RW-Team: Robust Team Formation using Random Walk

John Nemec
john.nemec@ontariotechu.ca
Ontario Tech University, Canada

Heidar Davoudi
heidar.davoudi@ontariotechu.ca
Ontario Tech University, Canada

Lukasz Golab
lgolab@uwaterloo.ca
University of Waterloo, Canada

Mehdi Kargar
kargar@ryerson.ca
Ryerson University, Canada

Yuliya Lytvyn
ylytvyn@uwaterloo.ca
University of Waterloo, Canada

Piotr Mierzejewski
piotrm@ca.ibm.com
IBM Ltd., Canada

Jaroslaw Szlichta
jarek@ontariotechu.ca
Ontario Tech University, Canada

Morteza Zihayat
mzihayat@ryerson.ca
Ryerson University, Canada

ABSTRACT

There is a growing need to find meaningful teams in expert networks such as DBLP and GitHub. However, existing team formation methods, such as those based on shortest paths between experts, may generate weakly-connected teams. We demonstrate RW-Team, a robust team formation framework based on a random walk with restart (RWR). We introduce a greedy algorithm to reduce the search space, and we use a Monte Carlo approximation of RWR to improve performance. To handle large graphs, we implement RW-Team in Apache Spark. The proposed demonstration will allow participants to form teams of researchers having various skill sets and explore connections among team members using several graph visualization techniques.

CCS CONCEPTS

• **Information systems** → **Data management systems**.

KEYWORDS

Team Formation; Attributed Graphs; Social Networks

ACM Reference Format:
John Nemec, Heidar Davoudi, Lukasz Golab, Mehdi Kargar, Yuliya Lytvyn, Piotr Mierzejewski, Jaroslaw Szlichta, and Morteza Zihayat. 2021. RW-Team: Robust Team Formation using Random Walk. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3459637.3481969

1 INTRODUCTION

An expert network is a social network that represents professionals and their skills. Expert network providers include GitHub, LinkedIn and the computer science bibliography DBLP. Expert networks can be modeled as graphs whose nodes correspond to experts (labeled with their skills) and whose edges represent past collaborations, e.g., co-authoring a paper in DBLP or collaborating on a project in GitHub. Edge weights encode relationship strength.

There has been recent interest in the problem of finding teams in expert networks [2, 7–9]. Given a project that requires a set of skills, the goal is to find a team of experts who collectively possess all the required skills while minimizing communication costs. Many existing approaches use shortest paths between experts as indicators of communication costs [1, 6, 7]. However, the shortest path approach lacks robustness as it does not consider the structure of the whole graph. To address this limitation, we present the RW-Team[1] system that forms robust teams in large expert networks[2].

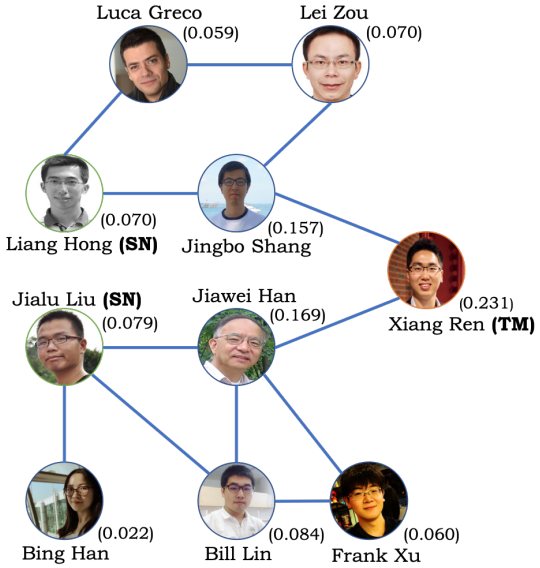


Figure 1: A portion of the DBLP network.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '21, November 1–5, 2021, Virtual Event, Australia.
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3481969

[1]System available at http://data.science.uoit.ca:8180/TeamFormation-demo
[2]Video available at https://vimeo.com/314362104

