_1$
    **else if** $i$ of $r_{i_k}$ is less than $s$ {did not apply the last strong requirement yet} **then**
      *classification_requirement* = $r_{i+1}$
    **end if**
    **for** each option $r_{j_k}$ of *classification_requirement* **do**
      create a tree node $c = \{r_{j_k}, \emptyset, \emptyset\}$
      remove every member who meet $r_{j_k}$ in *member_list* of $t$ and add to the *member_list* of $c$
      **if** *member_list* of $c$ is not empty **then**
        add $c$ to the *child_list* of $t$
        add $c$ to $Q$
      **end if**
    **end for**
  **end if**
**end while**

The classification tree of the rock climbing gym example is shown in Figure 1. Usually, there is no 5.12 and 5.13 top rope routes in an indoor rock climbing gym, so there is no child node for levels 5.12 to 5.13 for top rope. Also, lead climbers do not climb easy routes like levels 5.6 and 5.7, so there is no node for levels 5.6 to 5.7 for lead climbing. The node for level $\geq$ 5.12 of lead climbing is not partitioned further since there are not enough climbers in the group to support further partition on schedule [20].

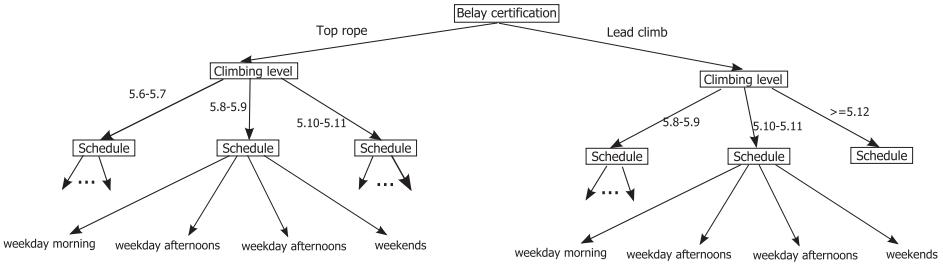


**Figure 1 Classification tree of rock climbing gym.**

After the classification tree is generated, we can search members who meet certain requirements easily via visiting from the top to the corresponding leaf node. If a new member just joins in, we can use his/her profile to search the tree and find the matched leaf node to insert the member. Each member in the searching pool is stored in one leaf node of the classification tree $T$. If a member's certain attributes in the profile are updated, we can relocate the member to the proper leaf node by removing the member and then reinserting into the tree.

By using the classification tree to partition members, the searching performance can be improve significantly since only one or a few nodes are accessed instead of the whole searching pool.

#### Partner-matching

Maintaining as many stable partnerships as possible is the optimization goal of the partner-matching application. When a partner request $pr$ is received, we will search the classification tree to find all members who meet the strong requirements of the $pr$ and add them to candidate set $C_{pr}$. If there is no more than ten candidates found in the corresponding leaf node, add left and right direct sibling's members into the candidate set. If a user does not think his/her current partner is the best partner he/she can find, he/she will keep trying to send partner requests or accept requests until the best match is found. In order to make each user find their matched partner as early as possible, the method of ranking candidates becomes significant. For a member $m$ in $C_{pr}$, function $f(m)$ is designed to measure how well the member $m$ matches the request $pr$.

$$f(m) = \sum_{1}^{n} w_i * \text{meet}_i \tag{3}$$

where $w_i$ is the weight of requirement $i$. $\text{meet}_i$ is 1 if member $m$ meets the requirement $i$; otherwise, $\text{meet}_i$ is 0. The more requirements a candidate satisfy, the bigger is his/her $f(m)$.

How well a pair of partners match relies on both parties. The candidate with the largest $f(m)$ is the best candidate for the requester. However, the requester might not meet the expectation of the candidate. Therefore, only considering the benefits of the requester is not sufficient. Benefits of the candidates are also important for maintaining stable partnerships. Therefore, we redefine $f(m)$ as follows:

$$f(m) = \left(\sum_{1}^{n} w_i * \text{meet}_i\right) * \left(\sum_{1}^{n} w_i * \text{meet}_{i_m}\right) \tag{4}$$

where $\text{meet}_{i_m}$ is used to indicate whether the requester meets the candidate $m$'s requirement $i$. $f(m)$ in Formula 4 is suitable to measure how well the requester and the candidate $m$ match each other.

