Tackling Cold Start for Job Recommendation with Heterogeneous Graphs

Eric Behar[1,2,*], Julien Romero[1], Amel Bouzeghoub[1] and Katarzyna Wegrzyn-Wolska[3]

[1]Telecom SudParis, IPParis, SAMOVAR, Évry, France
[2]EasyPartner, France
[3]EFREI, Villejuif, France

Abstract

Recruiting changed drastically with the emergence of professional social networks that bring together many people and companies. It is a chance as it helps to increase the adequacy between a position and a candidate. However, it creates new challenges. First, the many possible combinations make it hard to find the perfect match. Second, it brings together talents with many different skills and backgrounds that can be hard to understand for a recruiter. In particular, in computer science, technologies tend to change quickly and can be obscure for non-technical employees. Therefore, using automatic tools is crucial to guide the recruiting process. More specifically, a recommender system that matches candidates with open positions can improve the overall satisfaction of all the agents in our system. Yet, job-matching data suffers from the cold start problem: Once a person gets a position, they are very unlikely to obtain a new one soon. Thus, traditional techniques based on collaborative filtering are very limited, and we must rely on the unique characteristics of each candidate.

In this paper, we propose a new recommender system based on a recruiting heterogeneous graph. This graph brings together information about a job posting and the personal knowledge graph of the candidates. We tested our model on a new real-world dataset, and we showed that it outperforms state-of-the-art methods.

Keywords

Recommender Systems, Recruiting, Cold Start

1. Introduction

According to a survey from 2017 [1], job seekers predominantly use online applications to find a job: 77% use company websites, 58% use job posting sites, and 47% use a professional social network. These numbers keep increasing, and COVID-19 enforced the trend [2].

Dealing with the high number of candidates and job postings is a challenge for all the actors in the recruiting ecosystem. The applicants cannot browse through millions of open positions and need to be recommended the most relevant ones for their skills and experience. Likewise, recruiters cannot look at all possible candidates on a professional social network. Besides, it might be hard for them to fully understand a technical market such as IT (Information Technology), where skills are constantly changing. They need to find quickly the best talents who are most likely to join the company. The process is crucial as it can become costly for the company ($4,129 on average [3]).

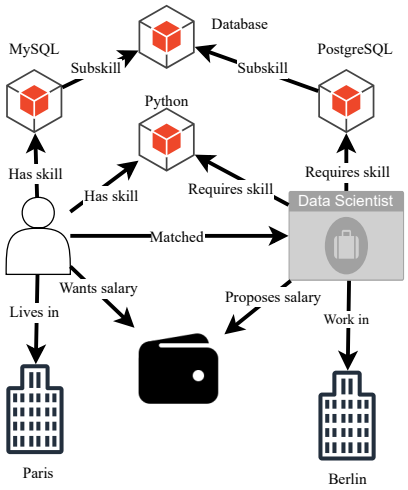

Figure 1: Example of a Recruiting Graph

Therefore, many recruiting companies power their websites with recommender systems that help to find the perfect match. However, the recruiting domain differs from traditional domains like movie or book recommendations. The main difference comes from the sparsity of

RecSys in HR'23: The 3rd Workshop on Recommender Systems for Human Resources, in conjunction with the 17th ACM Conference on Recommender Systems, September 18–22, 2023, Singapore, Singapore.
*Corresponding author.
eric.behar@telecom-sudparis.eu (E. Behar);
julien.romero@telecom-sudparis.eu (J. Romero);
amel.bouzeghoub@telecom-sudparis.eu (A. Bouzeghoub);
katarzyna.wegrzyn@efrei.fr (K. Wegrzyn-Wolska)
https://julienromero.fr (J. Romero);
https://amel.wp.imtbs-tsp.eu/ (A. Bouzeghoub)
0000-0002-7382-9077 (J. Romero); 0000-0003-4890-9005 (A. Bouzeghoub); 0000-0002-9776-3842 (K. Wegrzyn-Wolska)

training data: Once a candidate gets a job, they are unlikely to get another one soon and on the same platform (on average, a person holds 12.5 jobs in their lifetime [4]). Conversely, we expect an individual to have seen many series and movies, even on a single website. Therefore, we cannot use traditional collaborative filtering techniques and must rely on additional features.

One could only limit themselves to applications rather than successful recruitments. This approach has the advantage of creating more training data but also produces a lot of noise for the recruiters. Indeed, candidates usually apply to many positions, especially online. According to TalentWorks [5], one needs between 100 and 200 applications to get a job offer. From the recruiter's point of view, most of the applicants are irrelevant and, therefore, do not constitute valuable positive training data for us. Of course, if we had the feedback from the final recruiters, it would bring a high value, but this is not the case in existing datasets.