Figure 1 illustrates a fragment of the DBLP co-authorship network. Given a project that requires expertise in *Social Networks (SN)* and *Text Mining (TM)*, consider two teams of researchers, T_1: {*Liang Hong*, *Xiang Ren*} and T_2: {*Jialu Liu*, *Xiang Ren*}. Both T_1 and T_2 cover (*SN*) and (*TM*). The shortest path from *Liu* to *Ren* passes through *Jiawei Han*, the shortest path from *Hong* to *Ren* goes through *Jingbo Shang*, and both have length 2. Thus, the shortest path approach cannot distinguish between these two teams. However, *Liu* has more connections than *Hong*. Furthermore, he is connected to *Ren* through *Jiawei Han* who is a senior researcher and himself has many connections to other researchers (who are connected to other researchers and so on). Hence, we argue that *Ren* forms a more robust team with *Liu* than with *Hong*. Using a random walk with restart (RWR) [3] instead of shortest paths, RW-Team identifies such robust teams. Each node in Figure 1 is labelled with a score calculated by running RWR with *Ren* as the restart point, and we see that *Liu* has a higher score than *Hong*, meaning that T_2 would be chosen over T_1. In [4], we found that RWR-based methods form higher-quality teams compared to shortest path and embedding based methods.

Our contributions and demonstration goals are as follows.

(1) We discuss the algorithmic details behind RW-Team. Since the underlying problem of robust team discovery is NP-hard, we use a greedy heuristic and a Monte Carlo approximation of RWR.
(2) We describe a distributed implementation of RW-Team in Apache Spark, including efficient partitioning and load balancing techniques. Our framework is general enough that it can support any RWR algorithm (e.g., the power iteration method and the approximate Monte Carlo method [3]).
(3) We demonstrate RW-Team using popular expert networks including DBLP and GitHub. In addition to the RWR method that we propose, we also implemented a shortest path method and an embedding-based method from previous work. Participants will interactively examine the effect of different ranking strategies and explore the connections among team members using several graph visualization techniques.

2 SYSTEM DESCRIPTION

2.1 Definitions and Problem Statement

We model an expert network as an undirected and unweighted graph G. Each node in G represents an expert and an edge indicates a relationship or collaboration between two experts (we will use the terms expert and node interchangeably). A set of n experts is represented by $E = \{e_1, e_2, \ldots, e_n\}$, and a set of m skills is specified as $S = \{s_1, s_2, \ldots, s_m\}$. An expert e_i has a set of skills $S(e_i) \subseteq S$. Furthermore, a subset of experts $E' \subseteq E$ have skill s_j if at least one of them has s_j. For each skill s_j, the set of all experts that have s_j is denoted as $E(s_j) = \{e_i | s_j \in S(e_i)\}$. A project R needs requires a set of skills, $S_R \subset S$, for completion. Finally, a subset of experts $E' \subseteq E$ *covers* a project R if $\forall s_j \in S_R \; \exists \; e_i \in E'$ such that $s_j \in S(e_i)$.

Definition 2.1. **Team of Experts:** Given an expert network G and a project R that needs a set of skills $S_R = \{s_1, s_2, \ldots, s_k\}$ for completion, a *team of experts* T is a subgraph of G whose nodes cover R. A set of k skill-expert pairs are associated with a team which is represented by $T = \{(s_1, e_{s_1}), (s_2, e_{s_2}), \ldots, (s_k, e_{s_k})\}$, where e_{s_j} is an expert in T that has skill s_j for $j = 1, \ldots, k$.

Let $dist(e_i, e_j)$ be the distance between experts e_i and e_j. Existing approaches define distance as the shortest path between e_i and e_j in G [1, 6, 7]. Recent embedding-based methods (such as *node2vec* [5]) that transforms the graph into a vector space can also be used to define distance. In this case, the distance between e_i and e_j is equal to the Euclidean distance between their respective vectors in the vector space [4]. In RW-Team, we define distance using the notion of a random walk with restart.

Random Walk is a method to determine the importance of nodes in a network [3]. In general, a random walker "surfs" among the nodes and randomly chooses which edges to follow. When the process reaches a stationary condition, nodes receive scores that determine their importance. However, random walk does not model the distance between nodes and thus it is not suitable in our case. Instead, we use a modification called random walk with restart (RWR), in which one node (or a set of nodes) is the restart node(s) n_{res}. Here, in addition to following random edges, the random walker might teleport to a node(s) in n_{res} with probability β (common values for β are between 0.1 and 0.2). Thus, the score of each node in RWR determines its importance in the network relative to its connections to the restart node(s) n_{res}. In this work, the distance between node e_i and e_j is equal to the reverse score of e_j in the RWR process when the restart node is set to e_i (note that a higher RWR score implies e_j is more important to e_i). We can extend this notion and define the distance between a set of nodes E' and node e_j. In order to do that, the restart set is E' and the stationary score of the RWR is the distance of E' to r_j.