If the best candidate is satisfied with the current partner, the possibility he/she accepts the request would be low. If the request sender was rejected by the best candidate, he/she also possibly misses the chance to win the second or third best candidate. Thus, the possibility of being accepted by a candidate also needs to be considered when we rank all candidates.

$$f(m) = \left(\sum_{1}^{n} w_i * \text{meet}_i\right) * \left(\sum_{1}^{n} w_i * \text{meet}_{i_m}\right) * \text{rate}_m \tag{5}$$

As shown in Formula 5, we add one more factor $\text{rate}_m$ which is the possibility candidate $m$ accepts a new request. The estimated possibility of accepting a request is related to the rating of the current partner. Table 3 shows our estimated possibilities. The lower the rating to the current candidate is, the higher possibility the user will send a new partner request. If a person never had a partner, we set the $\text{rate}_m$ to 50%.

$f(m)$ is used to rank all candidates of the request $pr$. Then, the system will send a request to the top one candidate. The request includes the profile of the request sender. To protect privacy of members, personal information such as name and contact information are not included. Age is sent as a range format instead of using exact age. If the request is accepted, a new partnership is established. Otherwise, send a request to the next candidate in the ranked list. The system repeatedly sends the request out until someone accepts the request or all candidates are probed already. A reply waiting time is set up. If the receiver did not reply in time, the request is withdrawn and continues to probe to the next candidate. After probing all candidates in the list, if no one in the candidate set accepts the request, the partner request sender can file another request. Candidates who made a rejection prior might accept the request from the same sender since they are not satisfied with the current partner.

Since a member changes a partner when there is a better choice, most members will have a stable partnership over time. A few members with improper behaviors are not capable of maintaining a long-term partnership. Therefore, they have to keep sending requests and changing partners.

## Simulation

We use the rock climbing gym example to evaluate the proposed framework. In our simulation, there are total 500 members. One hundred of them requests partners initially. If a member gave the current partner a poor rating, he/she definitely sends a new request. Whenever someone accepts the request, he/she will break the partnership with the current partner. The member abandoned by the partner will send a new partner request to find a new partner.