A further distinction in the recruiting domain is the strong presence of relevant features. When one creates an account on a streaming website, they do not fill in much personal information. As the number of training data is enough, straightforward collaborative filtering algorithms can produce good results. It is different for recruiting as a job or a candidate usually comes with information that helps both parties find the perfect match. For example, we often find the city, skills, experience, or education of a person.

These differences make the problem of job recommendation worth studying, especially in the case of cold start recommendations with semantic information. However, few datasets allow testing this configuration, so few proposed systems solve this problem.

This paper studies this specific problem and makes the following contributions:

- The creation of a dataset for job recommendation with strong semantic information.
- The building of a recruiting graph connecting the candidates with jobs.
- The implementation of a recommendation system based on the created recruiting graph.

We will start by introducing the relevant state-of-the-art methods in Section 2. Then, we will formally define our problem in Section 3. Section 4 presents our solution, and we give implementation details in Section 5. Finally, we compare our approach with the state-of-the-art in Section 6.

2. Previous Work

In this section, we introduce previous works. We divide them into two parts. First, we study the existing datasets and explain why they are too limiting or unsuitable in our case. Then, we look at existing recommender systems, particularly those that could work in a similar scenario.

2.1. Datasets

A crucial component of a recommender system is the dataset used to train it. We want to compare them regarding our problem, which is job recommendation with cold start and semantic information. We include in our study the following datasets:

- MovieLens (ML) 100k, 1M, 10M [6] is the most common dataset used for benchmarking in recommender systems. It can be found at multiple scales depending on the use. It comprises data about users (age, sex, occupation), movies (title, release date, genres), and the users' ratings that may be associated with a comment. It is essential to notice that the users' information is unavailable for MovieLens 10M.
- Gowala [7] is a location-based social networking website where users share their locations by checking in. It contains pairs of user and location identifiers associated with a timestamp and GPS coordinates.
- Yelp [8] is a website allowing users to review businesses and other locations. The dataset contains reviews and classic information about the business, like the name, address, categories, or opening hours.
- CareerBuilder's job recommendation challenge [9] is a dataset published for an online data science hackathon on Kaggle by CareerBuilder. This dataset is the one that shares the most similarity with our dataset.
- Our newly created dataset, JobTrackingHistory (JTH). Further details are provided in Section 4.1.

We compare these datasets according to several metrics. The standard ones are the size, the number of users, and the number of available features. Then, we introduce additional metrics to describe the cold start problem better. First, we have the density, which is the number of user-item associations divided by the total number of possible recommendations. This number highly correlates to the number of items and users. Therefore, we also include the average number of user and item associations. Finally, we report the number of users and items without an association and whether or not the dataset has a temporal dimension. The results are presented in Table 1.

From this table, we can see that our dataset stands out on two points. First, we have many users and items that

Name	ML-100K	ML-1M	ML-10M	Gowalla	Yelp	CareerBuilder	JTH
Domain	Entertainment	Entertainment	Entertainment	Social Network	Food Review	Recruitment	Recruitment
Size (approx)	27.3 MB	400 MB	300 MB	700 MB	8 GB	6.1 GB	2.7 GB
#users with reco	943	3,706	5,857	107,092	1,987,897	321,231	26,078
#items with reco	1,682	6,040	9,394	1,280,969	150,346	365,649	4,026
#features	10	10	10	3	27	12	37
Density	6.3%	4.47%	1.46%	0.0029%	0.0023%	0.0014%	0.04%
Avg #items/users	106	165.6	137.1	60.2	3.5	4.99	2.58
Avg #users/items	59.45	269.9	85.5	5	46.49	4.38	16.69
#users w/ reco	0	0	183	0	0	0	45,938
#item w/ reco	0	117	1287	0	0	0	771
Has time	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 1
Comparison of the Datasets For Recommender Systems

do not belong to any association. It creates a pool of potential candidates and jobs that we could exploit. Second, the number of relations a user engages in is below all other datasets, which stresses the cold start problem. Note here that we do not make a difference between a candidate application, a recruiter selection, and a successful recruitment (see Section 4.1). Compared to the CareerBuilder dataset, our dataset is much smaller. However, it contains better features and a finer granularity for the association between a candidate and a job. This fundamental difference will allow us to tackle the cold start problem.

2.2. Existing Architectures

In this section, we focus on existing systems for job recommendations. Although many systems are general, we discuss the particularities of recommending a job to a candidate.

