TIMBRE: Efficient Job Recommendation On Heterogeneous Graphs For Professional Recruiters

Behar Éric, Julien Romero, Amel Bouzeghoub, Katarzyna Wegrzyn

▶ **To cite this version:**

Behar Éric, Julien Romero, Amel Bouzeghoub, Katarzyna Wegrzyn. TIMBRE: Efficient Job Recommendation On Heterogeneous Graphs For Professional Recruiters. 23rd IEEE/WIC International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) - 23e Conférence internationale IEEE/WIC sur l'intelligence Web et la technologie des agents intelligents, 2024, 2024. hal-04777174

HAL Id: hal-04777174
https://hal.science/hal-04777174v1

Submitted on 15 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

TIMBRE: Efficient Job Recommendation On Heterogeneous Graphs For Professional Recruiters

Éric Behar
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris
91120 Palaiseau, France
eric.behar@telecom-sudparis.eu

Julien Romero
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris
91120 Palaiseau, France
julien.romero@telecom-sudparis.eu

Amel Bouzeghoub
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris
91120 Palaiseau, France
amel.bouzeghoub@telecom-sudparis.eu

Katarzyna Wegrzyn-Wolska
EFREI, 75000 Paris, France
katarzyna.wegrzyn@efrei.fr

***Abstract*—Job recommendation gathers many challenges well-known in recommender systems. First, it suffers from the cold start problem, with the user (the candidate) and the item (the job) having a very limited lifespan. It makes the learning of good user and item representations hard. Second, the temporal aspect is crucial: We cannot recommend an item in the future or too much in the past. Therefore, using solely collaborative filtering barely works. Finally, it is essential to integrate information about the users and the items, as we cannot rely only on previous interactions. This paper proposes a temporal graph-based method for job recommendation: TIMBRE (Temporal Integrated Model for Better REcommendations). TIMBRE integrates user and item information into a heterogeneous graph. This graph is adapted to allow efficient temporal recommendation and evaluation, which is later done using a graph neural network. Finally, we evaluate our approach with recommender system metrics, rarely computed on graph-based recommender systems.**

***Index Terms*—recommender systems, knowledge graph, job recommendation, temporal.**

I. Introduction

Today's job market is extremely dynamic and competitive, particularly in the IT sector. One consequence is the multiplication of applicants for each position opening [1], leading to a heavy workload for companies. Therefore, they often decide to externalize the process to specialized recruiting firms that can handle many candidates and a wide range of skills in the market. Still, even in these firms, the recruiters have to filter many applicants, leading to two behaviours. First, they resort to automatic ATS (Applicant Tracking System) software that parses the resumes and filters the candidates using simple rules like keyword matching. Second, they often prefer to reverse the process by directly head-hunting good candidates and gathering relevant information about them. Then, when they receive a new job opening, they first look in their database.

Many works propose building recommender systems to assist with the recruiting process [2]–[7]. However, most of them use direct applications from the candidates as training data, leading to noisy input. Very few works were trained on real-life data annotated by professional recruiters. Yet, this kind of recruiter-oriented recommender system can greatly impact the productivity of recruiting firms. Building recommender systems for job openings encounters many challenges that are also present in other kinds of recommendations but are often emphasized in this case. First, the **cold start problem** is recurring in almost all recommendations and is the focal point of many work on recommender systems in the literature [8]–[12]. The candidates and the job openings have a **short lifespan** of a few weeks. Therefore, we have few training points and cannot rely on previous interactions. On the contrary, for more mainstream recommendations like movie recommendations, we generally assume that both the user and the item are here to stay for quite some time.

Second, because of this limited lifespan, the **temporal dimension is crucial**. Given a candidate (or a job), we cannot recommend an item in the future (mainly a problem during training) or too far in the past (as the position is likely to be already filled). Therefore, we need to adapt our representations to embrace the dynamicity of the data and not bias the results, particularly during training. The temporal component was often studied in the literature [13]–[16], but often with an assumption that does not hold in our case: User preferences evolve through time. This assumption makes sense when recommending a book or a movie but not for job applications because of the limited lifespan of the user and the item. Therefore, there is **no need to model the temporal evolution of user and item** representations, as it will only lead to more noise. Finally, as we cannot rely enough on previous interactions, we must strongly emphasize **additional information** about the users and the items. This means that we need to extract information from various sources and structure them in a way that is exploitable by a recommender system.