Definition 2.2. **Communication Cost (CC):** Given team T as a set of k skill-expert pairs $T = \{(s_1, e_{s_1}), \ldots, (s_k, e_{s_k})\}$, the communication cost of T is defined as: $CC(T) = \sum_{i=0}^{n} \sum_{j=i+1}^{n} dist(e_{s_i}, e_{s_j})$.

Problem 1. **Robust Team Formation:** *Given a graph G, a project R and the distance function based on RWR, find a team of experts T for R with minimal communication cost $CC(T)$.*

2.2 RWR Algorithm

Since Problem 1 is NP-hard (by reduction from 3-SAT; details in [4]), RW-Team uses an efficient and effective greedy algorithm. The idea is as follows. For a given project R, take each holder of the rarest skill, $E(s_{rare})$ and form a team around that expert. The team's score is initialized at 0. Then, in each iteration, we cover one of the uncovered skills. In order to do that, we run the RWR and set the restart nodes to the current members of the team. At the beginning, the only member of the team is the one with the rarest skill. When the RWR process is finished, the node with the highest score that also covers the current required skill is selected as the best member of the team. Among all the teams that are formed around the holders of the rarest skill, the one with the highest sum of RWR scores is selected as the best team. Algorithm 1 represents this process in detail. The input is the expert network G, the project R, and the skill holders of each required skill. The output is the most collaborative and robust team that covers the required skills.

Algorithm 1 Finding a Team with RWR Approach

Input: graph G with N nodes; project $R = \{s_1, s_2, \ldots, s_k\}$; the set of experts that have each skill s_i, $E(s_i)$, for $i = 1, \ldots, k$.
Output: most robust team of experts

$bestTeam \leftarrow \emptyset$; $maxTeamScore \leftarrow -\infty$
$s_{rare} \leftarrow$ skill with smallest cardinality of experts
for $i \leftarrow 1$ **to** $|E(s_{rare})|$ **do**
  $team \leftarrow$ initialize a new team
  $score \leftarrow 0$ (or run an initial random walk to get a score)
  $team$.add($\langle s_{rare}, e_i \rangle$)
  for $j \leftarrow 1$ **to** k **and** $j \neq rare$ **do**
    $resSet \leftarrow$ current members of $team$
    run a random walk with restart (RWR) to set $resSet$
    $p_j \leftarrow \{p \in E(s_j) | p$ gets highest score in the RWR$\}$
    $team$.add($\langle s_j, p_j \rangle$)
    $score_j \leftarrow$ the score of p_j in RWR
    $score \leftarrow score + score_j$
  end for
  if $score > maxTeamScore$ **then**
    $bestTeam \leftarrow team$; $maxTeamScore \leftarrow score$
  end if
end for
return $bestTeam$

The complexity of our algorithm is $O(|E(s_{rare})| \times k \times RWR_{time})$, where $|E(s_{rare})|$ is the number of rarest skill holders, k is the number of required skills, and RWR_{time} is the runtime of RWR. Note that RWR_{time} is linear when using the Monte Carlo approximation and is sub-linear in a distributed environment (as we will explain shortly).

2.3 Monte Carlo Approximation of RWR

RWR is traditionally computed via power iteration, which is prohibitively expensive for large graphs. However, there exists a Monte Carlo method that provides an accurate estimate to RWR with significantly lower runtime [3]. The Monte Carlo method simulates "surfers" traversing the graph at random, and does not require the entire graph as a $N \times N$ matrix in each iteration, where N is the number of nodes.

To estimate RWR for a single expert, we create N random walkers as ($source$, $currentPosition$) tuples, where the source and current position are initialized at the node in G that corresponds to the expert(s) (team members) we are are starting the walk from. The walkers follow the same parameters as the standard approach, i.e., each walker takes ns steps (or iterations), and at each step the walker has a probability β to restart on the $source$ node or a $(1-\beta)$ probability to continue walking, where it moves to a random node adjacent to $currentPosition$. The random walkers will continue to walk until termination when all steps are taken. The final distribution of walkers represent an accurate estimate of RWR for nodes. If a node is too far or receives no walkers, it is assigned a score of 0. As this is a Monte Carlo method, we can increase the accuracy of the approximation by increasing the number of random samples we perform. This is analogous to increasing the number of random walkers we start on each member of $E(s_{rare})$. This helps us to select an optimal number of walkers as the values stabilize beyond a certain threshold.