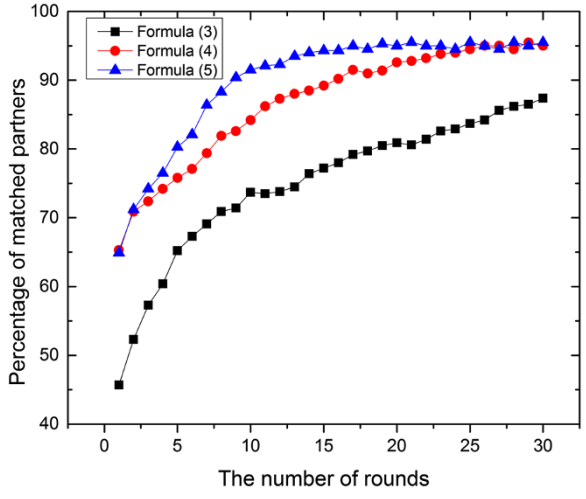


**Figure 2 Percentage of matched partners.**

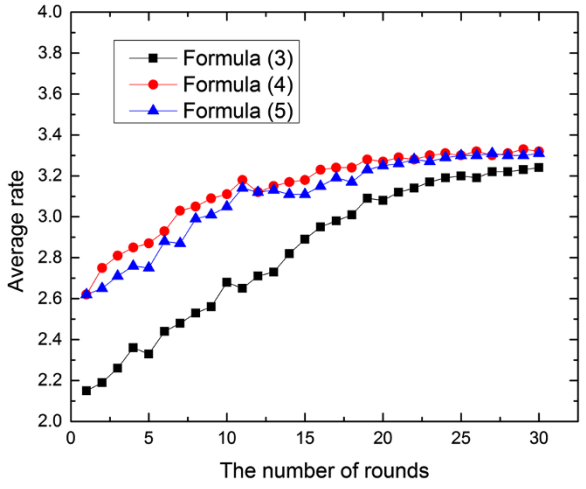


**Figure 3 Average rate to partners.**

We use the ranking function $f(m)$ in Formulas 3, 4, and 5, respectively to evaluate the performance of our partner-matching mechanism. The matching program is run by rounds. In each round, a person can only send or receive one request. Figure 2 shows the percentage of matched partners within 30 rounds. The percentage of matched partners is calculated by following Formula 6.

$$\frac{\text{number of requesters who currently have a partner}}{\text{the total number of requesters}} \tag{6}$$

As seen in Figure 2, the effectiveness of Formula 3 is the worst among these three formulas. The reason is that only the requester's need is considered; however, a request might not satisfy his/her best candidate's requirements. Formulas 4 and 5 perform better since requirements of both requesters and candidates are considered. Overall, Formula 5 achieves the fastest matching progress since it also considers the possibility of the candidates to accept a new request. Formulas 4 and 5 have reached almost stable status after 20 rounds; however, Formula 3 needs more rounds to reach a stable status.

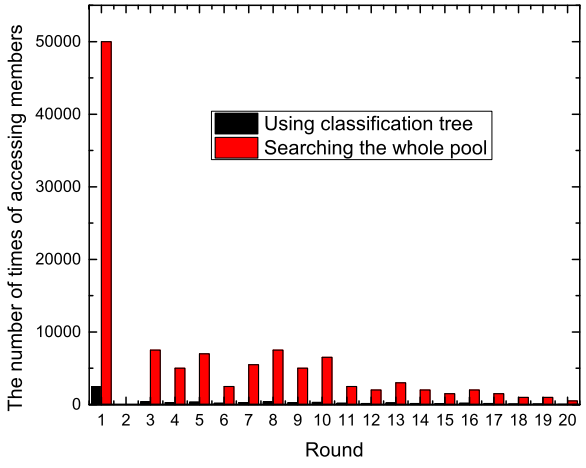


**Figure 4 The number of times of accessing members for each round.**

The average rate to partners is shown in Figure 3. Formulas 4 and 5 perform better than Formula 3 since Formula 3 does not consider the need of candidates. Thus, if the chosen candidate accepted the request, the possibility of satisfaction is not high. Formula 4 has higher average rate to partners than Formula 5 because it always chooses the best matched partners for each other. However, Formula 5 sacrifices the best match to achieve a higher request acceptance rate. Overall, Formula 5 has the fastest matching progress and gains acceptable partner rate scores.

Figure 4 shows the difference between using and without using the classification tree. As shown in the figure, in each round, the number of times of accessing members for searching candidates is significantly low when we use the classification tree compared to searching the whole pool. Therefore, the classification tree strategy obviously improves the performance of finding candidates.

## Conclusion

In this paper, a partner-matching mechanism is proposed to help people find partners in a social activity community. A classification tree is used to partition users into groups to reduce the candidate-searching time complexity. To achieve the optimization goal to maintain stable partnerships in the community, we design three matching functions. The simulation results show that the proposed framework gains good partner-matching performance. In our future work, we will consider partnerships which require more than two parties. Also, we will conduct real experiments on social network communities.

**Competing interests**
The authors declare that they have no competing interests.

**Authors' contributions**
CA designed the classification algorithm and partner matching formulas and drafted the manuscript. WZ participated in classification algorithm design and simulation design. MY implemented the simulation. FG participated matching formula design. All authors read and approved the final manuscript.

**Acknowledgement**
This work is partially supported by a RISE grant from the Office of the Vice President for Research at the University of South Carolina.