2.2.1. Traditional Recommender Systems

The most popular systems are based on collaborative filtering [10, 11] in which the idea is to find similar users based on their shared items (or the opposite). Therefore, it performs poorly with sparse data and suffers from the cold start problem, i.e., it has difficulties making recommendations when a user has no or few interactions with other items. Some works [12, 13] try to include additional features in the process, but it generally requires a lot of work for feature engineering and encoding. Besides, due to the nature of the datasets, the features are rarely semantic.

Another category of algorithms is based on matrix factorization [14, 15, 16]. They also suffer from sparsity and cold start issues as they are using the interaction matrix.

2.2.2. Graph-based Recommender Systems

Homogeneous and heterogeneous graphs are the natural structures to include semantic information, leading to graph-based recommender systems [17, 18, 19, 20]. Most of them are based on Graph Neural Networks (GNN). Authors in [21] construct a knowledge graph for movie recommendation using DBpedia [22] and connect it to items only. Then, using Node2Vec [23] they compute embeddings for users and items and plug them into a classifier. Here, the representation of the entities is not directly linked to the recommendation. KTUP [24] is inspired by TransH [25]. It contains two components: A knowledge graph encoder (TransH) and a user preference generator. This last part needs to learn embeddings for all users and items solely from user-item interactions, which is impossible in our case due to the sparsity of data. KGAT is [26] is based on an attention network on top of a heterogeneous graph and was tested to recommend books, music, and places (Yelp). However, they just consider the case where only items have features and, therefore, connections to semantic nodes. This makes it possible to leverage general paths in the graph. In our case, we have a deeper and denser semantic network. HAGERec [27] makes similar assumptions as KGAT, with a network composed of hierarchical attention and convolution layers.

2.2.3. Job Recommender Systems

Job recommendation is less studied than other topics like books and movies because of the lack of data. In particular, we find the CareerBuilder and Xing dataset [28]. This last dataset cannot be accessed anymore. Both of these datasets have very little semantic information. The simplest algorithms build a traditional classifier on top of manually designed features [29]. Most approaches are variations of conventional collaborative filtering [30, 31]. In [32], the authors use a pure feature approach. They give an autoencoder the history of interactions with jobs and a few features on these jobs. In this paper, we are working on more connected data and still want to have a collaborative filtering part. [33] also uses a pure feature approach by building an embedding for each user-item pair based solely on the features. In [34], they tackle

the problem of cold start by using textual job descriptions. They aim for a new item to find an old, similar item and copy its interactions. However, if a new job category arrives (e.g., a new technology is introduced), then the system will perform poorly as there was no similar item before. [35] uses a graph-based approach for a job recommendation, but they have very little semantic information. [36] is a system that recommends jobs from a skill list. It first learns embeddings for each skill from the ESCO ontology and then computes a similarity with job postings. This system does not learn from training for the recommendation part. Besides, ESCO contains mostly high-level skills. We will show it is possible to complement it with other sources.

We also find other kinds of recommender systems in the job market. For example, [37] recommends useful skills to learn. They construct of graph of skills and apply topic-modelling techniques to make the recommendation depend on the context. However, their approach only considers job postings (no candidates), and the graph they use is a simple cooccurrence graph. Still, they investigate useful features for job postings. [38] focuses on skill recommendation from the candidate's point-of-view. They construct a skill ontology based on information mined on several websites. The job postings are not considerate. In our work, we use an existing ontology (ESCO) that we complete with an external knowledge base (Wikidata).

3. Problem Formulation

Heterogeneous Graph (or Knowledge Graph, KG) is a tuple $G = (V, R, E)$ where V is the set of all the nodes (or entities), R is the set of all relationships, and E is the set of edges (e_0, r, e_1) where $e_0 \in V$, $e_1 \in V$, and $r \in R$. Besides, each node is associated with a type that is a subset of V.

In our case, the nodes will be composed of types like candidates, job postings, skills, or salaries. The relations will encode semantic information like "has skill", "wants salary", "requires skill", or "was selected for job". Figure 1 shows an example of a heterogeneous graph.

Graph-Based Recommendation Given a heterogeneous graph $G = (V, R, E)$, a type U (the users or candidates), a type I (the items or jobs), and a relation $r \in R$, we want to predict for all $u \in U$ and all $i \in I$ if $(u, r, i) \in E$.

In our case, we are mainly interested in relationships between candidates and job postings that indicate a positive recruitment. Of course, when we make the prediction, we need to be careful not to use the original edge if it exists.

In this paper, we tackle the case of heterogeneous graph-based recommendation for job recommendation. The sparsity of this use case also raises the problem of the cold start, i.e., making recommendations for a user with no or very few interactions with other items.