2.4 Distributed Algorithm

Each walker in our approach is independent and each walk can be parallelized. Below, we describe a parallel implementation of RW-Team using Apache Spark.

RDD Variables: *GraphRDD* is the given expert network which is divided into into k random partitions. Later, *GraphRDD* holds the partitions created by our proposed custom partitioning strategy. *WalkerRDD* contains the walkers with the same number of partitions as *GraphRDD*. Since these were created using the same partitioner, the partitions within $walkerRDD$ and $graphRDD$ correspond one-to-one and as a result become co-located. This helps us obtain iterators over both the $WalkerRDD$ and $GraphRDD$ partitions, therefore, we can iterate over the $walkerRDD$ partition and select the neighbors of a walker from the $GraphRDD$ partition iterator. The last variable is *TeamRDD* which keeps track of teams found by the algorithm.

Custom Partitioning: A challenge in parallelizing the team formation algorithm is that random partitioning is inefficient. This is due to the fact that to run random walkers over partitions, a random walker needs to have access to the data in another partition to take further steps. To avoid this, our proposed partitioner first distributes rarest skill holders across k partitions. Then, using a breadth first search strategy, all the neighbors of a rarest skill holder, and so on (up to a predefined threshold), are assigned to the same partition. (The threshold is a stopping point to avoid covering the entire graph for each rarest skill holder.) This increases the probability of visiting the next node of the walker in the same partition and avoids data shuffling.

Load Balancing: We use two strategies. Using our custom partitioner, to avoid overlap among partitions, we assign the neighbor expert to one of the desired partitions by selecting the one with the lowest cardinality/fewest experts. In this way, each partition traces almost similar search space size. Moreover, we limit the number of steps a random walker can take (as described in Section 2.3). Using these two strategies, the breadth of the search space is similar for different executors, and the depth of the search space is monitored.

For each required skill, excluding the rare skill, a *WalkerRDD* is created and the existing team members (initially the rarest skill holder) become the restart set for the walker. Each walker retrieves the list of neighbors from its *GraphRDD* and moves to one at random. If the walker has not reached the maximum steps it can take, and the current *GraphRDD* partition contains the data for its new position, the walker keeps walking. Otherwise, the walker waits to be shuffled. Once all walkers terminate, the ones corresponding to a specific team are located in the same partition. Then, the existing teams are updated and the process is repeated for the rest of the uncovered skills. At the end, teams in *TeamRDD* are sorted based on their scores and the best teams are discovered.

2.5 Visualization of Teams

Aside from computational complexity, another challenge in team formation is how to display the results. In addition to identifying

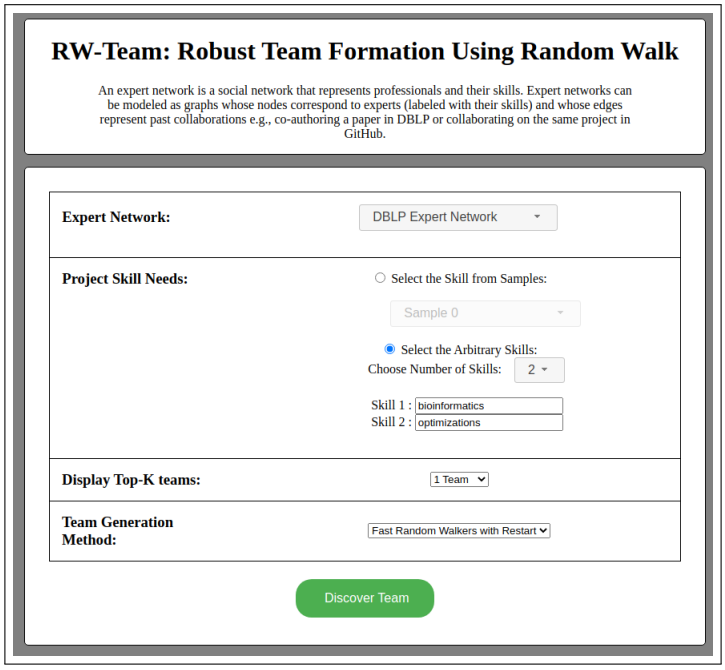


Figure 2: Front page of RW-Team interface.

the team members, we want to show how they are connected in the underlying network, especially when the chosen experts are not directly connected and would need to be introduced to each other via other individuals. RW-Team includes the following visualization options.