To solve these challenges, we introduce ***TIMBRE*** (Temporal Integrated Model for Better REcommendations), a temporal graph-based recommender system. TIMBRE first ingests data from multiple sources (resumes, job descriptions, recruiter notes, external knowledge bases) and structures them into a **unified heterogeneous graph**. Then, it adapts this graph to facilitate temporal recommendation. Next, it runs a graph neural network (GNN) to generate a score for a user-item pair. The particularity of this GNN is that it emphasizes the

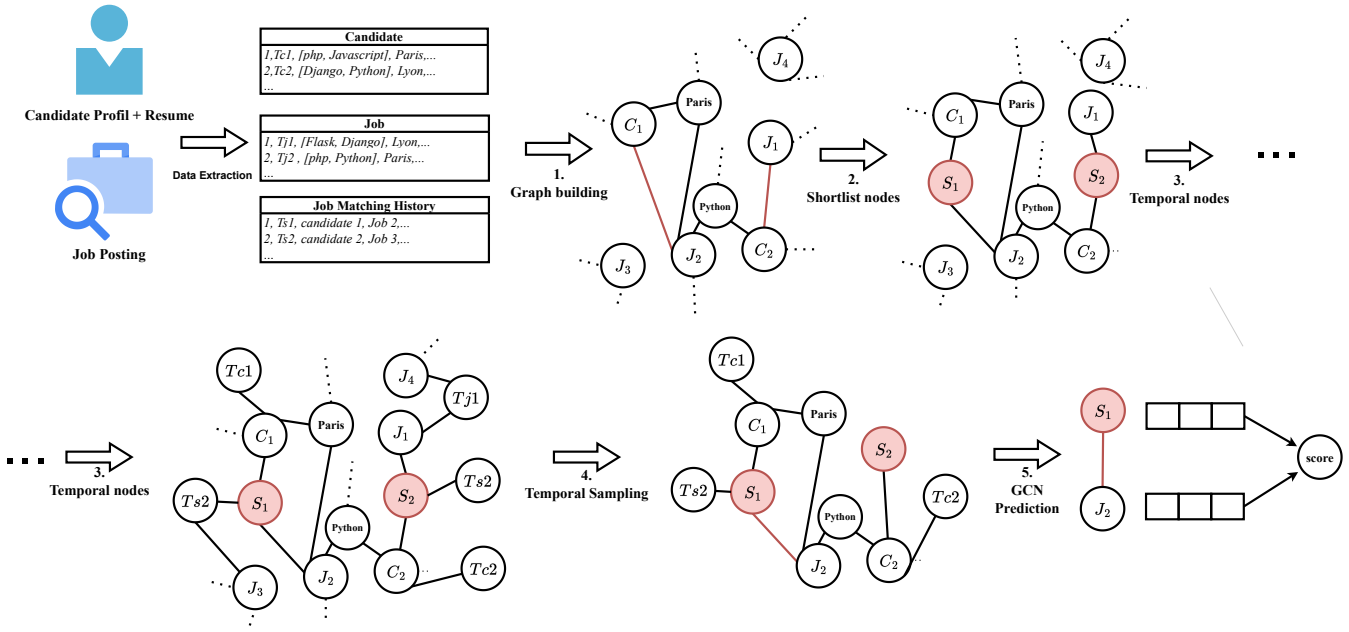

Fig. 1. Our complete job recommender system pipeline. 1. We turn our input data into a heterogeneous graph. 2. We replace relations between candidates and jobs with a shortlist node. 3. We add temporal nodes. 4; We apply our temporal sampling algorithm. 5. We apply a graph convolution network and make a prediction based on the representations of the shortlist node and the job node.

graph's structural aspect rather than the node's representation to counter the effect of the cold start. The GNN is trained by considering the temporal dimension to avoid training bias (mainly from the recruiters [17]).