4. Methodology

In this section, we introduce our approach. It comprises three parts: Creating a job recommendation dataset, creating the heterogeneous graph enriched with semantic information, and applying the recommendation algorithm to this graph.

4.1. Job Tracking History Dataset

Our dataset was constructed from a real-life IT recruitment system containing three main types of entities. The first one is the candidates who are looking for a job. The second one is job positions that need to be filled. The last one is recruiters, whose primary role is to match candidates with positions. Once this matching is done, the company behind the job posting can choose to interview the candidates and potentially recruit them. Therefore, we do not have a binary association between users and items but rather a gradual score. In comparison with the CareerBuilder dataset, we have a less noisy dataset. In their case, an interaction is simply when a candidate applies for a job, whereas in our case, at least a professional recruiter validated the match.

To help the recruiter in this process, semantic information surrounds the candidates and the job postings. For the candidate, we have the resume from which skills are extracted, the current position, the current salary, the experience (in years), the asked salary, the area of expertise, the source (the system finds candidates through various websites), the desired type of contract, and the location. For the job postings, we have the company that emitted it, the type of contract, the area of expertise required, a description, the required skills, the proposed salary, the date of emission, and the location. Most features are manually entered into the system by expert recruiters. The data also comes with the timestamp of significant events (creation of a posting, date of an interview, creation of a profile).

We call our final dataset Job Tracking History (JTH). Statistics about it can be found in Table 1.

4.2. Recruiting Graph

From JTH, we can start building our recruiting graph. It comprises eleven kinds of nodes: the candidates, job postings, skills, salaries, years of experience, recruiters, companies, areas of expertise, employment types, localization, and candidate source. The relations are the ones described in Section 4.1. For candidates, job postings,

and skills, we also add a feature vector corresponding to the embeddings of the resume, job description, and skill name obtained with a sentence encoder built on top of MiniLM [39]. For some nodes representing continuous values (salaries, years of experience, location), we created nodes representing value ranges.

Then, we enhanced the semantic information for skills using two external knowledge bases: ESCO [40] (European Skills, Competences, Qualifications, and Occupations) and Wikidata [41]. ESCO is a European taxonomy of skills, competencies, and occupations. We only use the skill hierarchy from this taxonomy, i.e., how the skills are classified. As ESCO is limited regarding IT-specific skills (libraries, platforms), we augmented it using information extracted from Wikidata. In this new taxonomy, a skill can be associated with several labels. We use them to merge previous skill names (e.g., Javascript and JS).

Figure 1 shows an example of our recruiting graph.

4.3. Job Recommendation System

In this paper, we will focus on predicting if a link between a candidate and a job posting exists, even if we have a finer granularity (matched, interviewed, selected). Each node in our graph is associated with an embedding. For nodes with features, the embedding combines the graph embedding and the feature vector through a linear layer.

Given a candidate c and a job posting j, we start by sampling a subgraph centered on u and j. To do so, we follow [42] by sampling interactively a certain number of neighbors given previously known nodes. This step is crucial as running a GNN on the entire graph would be impossible in a reasonable time.

Then, we pass our subgraph into a multilayer graph convolutional network (GCN) following a similar architecture as [42]. Initially, this architecture only worked for homogeneous graphs, so we used the transformation presented in [43]. The idea of this transformation is to duplicate the original network for each relation and then recombine the obtained representations. After the GNN layers, we receive a vector representation for u and i. We take the dot product of the two to make the prediction. We used the standard cross-entropy loss to evaluate the performance of the classifier. The architecture is presented in Figure 2. We call our final model RecruiterGCN.

5. Experiment Setup

5.1. Implementation

We used Python and the PyTorch Geometric library [44]. For the implementation of the baselines, we used LibRecommender [45]. We split our dataset into train, validation, and test sets following the proportion 0.8/0.1/0.1. We ran the experiments on a computer with a processor at 2.2GHz and 14 cores and an NVIDIA Tesla V100 GPU with 32 GB of RAM. We stop the training phase following an early stop mechanism on the loss function. In total, for a given set of hyperparameters, the computation time was around five hours.

For the hyperparameters, we vary the number of layers, the number of neighbors used during sampling, the learning rate, and the negative sampling rate. We found the best results with six layers, a learning rate of 0.0001, a weight decay of 0.001, a two-stage sample with first twenty neighbors and then ten neighbors, and the creation of one random negative sample for each positive sample.

Our final code is available on GitHub github.com/EricPoulet/RecSysInHR2023.