- **BFS**: We traverse the graph in breadth-first fashion, starting at each team member. When the first node that is connected to all team members is found, we stop. Then, all the nodes and edges that have been visited so far are presented to the user.
- **Pruned BFS**: This is similar to BFS. However, we only show the nodes with high RWR scores. Note that this visualization method cannot be used with previous approaches that measure distance using shortest paths since these approaches do not compute RWR scores.
- **Clique**: For each pair of team members, we display the nodes and edges on a shortest path between them. If there are multiple shortest paths, we select one at random. Note that some pairs might share some of their shortest paths with other pairs. We expect this method to generate a concise subgraph.
- **Clique + Neighbours**: We start with the output of the Clique method and we add the immediate neighbours of every node (which could be a team member or a node lying on the shortest path between two team members).

3 DEMONSTRATION OUTLINE

RW-Team features a Java web-based interface for interactive team formation in expert networks. On the front page (Figure 2), users enter the required skills and the dataset. On the next page, the teams produced by the selected ranking method are shown. We use BFS visualization by default, with the other three methods also available. Our graphical presentation is based on GraphViz (https://www.graphviz.org). Two datasets are used in this demo: DBLP and GitHub. DBLP[3] is a network of co-authors, in which each node corresponds to an author, and the expertise of an author is extracted from her publications' titles. Two authors are connected to each other if they publish a paper together. GitHub[4] is a web-based hosting service for a variety of IT projects. Nodes of the GitHub are IT professionals (e.g., developers), and they are connected to each other if they work on the same project together. Expertise of GitHub experts are taken from the titles of the projects they worked on.

The main objective of this demonstration is to allow users to interactively explore different ranking algorithms including ours based on random walk, and shortest path and embedding based, to form teams of experts. Furthermore, users will explore the underlying connections among team members and will see how a minimalist approach to revealing connections (e.g., Clique) might be different than a comprehensive approach (e.g., BFS). Participants will gain a better understanding of the factors affecting the search results in various datasets. Furthermore, demonstrating RW-Team will allow us to collect feedback from participants about our system, and on how to improve the search and team visualization experience. We will also provide a variety of pre-computed scenarios in which different ranking and visualization techniques produce different answers and different insights.

[3]https://dblp.org/
[4]https://github.com/

REFERENCES

[1] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. 2012. Online Team Formation in Social Networks. In *WWW*. 839–848.

[2] A. Anagnostopoulos, C. Castillo, A. Fazzone, S. Leonardi, and E. Terzi. 2018. Algorithms for Hiring and Outsourcing in the Online Labor Market. In *KDD*. 1109–1118.

[3] B. Bahmani, A. Chowdhury, and A. Goel. 2010. Fast Incremental and Personalized PageRank. *Proc. VLDB Endow.* 4, 3 (Dec. 2010), 173–184.

[4] S. Bryson, H. Davoudi, L. Golab, M. Kargar, Y. Lytvyn, P. Mierzejewski, J. Szlichta, and M. Zihayat. 2020. Robust keyword search in large attributed graphs. *Information Retrieval Journal* 23, 5 (2020), 502–524.

[5] A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.

[6] M. Kargar, M. Zihayat, and A. An. 2013. Finding Affordable and Collaborative Teams from a Network of Experts. In *SDM*. 587–595.

[7] T. Lappas, L. Liu, and E. Terzi. 2009. Finding a Team of Experts in Social Networks. In *KDD*. 467–476.

[8] L Li, H. Tong, N. Cao, K. Ehrlich, Y. Lin, and N. Buchler. 2015. Replacing the Irreplaceable: Fast Algorithms for Team Member Recommendation. In *WWW*. 636–646.

[9] S. B. Roy, L. Lakshmanan, and R. Liu. 2015. From Group Recommendations to Group Formation. In *SIGMOD*. 1603–1616.