In the end, we propose to evaluate our approach using traditional recommender system metrics. Although this should be automatic, most of the literature on graph-based recommendations ignores them as they are difficult to compute. Instead, they prefer metrics based on negative sampling, which are far from reliable. We implemented our evaluation on several baselines and compared them with TIMBRE, showing a clear advantage for our approach in the case of job recommendation.

To summarize, our contributions are the following:

1) Extraction and representation of information into a unified temporal heterogeneous graph.
2) Graph adaptation for a temporal recommendation through reifying the temporal interaction and introducing temporal nodes.
3) Time-dependent training and evaluation of our GNN using a sampling method boosting collaborative filtering.
4) Evaluation of a graph-based solution with recommender system metrics.

II. Previous Work

a) Job Recommendation: Many works in the literature tackle job recommendation as a user-item recommendation scenario and thus use a collaborative filtering approach where a user gets recommended the jobs of similar users [18]–[21]. Even though more and more data are produced, they remain primarily inaccessible to private research due to ethical concerns, privacy laws, and strategic concerns. Some effort has been made to provide an anonymized dataset, such as the Xing dataset [22], but it is now unavailable. Some works [23], [24] include additional features during the recommendation; however, they often require a lot of engineering. Some approaches emphasize helping the human recruiter review the candidate's profile by performing resume screening using natural language processing [25]–[27] or develop resume parsing framework [28]–[30]. These methods have two limitations. First, they only work if you have a specific job with a pre-selected pool of candidates. This does not work for proactive job recommendations such as headhunting. Secondly, it implies that the information in the resume is accurate. In our context, candidates' profiles are composed of a resume and information collected by an expert recruiter during an interview with the candidate. We must also mention some recent work shows promising results to either improve part of a recommender system or let a large language model (LLM) decide on job-candidate match [31]–[33]. The results are, however, limited to small-scale scenarios, as properly ingesting thousands of resumes is a challenging task for the current LLM frameworks.

b) Graph-Based Recommendation: Graph-based approaches to recommendations are equivalent to the link prediction task: Given a graph, we want to predict whether there will be a connection between a user and an item. The advantage of homogeneous and heterogeneous graphs is that they can represent connected data and semantic information that can be used to make recommendations [34]–[37] using graph neural networks (GNN), even in the case of jobs [38]. However, the construction of the graph can be problematic due to the absence or few numbers of features [39], [40], which sometimes leads to using an external knowledge base like DBpedia for famous entities [41]. Concerning the architecture, we find variations on top of graph convolutional networks [17],

[35], [37] and graph attention networks [39], [40].

c) Temporal Recommendation: In many applications, it is crucial to model the change in user preferences for long-term and short-term modifications [42], [43]. In the literature, several techniques are used to model the user representation change through time. We can cite those using latent Dirichlet allocation [44], deep learning techniques [45], reinforcement learning [46], matrix factorization [47], recurrent neural networks (RNN) [48], or Markov chains [49]. We find similar techniques for temporal graph recommendations. The Neighborhood-Aware Temporal network [50] (NAT) stores the temporal modifications of the neighbor of a node in a dictionary and then uses an RNN to make the representation of a node evolve. Temporal Graph Network (TGNs) [14] makes the embeddings of each node evolve through time using a graph attention network and an indication of the time delta since the previous interaction. A variant of the temporal recommendation is the sequential recommendation, where the goal is to predict the next interaction [51]–[53]. However, this setup mostly disregards the time of the interactions and can only make predictions with enough previous interactions (often at least three), which avoids the problem of the cold start.

d) Temporal Graph Neural Networks: Many applications consider the temporal dimension in a graph, mainly through the representation of an event stream as a graph. For spatio-temporal events, the temporal dimension is either used to adjust a distance function defining the neighbors of a node [54], [55] and to create graph snapshots that contain all the events in a time window [56].

e) Datasets for temporal recommendation with side information: Some public datasets exist for temporal recommendation [57]–[59]. However, due to privacy limitations, they often come with very limited side information, especially for the users. Besides, they often supposed that most users and items have enough recommendations to make relevant recommendations. However, this is not the case for job recommendations, which makes it necessary to develop new techniques that can better balance external information and interactions.