5.2. Baselines

We include a large variety of algorithms in our baselines. First, we have traditional collaborative filtering approaches. UserCF [11] is a user-based collaborative filtering based on the interaction matrix. Item2Vec [13] is an item-based collaborative filtering algorithm that learns embeddings for each item. ALS [16] is a matrix factorization algorithm. YouTubeRanking [12] is an approach that emphasizes external features more strongly. We manually create a feature vector that, for a given user and item, is composed of the number of common skills, if they have the same expertise area, the kind of contract, the source of the candidate, the years of experience, the salary, the distance to go to the job, and the cosine similarity between the resume and the job description. AutoInt [46] is another system that also leverages user-item features.

For the graph-based approaches, we used LightGCN [47], which takes the interaction graph (only candidates and jobs) as input and is based on a GCN. GraphSage [48] is a more advanced variation that takes similar input but puts more emphasis on sampling.

5.3. Metrics

In this paper, we report four metrics traditionally used in recommender systems: the mean precision at K (P@K), the mean recall at K (R@K), the mean average precision at K (MAP@K), and the normalized discounted cumulative gain at K (NDCG@K).

6. Results

6.1. Comparison With Baselines

Table 2 compares our approach with the baselines. We can see that RecruiterGCN beats all baselines for the pre-

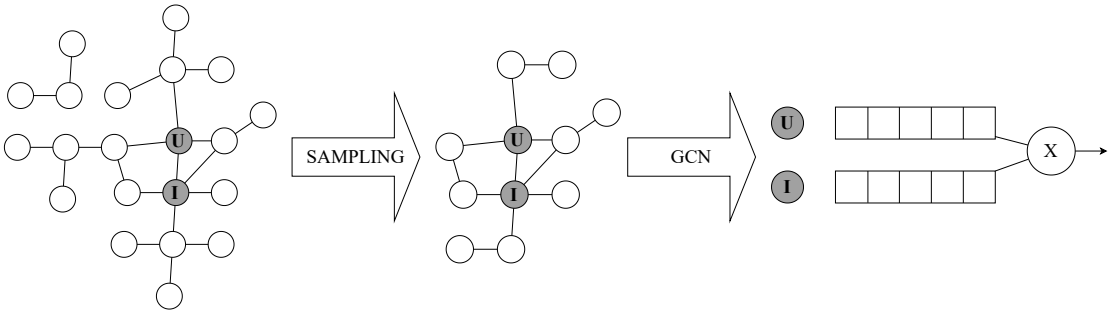

Figure 2: The Multilayer Graph Network Architecture For Job Recommendation

.

Algorithm	P@10	R@10	MAP@10	NDCG@10
RecruiterGCN	**0.0175**	**0.159**	0.0505	0.051
UserCF [11]	0.0111	0.0917	**0.0534**	**0.0826**
Item2Vec [13]	0.0097	0.0801	0.0343	0.0471
GraphSage [48]	0.002	0.0157	0.0045	0.0079
LightGCN [47]	0.0108	0.0926	0.0463	0.0597
AutoInt [46]	0.001	0.0075	0.0032	0.0047
YouTubeRanking [12]	0.0005	0.0041	0.001	0.002
ALS [16]	0.0048	0.0401	0.0253	0.0303

Table 2
Comparison of RecruiterGCN With the Baselines

cision at K and the recall at K, but not for MAP@K and NDCG@K. These two last metrics emphasized the order of the recommendations, showing that our model might not be able to rank the top recommendations correctly. Still, if the order is unimportant (a recruiter can easily post-process the ten suggestions), our algorithm brings extra value compared to the competitors. In particular, it improves over the other graph-based baselines. Regarding the model, it remains close to them in structure, which shows the importance of semantic information.

The simple user-based collaborative filtering algorithm works surprisingly well when we look at the baselines. This is not intuitive as the cold start problem should disadvantage it. However, looking more into the data, we notice that recruiters like to cluster similar candidates and match them to identical job postings. Therefore, it biases the results as knowing the group of a user makes the predictions easier. This suggests we should use a more advanced sampling mechanism and train/validation/test split based on the date and by ignoring some types of nodes.

The results are disappointing when we look at feature-based baselines (AutoInt and YoutubeRanking). We had to manually translate the semantic information into a continuous vector for them. Therefore, the features engineering can be long and fastidious. On the contrary, semantic information is naturally represented with a graph with minimal human actions. RecruiterGCN automatically extracts relevant features and integrates them into the final results. Besides, it is more flexible as adding features does not require more feature engineering.