**Author details**
[1]Division of Mathematics and Computer Science, University of South Carolina Upstate, 800 University Way, 29303 Spartanburg, SC, USA. [2]Department of Computer Science, Georgia State University, P.O. Box 3994, 30302 Atlanta, GA, USA. [3]Department of Computer Science, College of Staten Island, The City University of New York, 2800 Victory Boulevard, 10314 Staten Island, New York, USA.

Received: 11 August 2014 Accepted: 10 September 2014
Published online: 03 December 2014

**References**

1. Adamic, L, Adar, E: How to search a social network. Social Netw. **27**(3), 187–203 (2005)
2. Kumar, R, Novak, J, Tomkins, A: Structure and evolution of online social networks. In: Yu, PS, Han, J, Faloutsos, C (eds.) Link Mining: Models, Algorithms, and Applications, pp. 337–357. Springer, New York, (2010)
3. Cho, E, Myers, SA, Leskovec, J: Friendship and mobility: user movement in location-based social networks. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '11, pp. 1082–1090. ACM, New York, NY, USA, (2011)
4. Xiang, R, Neville, J, Rogati, M: Modeling relationship strength in online social networks. In: Proceedings of the 19th International Conference on World Wide Web. WWW '10, pp. 981–990. ACM, New York, NY, USA, (2010)
5. Szell, M, Lambiotte, R, Thurner, S: Multirelational organization of large-scale social networks in an online world. Proc. Natl. Acad. Sci. **107**(31), 13636–13641 (2010)
6. Social Networking Fact Sheet. http://www.pewinternet.org/fact-sheets/social-networking-fact-sheet/. Accessed 21 Aug 2014
7. Gale, D, Shapley, LS: College admissions and the stability of marriage. Am. Math. Mon. **69**(1), 9–15 (1962)
8. Dubins, LE, Freedman, DA: Machiavelli and the Gale-Shapley algorithm. Am. Math. Mon. **88**(7), 485–494 (1985)
9. Gale, D, Sotomayor, M: Some remarks on the stable matching problem. Discrete Appl. Math. **11**(3), 223–232 (1985)

10. Roth, AE: Misrepresentation and stability in the marriage problem. J. Econ. Theory. **34**(2), 383–387 (1984)
11. Roth, AE: The college admissions problem is not equivalent to the marriage problem. J. Econ. Theory. **36**(2), 277–288 (1985)
12. Roth, AE: Stability and polarization of interests in job matching. Econometrica. **52**(1), 47–58 (1984)
13. Roth, AE: Common and conflicting interests in two-sided matching markets. Eur. Econ. Rev. **27**(1), 75–96 (1985)
14. Mortensen, DT: Matching: finding a partner for life or otherwise. Am. J. Sociol. **94**, 215–240 (1988)
15. Ashlock, D, Smucker, MD, Stanley, EA, Tesfatsion, L: Preferential partner selection in an evolutionary study of prisoner's dilemma. Biosystems. **37**(1–2), 99–125 (1996)
16. Beckman, CM, Haunschild, PR, Phillips, DJ: Friends or strangers? Firm-specific uncertainty, market uncertainty, and network partner selection. Organ. Sci. **15**(3), 259–275 (2004)
17. Iwama, K, Miyazaki, S: A survey of the stable marriage problem and its variants. In: Informatics Education and Research for Knowledge-Circulating Society, 2008. ICKS 2008. International Conference on, pp. 131–136. IEEE, Kyoto, (2008)
18. Irving, RW: An efficient algorithm for the 'stable roommates' problem. J. Algorithms. **6**(4), 577–595 (1985)
19. David, J, Abraham, PB, Manlove, DF: 'Almost stable' matchings in the roommates problem. Approximation Online Algorithms. **3879**, 1–14 (2006)
20. Rock Climbing Ratings from 5.0 to 5.15. http://outdoorswithdave.com/climbing/climbing-ratings. Accessed 12 Dec 2013

doi:10.1186/s40649-014-0005-0
**Cite this article as:** Ai *et al.*: A partner-matching framework for social activity communities. *Computational Social Networks* 2014 **1**:5.