III. Problem

a) Background: In this paper, we will consider heterogeneous graphs. A heterogeneous graph G is defined by a tuple (V, R, D, E, Ω) where V is a finite set of nodes, R is a finite set of relationships, D is a set of types, $E \subset V \times R \times V$ is the set of edges, and $\Omega \subset V \times D$ is the set of types associated to a node. We can associate properties with the nodes or edges of a heterogeneous graph using a function $P(v_1)$ or $P(v_1, r, v_2)$ where $v_1 \in V$, $v_2 \in V$, and $r \in R$. In this paper, we only consider properties for nodes. We talk about a **temporal heterogeneous graph** when the property represents temporal information. This information generally represents the date of a node or edge creation.

b) Our Problem: This paper uses a dataset of users (candidates) U and items (job postings) I. We suppose we can access a textual document for each user and item. In practice, the document will be a resume, information from recruiters for the candidates, and a job description for a job posting. Then, the dataset contains a set of N interactions $(u, i, t) \in U \times I \times T$ where T represents the timestamps of the interactions (e.g., a POSIX time). In the real world, an interaction means that a user u was selected for a job i by a recruiter at time t. We call the selection of a candidate **shortlisting**.

The problem we tackle is the following: Given a user $u \in U$ and a time $t \in T$, we rank all the items $i \in I$ such that the higher the rank, the better the recommendation at time t.

In our case, as we will explain in Section IV-A, we represent a user $u \in U$ and a time $t \in T$ by a new entity called a shortlist s. Our problem becomes to rank all the items $i \in I$ for a given shortlist s. We kept we original problem for the baselines without this shortlist entity.

This problem slightly differs from previous works in several aspects. First, it is widespread to focus on predicting the next interaction (sequential recommendation), ignoring the current time. However, the formulation of our problem makes the training phase easier, as we will see later. Besides, in practice, we are concerned about recommending when the recommendation is required. In our case, a job has a limited lifespan. Second, many works on temporal recommendation on graphs only focus on classifying a random negative sample and a true example. As we will notice later, this evaluation's results are unsuitable for recommender systems.

IV. Methodology

Figure 1 gives an overview of TIMBRE.

A. Graph Construction

a) Basic Structure: As input to our algorithm consists of candidate resumes, job postings, and information prefilled by the candidate or the recruiter. After discussing with professional recruiters, we selected features and represented them as a heterogeneous temporal graph (similar to [17]). More specifically, we have **eleven kinds of nodes**: candidates, jobs, companies, salaries, number of years of experience, skills, skill concepts (high-level skills), types of contract (permanent, temporary, freelance), location (through a zip code), job category, and candidate origins (recruiting platform, like Linkedin). All these fields are completed manually as part of the recruitment process of a company (either by the candidate when they apply for a position or by a recruiter when they enter a new position or candidate in the database), except for the skills, which are augmented automatically by searching for keywords in the resumes and job descriptions. These keywords come from two **external knowledge bases**: the European classification of Skills, Competencies, Qualifications, and Occupations [60] and Wikidata [61]. They also enrich the information about the skills by introducing a hierarchy of skills. The nodes of our graph are connected with named relations like *hasSkill* or *atLocation*. For nodes with temporal information (date of creation for the candidates and the jobs), we encode it as a timestamp in a property of the node. We set the timestamp to 0 for others to make temporal sampling easier. We also have a resume and a job posting as text, so we turned them

into embeddings using a sentence transformer model [62] and attached them to the nodes. In the end, our graph contains **44 different edge types** (half are, in fact, reverse edge types).

b) Shortlist Nodes: Most works represent a recommendation between a user and an item by an edge. However, it might cause problems for a temporal recommendation. First, **encoding the interaction time directly on an edge is hard**, and we cannot use the candidate and job nodes as they already have timestamps. Second, **generating a negative sample becomes harder**. What is the timestamp of the negative interaction, and where do we encode it, as the fake interaction is not part of the graph? To solve these problems, we created a new kind of node inspired by reification principles used in RDF (Resource Description Framework). For each interaction, (u, i, t), we create a new node $S_{u,i,t}$ called a **shortlist node** that is connected to u and i, has t as a timestamp and is linked to the corresponding temporal node. We do not have a direct connection between u and i. Now, if we want to create a negative sample for an interaction (u, i, t), we pick a random item i' and connect it to $S_{u,i,t}$. **The time of the negative sample is automatically managed**. Our final graph has **no edge between a candidate and a job**. The interaction goes through a shortlist node.