Setup	P@10	R@10	MAP@10	NDCG@10
All - Skill	0.0175	0.1592	0.0505	0.051
All - Salary	0.0173	0.1568	0.0511	0.0514
All	0.0163	0.1487	0.05	0.0504
All - Company	0.0161	0.1466	0.0514	0.0518
All - Origin	0.016	0.1468	0.0515	0.0517
All - Zip	0.0159	0.1451	0.0495	0.0496
All - Contract	0.0156	0.1429	0.0461	0.0462
All - Recruiter	0.0152	0.1389	0.045	0.0451
All - Experience	0.0142	0.1296	0.0404	0.0405
All - Concept	0.0136	0.1245	0.0377	0.038
All - Category	0.0094	0.0843	0.0246	0.0247
Job + Candidate	0.0043	0.0389	0.016	0.016

Table 3
Ablation Study - Ranked Results by P@10

6.2. Ablation Study

We performed an ablation study to evaluate the impact of each semantic information in our graph. For each type of node (except candidates and jobs), we remove it from the recruiting graph and rerun the experiments on it. The study results are presented in Table 3.

As expected, the graph without semantic information (Job + Candidate) gets poor results. However, the entire graph (All) does not get the best results. There can be three main explanations for that fact. First, we might have noise in the evaluation that can cause slight variations. Second, we remove potential noise in the graph by eliminating some nodes. Therefore, the network might be able to reach a better optimum. Third, because of the sampling strategy, we might add additional noise if we have too many semantic nodes. Here, we call "noise" the presence of too much information or irrelevant information in the graph for the final recommendation.

The best setup is obtained with no skill in the graph, which is a bit surprising. In fact, we still have general skills with the category nodes that are areas of expertise manually entered by recruiters. When we remove the skills, we remove some noise, and the network can better focus on other nodes. We can see that by eliminating areas of expertise (All - Category), we deteriorate the results deeply. We also notice decreased performance when we remove the skill hierarchy (All - Concept). This goes in a similar direction as the area of expertise: What is essential are general skills (e.g., databases) rather than specific skills (e.g., MySQL).

We notice that continuous values are hard to exploit for the network: Removing salaries (All - Salary) improves the performance, and removing the location (All - Zip) does not have much impact. However, these two factors should be crucial. As we used a graph structure, we had to discretize the values into ranges. We lost ordinal and comparative information in the process, making the end nodes useless.

Some nodes do not seem to have much impact. Removing the company (All - Company) does not change the results. This is expected, and no prior reason exists for a company to recruit a random candidate. Likewise, the candidate's origin does not change the results (All - Origin). It shows the quality of the candidates does not depend on the website where the recruiter found them.

Three other kinds of nodes have a negative influence on the results. First, the type of contract (All - Contract), which is logical as both the company and the candidate are generally stringent on this point. Second, the recruiter who found the candidate (All - Recruiter). This shows that some recruiters might be better at finding good candidates. Our recommender system could help junior recruiters to improve their skills. Finally, years of experience are crucial (All - Experience). This is also understandable, as a candidate with more experience is likelier to get a position.

6.3. Limitations And Future Works

The comparison with the baselines and the ablation study raised problems that can lead to thrilling future works. First, we saw that splitting the dataset into train, validation, and test sets is an essential step for preventing biases introduced by the recruiters. The temporal aspect has a huge impact, although it is less studied in the literature. Likewise, the sampling strategies must be improved to include time and prevent too much noise in the network.

Continuous values can be tricky to use. We must find a way to encode distances or cardinal orders between nodes to keep the standard graph representation. Otherwise, we should adjust the message-passing algorithm used in the GCN to have a special treatment for these nodes. In particular, we need a way to include the temporal dimension in the graph as it is crucial for recruitment: We cannot suggest an old job posting to a new candidate as it is very likely it is already filled.

Next, we ignored the granularity of the interactions between a candidate and a job, whereas it could help to reward top recommendations that lead to a new job rather than a simple match. In the future, we must include this granularity during training and testing. Therefore, we must adapt the loss functions and the evaluation metrics.

Finally, as our system is supposed to assist one of the actors of our system (candidate, company, or recruiter), we need to work on how to present the recommendation. Notably, we must be able to explain the recommendations using the semantic information in the graph.

7. Conclusion

In this paper, we studied the problem of job recommendation. We leveraged a new human-annotated dataset containing semantic information. We showed this information can be translated into a heterogeneous graph without much manual feature engineering traditionally used in other systems. Then, we applied a graph neural network to produce relevant recommendations. We showed that our system, RecruiterGCN, beats the state-of-the-art methods to isolate the top recommendations but still lacks more precise ranking capabilities. Our ablation study revealed that there are still points of improvement, particularly regarding the inclusion of time in our model. Our final code is available on GitHub github.com/EricPoulet/RecSysInHR2023.

Acknowledgments

We thank EasyPartner for providing the data for this project. We also thank the reviewers for their feedback. Our work would not have been possible without the resources provided by Lab-IA.

References

[1] I. Gallup, State of the american workplace, 2017. URL: http://www.gallup.com/reports/199961/state-american-workplace-report-2017.aspx.

[2] F. B. Inside, Online recruitment market size, share & covid-19 impact analysis, 2020. URL: https://www.fortunebusinessinsights.com/online-recruitment-market-103730.

[3] S. for Human Resource Management, 2016 human capital benchmarking report, 2016. URL: https://www.shrm.org/resourcesandtools/business-solutions/documents/human-capital-report-all-industries-all-ftes.pdf.

[4] U. B. of Labor, Labor statistics, 2020. URL: https://www.bls.gov/opub/mlr/2020/.

[5] TalentWorks, Science of the job search, https://web.archive.org/web/20190322214104/http://talent.works/blog/category/science-of-the-job-search, 2019.

[6] F. M. Harper, J. A. Konstan, The movielens datasets: History and context, ACM Trans. Interact. Intell. Syst. 5 (2015). URL: https://doi.org/10.1145/2827872. doi:10.1145/2827872.

[7] E. Cho, S. A. Myers, J. Leskovec, 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, Association for Computing Machinery, New York, NY, USA, 2011, p. 1082–1090. URL: https://doi.org/10.1145/2020408.2020579. doi:10.1145/2020408.2020579.

[8] N. Asghar, Yelp dataset challenge: Review rating prediction, arXiv preprint arXiv:1605.05362 (2016).

[9] CareerBuilder, Job recommendation challenge, https://www.kaggle.com/c/job-recommendation, 2012.

[10] J. B. Schafer, D. Frankowski, J. Herlocker, S. Sen, Collaborative filtering recommender systems, in: The adaptive web: methods and strategies of web personalization, Springer, 2007, pp. 291–324.

[11] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: Proceedings of the 10th international conference on World Wide Web, 2001, pp. 285–295.

[12] P. Covington, J. Adams, E. Sargin, Deep neural networks for youtube recommendations, in: Proceedings of the 10th ACM conference on recommender systems, 2016, pp. 191–198.

[13] O. Barkan, N. Koenigstein, Item2vec: neural item embedding for collaborative filtering, in: 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2016, pp. 1–6.

[14] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer 42 (2009) 30–37.

[15] Y. Koren, Factorization meets the neighborhood: a multifaceted collaborative filtering model, in: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, 2008, pp. 426–434.

[16] Y. Hu, Y. Koren, C. Volinsky, Collaborative filtering for implicit feedback datasets, in: 2008 Eighth IEEE international conference on data mining, Ieee, 2008, pp. 263–272.

[17] S. Wu, F. Sun, W. Zhang, X. Xie, B. Cui, Graph neural networks in recommender systems: a survey, ACM Computing Surveys 55 (2022) 1–37.

[18] Q. Guo, F. Zhuang, C. Qin, H. Zhu, X. Xie, H. Xiong, Q. He, A survey on knowledge graph-based recommender systems, IEEE Transactions on Knowledge and Data Engineering 34 (2022) 3549–3568. doi:10.1109/TKDE.2020.3028705.

[19] S. Wang, L. Hu, Y. Wang, X. He, Q. Z. Sheng, M. A. Orgun, L. Cao, F. Ricci, P. S. Yu, Graph learning based recommender systems: A review, 2021. arXiv:2105.06339.

[20] S. Wu, F. Sun, W. Zhang, X. Xie, B. Cui, Graph neural networks in recommender systems: A survey, ACM Comput. Surv. 55 (2022). URL: https://doi.org/10.1145/3535101. doi:10.1145/3535101.

[21] E. Palumbo, G. Rizzo, R. Troncy, Entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation, in: Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 32–36. URL: https://doi.org/10.1145/3109859.3109889. doi:10.1145/3109859.3109889.

[22] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, Dbpedia: A nucleus for a web of open data, in: international semantic web conference, Springer, 2007, pp. 722–735.

[23] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[24] Y. Cao, X. Wang, X. He, Z. Hu, T.-S. Chua, Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences, in: The World Wide Web Conference, WWW '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 151–161. URL: https://doi.org/10.1145/3308558.3313705. doi:10.1145/3308558.3313705.

[25] Z. Wang, J. Zhang, J. Feng, Z. Chen, Knowledge graph embedding by translating on hyperplanes, in: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI'14, AAAI Press, 2014, p. 1112–1119.

[26] X. Wang, X. He, Y. Cao, M. Liu, T.-S. Chua, Kgat: Knowledge graph attention network for recommendation, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 950–958. URL: https://doi.org/10.1145/3292500.3330989. doi:10.1145/3292500.3330989.