c) Temporal Nodes: In most works, the temporal information is encoded by the nodes' embeddings depending on time. This article chose a simpler yet effective approach. The timestamp property of the nodes will be used for sampling (see later). Besides, time is crucial when recommending a job, as the job might be too old. Therefore, we introduced a **new type of node representing the number of months since the first shortlist**. This node has a feature composed of a single number, the number of months, and a timestamp corresponding to the time at the beginning of the month. **Temporal nodes** are connected to candidates, jobs, and shortlist nodes.

B. Job Prediction

We train our network using the **link prediction** task. The goal of this task is to predict whether there is a link or not between two nodes in the graph. In our case, we want to predict whether there is a **link between a shortlist node and a job node**, equivalent to recommending a job for a candidate at a given timestamp. To recommend a candidate for a job, we can predict a link between a shortlist node and a candidate node. For the training part, we need to have positive samples (coming from the dataset) and negative samples (generated randomly as explained in Section IV-A).

Because of the size of the graph, using the entire graph to make the recommendation is too expensive. Therefore, following previous works [15], we decided to **sample a sub-graph** around the shortlist node and the job node we consider. During this sampling, it is crucial to ignore the nodes that do not exist at the moment of the recommendation, i.e., we can only keep the candidates, job postings, and shortlisting events anterior to the current shortlist node we consider. This kind of filtering is not necessarily done in the literature, and it causes problems in the case of data annotated by recruiters as they tend to create groups of candidates and submit them to the same jobs [17]. Therefore, the component of our sampling strategy includes a **temporal filtering of future events**.

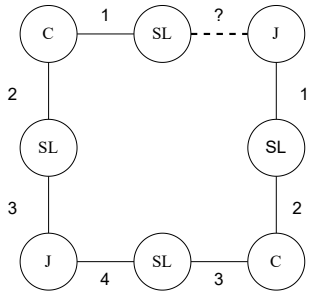

Fig. 2. Sampling Distance For CF. C=Candidate, J=Job, SL=Shortlist.

In most cases, sampling a sub-graph containing the nodes at a distance two or less from the original nodes gives good results. However, in our case, given the diversity of edge types and the presence of the shortlist node, we need to pay close attention. Particularly, if we want to have a form of collaborative filtering (CF), we need to sample candidates who applied for similar jobs. In our case, we need to have a sampling depth of four (see Figure 2), but we need to be careful not to sample certain edge types. Indeed, if we follow the edges going from a skill to a candidate, we will sample too many nodes (all the candidates with that skill). Therefore, we used a **selective sampling** by only sampling edges going from a candidate, shortlist, or job node to another node (23 edge types over 44 in total. E.g., the edges (shortlist, has_application, job), (candidate, has_skill, skill), or (job, has_experience, experience)). Besides, we decided to **take all the edges at a certain depth** and not a sample. Although it makes the computation longer, it introduces less noise in the results and makes the sampling deterministic.

C. Graph Architecture

This paper uses a graph neural network (GNN), more precisely a **graph convolutional network** (GCN), to make the predictions. Each node in our graph is associated with an embedding that does not depend on time. For nodes with features (candidates, jobs, time nodes), the final embedding is a linear combination of a learned embedding and the feature vector. Then, we have several layers of SAGE convolutions [15], where each of them is normalized using a layer normalization [63]. The non-linearity function is a GELU (Gaussian Error Linear Unit) [64]. After the convolutions, we get a vector for the shortlist node and the job we consider, and we compare them using cosine similarity. Finally, we apply the binary cross entropy loss. As we have a heterogeneous graph, we used the transformation from [65] to adapt our network.

D. Evaluation

Most graph-based approaches from the literature [14], [16], [50], [66] only report metrics like **precision, recall, and area under the curve (AUC)** on the task of link prediction. This biased evaluation gives minimal insight into the model's performance. Indeed, these metrics **only evaluate the capability**

to separate negative samples from positive samples. The negative samples are often drawn randomly, and these random items are effortless to differentiate from positive examples. Therefore, the metrics reported in previous works are very high but without much interest.

Instead, **we evaluate the capability of the models to rank the items for a given user**. With a matrix-based approach with fixed precomputed embeddings of each user and item, the ranking is easy and fast to compute: We perform a matrix multiplication and sort the results. However, this is impossible for graph-based approaches trained for link prediction. Instead, we must **run the link prediction task for each user-item pair** many times and sort the results. This process can be very long compared to the training, explaining partially why the literature abandoned the ranking evaluation. Another reason is that, **for temporal graphs**, when predicting a user-item interaction, we need to decide when this interaction happens, and we go back to the problem raised in Section IV-A. We need to use the timestamp of real interactions, but it is unclear what to pick for false interactions. **Using our shortlist node solves the problem**. We focus on predicting a user-item interaction at a given time that is directly encoded in the node. Therefore, we do not have to care about the time, and we do as if the task was to rank all the jobs for a shortlist node.

V. Experiment Setup

a) Dataset: We used the JTH (Job Tracking History) dataset introduced in [17]. This dataset comprises 67k real candidate-job associations manually annotated by professional recruiters. We can access 67k candidates (only 26k have at least an associated job) and 4k jobs (most have a recommendation). Candidates have a resume, and jobs have a description. Besides, recruiters might add additional information like relevant skills or wanted salaries. We divided our dataset into train, validation, and test sets **using a temporal order** to prevent data leakage, with a proportion of 80/10/10. Due to the nature of the data, the testing dataset mostly contains **unseen users**, making the cold start problem central.

b) Baselines: We compared our approach with the following state-of-the-art approaches: Temporal Graph Network [14] (TGN), Neighbour Aware Temporal Network [50] (NAT), JODIE [66], DYREP [13], TGSRec [67]. Besides, we have two approaches based on large language models (LLM) that compute an embedding for each user and item and then rank the item using the cosine similarity in a deterministic way (therefore, no standard deviation). We used text-embedding-3 from OpenAI [68] and BGE-M3 [69].

c) Evaluation Setup: We focused on predicting a job for a given candidate, but the opposite would work the same. Note that for our approach, we actually want to predict a job for a shortlist node, i.e., a job for a candidate at a given time, making our problem harder than the one for the baselines. For simplicity, in what follows, we call "user" a normal user for the baselines but a shortlist node in our approach. During the training, all the baselines have access to the training and validation sets. Then, for the final evaluation of the test set, we proceed as follows. For each interaction between a user u and an item i, we first start by drawing a random negative sample i_{neg}, and we compute the scores for both i and i_{neg}. That allows us to get the **classification metrics** (see below). Then, for each item i' (not necessary in the test set), we compute the score between u and i' to **produce a ranking of all the items** for the user u. This ranking is used to compute the recommendation metrics (see below). Note that for each interaction, the model can have access to all previous interactions but not to future interactions.

d) Metrics: We will report two kinds of metrics: Metrics related to the classification task with negative samples (as used in [14], [16], [50], [66]) and metrics traditionally used for recommendation. We use the area-under-the-curve (**AUC**) and the **precision** for the classification metrics. Then, we use the mean reciprocal rank (**MRR**) and **Recall@10** for the recommendation metrics. For most experiments, we ran them over 10 different seeds and reported the mean score and the standard deviation (SD). In details, we have:

$$MRR = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{1}{rank_i} \tag{1}$$

$$Recall@K = \frac{1}{|D|}\sum_{i=1}^{|D|}\mathbb{1}(rank_i \leq K) \tag{2}$$

D is the test dataset, and $rank_i$ is the rank of the first positive answer. Note that these formulas work only in our case, as we have exactly one positive example by shortlist node (by construction). The formulas were adapted for the baselines, which do not use the shortlist node.

e) Configuration: We wrote our code in Python, using Torch and Pytorch-Geometric [70]. We ran our experiments on an NVIDIA Tesla V100 GPU. A training and evaluation cycle took between one day and two days to run (most of it is the evaluation). The optimizer is Adam [71] with a learning rate of $1e{-5}$ and a weight decay of $1e{-4}$. Our GCN has three layers. Three is a tradeoff between computation time and performance, as adding more layers makes the experiments longer. We make our code available on https://github.com/Aunsiels/job_recommendation. For the baselines, we reused the code provided by the original authors and adapted our data to fit their input format. Besides, we created a feature vector for each candidate and job to help the baselines.