[27] Z. Yang, S. Dong, Hagerec: Hierarchical attention graph convolutional network incorporating knowledge graph for explainable recommendation, Knowledge-Based Systems 204 (2020) 106194. URL: https://www.sciencedirect.com/science/article/pii/S0950705120304196. doi:https://doi.org/10.1016/j.knosys.2020.106194.

[28] F. Abel, A. Benczúr, D. Kohlsdorf, M. Larson, R. Pálovics, Recsys challenge 2016: Job recommendations, in: Proceedings of the 10th ACM Conference on Recommender Systems, RecSys '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 425–426. URL: https://doi.org/10.1145/2959100.2959207. doi:10.1145/2959100.2959207.

[29] P. K. Roy, S. S. Chowdhary, R. Bhatia, A machine learning approach for automation of resume recommendation system, Procedia Computer Science 167 (2020) 2318–2327. URL: https://www.sciencedirect.com/science/article/pii/S187705092030750X. doi:https://doi.org/10.1016/j.procs.2020.03.284, international Conference on Computational Intelligence and Data Science.

[30] S. Yang, M. Korayem, K. AlJadda, T. Grainger, S. Natarajan, Combining content-based and collaborative filtering for job recommendation system: A cost-sensitive statistical relational learning approach, Knowledge-Based Systems 136 (2017) 37–45. URL: https://www.sciencedirect.com/science/article/pii/S095070511730374X. doi:https://doi.org/10.1016/j.knosys.2017.08.017.

[31] R. Mishra, S. Rathi, Efficient and scalable job recommender system using collaborative filtering, in: ICDSMLA 2019: Proceedings of the 1st International Conference on Data Science, Machine Learning and Applications, Springer, 2020, pp. 842–856.

[32] E. Lacic, M. Reiter-Haas, D. Kowald, M. Reddy Dareddy, J. Cho, E. Lex, Using autoencoders for session-based job recommendations, User Modeling and User-Adapted Interaction 30 (2020) 617–658.

[33] J. Zhao, J. Wang, M. Sigdel, B. Zhang, P. Hoang, M. Liu, M. Korayem, Embedding-based recommender system for job to candidate matching on scale, arXiv preprint arXiv:2107.00221 (2021).

[34] J. Yuan, W. Shalaby, M. Korayem, D. Lin, K. AlJadda, J. Luo, Solving cold-start problem in large-scale recommendation engines: A deep learning approach, in: 2016 IEEE International Conference on Big Data (Big Data), IEEE, 2016, pp. 1901–1910.

[35] W. Shalaby, B. AlAila, M. Korayem, L. Pournajaf, K. AlJadda, S. Quinn, W. Zadrozny, Help me find a job: A graph-based approach for job recommendation at scale, in: 2017 IEEE international conference on big data (big data), IEEE, 2017, pp. 1544–1553.

[36] A. Giabelli, L. Malandri, F. Mercorio, M. Mezzanzanica, A. Seveso, Skills2job: A recommender system that encodes job offer embeddings on graph databases, Applied Soft Computing 101 (2021) 107049.

[37] T. Xu, H. Zhu, C. Zhu, P. Li, H. Xiong, Measuring the popularity of job skills in recruitment market: A multi-criteria approach, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'18/IAAI'18/EAAI'18, AAAI Press, 2018.

[38] A. Gugnani, V. K. R. Kasireddy, K. Ponnalagu, Generating unified candidate skill graph for career path recommendation, in: 2018 IEEE International Conference on Data Mining Workshops (ICDMW), IEEE, 2018, pp. 328–333.

[39] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, M. Zhou, Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, Advances in Neural Information Processing Systems 33 (2020) 5776–5788.

[40] ESCO, Esco: European skills, competences, qualifications and occupations, esco.ec.europa.eu, 2023. Accessed: 2023-08-01.

[41] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (2014) 78–85.

[42] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, Advances in neural information processing systems 30 (2017).

[43] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15, Springer, 2018, pp. 593–607.

[44] M. Fey, J. E. Lenssen, Fast graph representation learning with pytorch geometric, arXiv preprint arXiv:1903.02428 (2019).

[45] LibRecommender, Librecommender, librecommender.readthedocs.io, 2023. Accessed: 2023-08-01.

[46] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, J. Tang, Autoint: Automatic feature interaction

learning via self-attentive neural networks, in: Proceedings of the 28th ACM international conference on information and knowledge management, 2019, pp. 1161–1170.

[47] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, M. Wang, Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020. arXiv:2002.02126.

[48] W. L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, 2018. arXiv:1706.02216.