VI. Results

a) Main Results: Table I displays the comparison of our approach (TIMBRE) with the different baselines. We observe significant variations when looking at the metrics reported originally with the baselines (AUC and precision). Two methods seem to have an edge: Jodie and TIMBRE. However, **we do not observe the same behavior when looking at the recommendation metrics** (MRR and Recall@10). TIMBRE significantly outperforms all the other baselines, with a factor of 5 for the MRR and Recall@10. Besides, looking solely at the baselines that compute the AUC and the precision,

TGSRec got the best score, which was not true for the AUC and precision. Again, it shows that these non-recommendation metrics are helpful during training but do not fully indicate the system's final performance. Finally, we must note that the scores underestimate the model's performance in production. In practice, we can filter many results using simple filters, like checking if a position is still open. This also makes the generation of the recommendation faster. However, we wanted to test the capability to understand and model business rules and temporal events. Besides, we did not necessarily have access to all the data to write these filters.

The two main reasons for our approach's success are its capability to **integrate external information in a unified system** and its possibility to **leverage recent interactions without retraining** (not feasible in practice due to computation time), even in the test case. Therefore, we have much more success in tackling the cold start problem, which is generalized in the case of job recommendation, as discussed earlier.

b) Ablation Study: To understand which components were helpful or not, we performed an ablation study that we reported in Table II. The first thing to realize is that a significant component of our system is the **temporal nodes**. Although very simple (compared to the complex modeling in the baselines), they allow the model to discard jobs that are too old. Next, we observe that only one feature seems useless: The number of years of experience. Looking more closely at the data, we notice that most job postings (96%) do not have this information filled. Therefore, this feature creates noise. Removing the other feature harmed the results, which was expected. The categories (a manual classification of the domain of expertise made by the recruiter) seem to be the most essential feature. The reason is that it is pretty clean and has limited possible values. The zip code is also crucial, as we mostly want to recruit people near the job posting. The candidate's origin plays a significant role, which indicates that some sources of candidates are more reliable than others. As we could have guessed, the type of contract is also essential to know. However, knowing the recruiting company is not that helpful. Surprisingly, the salary feature is not that central. The reason is similar to the years of experience: Very few candidates choose to provide that information that information. Finally, removing the skills or the concepts seems to have a similar impact. As the concepts represent the hierarchy of skills, we also remove them by removing the skills. So, high-level skills appear more critical when assigning a job than fine-grain skills.

We continued the ablation study by analyzing several scenarios. First, we removed all the features (-all, including time nodes). We observed that TIMBRE still outperforms several baselines, which shows that it can leverage interactions as well as previous approaches. To understand what we are doing better, we tried to remove the features of the jobs and candidates (-features) and our collaborative filtering sampling (-collab.). For this last point, we sampled all the nodes at a depth of two. From the results, we can conclude that removing the features has a slightly negative impact, but **changing the sampling was the critical point**. It shows that our analysis was correct: We must pick the sampling correctly to ensure we allow collaborative filtering.

c) Error Analysis: To better understand the results of our experiments, we performed an error analysis. We took a random sample of 100 users in the test set and associated the most probable job position according to our model. Then, we asked a human annotator in a recruiting company to access the recommendations with the candidate profile and job description and manually label them as correct or incorrect. If the recommendation is wrong, the annotator must also give a reason. We also asked the annotator only to use the information in the resume and job description, making the evaluation stricter than what we would expect in real life (we do not account for job progression, for example). Our results are presented in Table III. As we can see, the top recommendation is often wrong. The most frequent cause is a mismatch of skills, which is consistent with Table II where we saw that removing skills was not that harmful. Interestingly, the job title is often correct (e.g., Frontend developer), but the technologies required do not match. A potential cause of why the skills are not correctly used is how they are extracted. Recruiters rarely fill them, but they are extracted automatically from resumes, thus creating a lot of noise. Besides, the annotator is not necessarily aware of similar libraries or skills that can quickly be acquired in a new position, making the annotation hard. Finally, the graph might not contain enough information to understand a skill. A solution could be to develop a more fine-grained ontology for skills. Another reason for error is a time inconsistency between a candidate and a position (the candidate was created too much in the past or future). A possible way to solve this problem would be to hardcode a time filtering or to adapt the negative sampling to learn the time constraints better. Next, we observed issues related to a mismatch in experience: A junior position is often assigned to a senior or team leader person (the opposite is rarely true). As we already mentioned, the experience of a candidate and a job is seldom filled by the recruiter and, therefore, hard to exploit, although it can be found or guessed from the resume. Finally, we see very few errors due to a wrong location. This field is complex to access as a candidate might be willing to move to a new place. It is also an information that is time-dependent as candidate addresses can become obsolete. To conclude, all the problems reported here were also present in the baselines, which shows that these recommender systems cannot be used directly out of the box. In a practical case, our system could be included in a broader system with a possible post-filtering to refine the results.

VII. Conclusion

This paper introduces TIMBRE, a temporal job recommender system based on graphs. TIMBRE first integrates all the available information into a temporal heterogeneous graph. Then, it uses three components to facilitate the temporal recommendation and improve performance: The inclusion of a reification node (the shortlist node) that represents

Metrics	NAT	TGN	Jodie	DYREP	TGSRec	OpenAI	BGE-M3	TIMBRE
AUC	0.7893 (SD 0.0091)	0.4283 (SD 0.0607)	0.9356 (SD 0.0041)	0.7489 (SD 0.0052)	0.7175 (SD 0.1007)	-	-	**0.9479** (SD 0.0089)
Precision	0.7938 (SD 0.0097)	0.4579 (SD 0.0116)	**0.9142** (SD 0.0008)	0.7375 (SD 0.0046)	0.8605 (SD 0.0457)	-	-	0.8640 (SD 0.0112)
MRR	0.0048 (SD 0.0018)	0.0004 (SD 0.0001)	0.0049 (SD 0.0026)	0.0022 (SD 0.0026)	0.0106 (SD 0.0051)	0.0117 (SD 0)	0.0173 (SD 0)	**0.0909** (SD 0.0270)
Recall@10	0.0037 (SD 0.0012)	0.0005 (SD 0.0000)	0.0029 (SD 0.0014)	0.0018 (SD 0.0011)	0.0200 (SD 0.0115)	0.0200 (SD 0)	0.0353 (SD 0)	**0.1965** (SD 0.0501)

TABLE I
COMPARISON WITH THE BASELINES. SD = STANDARD DEVIATION.

Setup	AUC	Precision	MRR	Recall@10
All	0.9509	0.8717	0.0763	0.1651
-experience	0.9479	0.8640	0.0909	0.1965
-company	0.9531	0.8665	0.0652	0.1476
-salary	0.9486	0.8794	0.0593	0.1258
-skill	0.9470	0.8723	0.0577	0.1245
-concept	0.9515	0.8551	0.0575	0.1305
-contract	0.9486	0.8731	0.0547	0.1302
-origin	0.9424	0.8650	0.0529	0.1133
-zip	0.9441	0.8697	0.0438	0.0945
-categories	0.9439	0.8731	0.0413	0.0868
-temporal nodes	0.6533	0.7005	0.0058	0.0058
-all	0.6920	0.6678	0.0095	0.0153
-all -features	0.7236	0.6651	0.0070	0.0118
-all -collab.	0.7050	0.6448	0.0066	0.0094
-features	0.9558	0.8668	0.0692	0.1549
-collab.	0.8877	0.7606	0.0397	0.0745

TABLE II
ABLATION STUDY

	Percentage
Correct	21 %
Incorrect	79 %
Incorrect - Mismatch skills	60 %
Incorrect - Incorrect temporality	14 %
Incorrect - Lack of experience	11 %
Incorrect - Wrong Location	4 %
Incorrect - Overqualified	1 %

TABLE III
ERROR ANALYSIS

an interaction at a given time, the addition of a temporal node that encodes a point in time, and smart sampling that enables collaborative filtering. Our experiments showed that our methodology outperforms state-of-the-art temporal graph recommendation methods, particularly using recommendation metrics that were rarely used before with graphs.

a) Limitations and Future Works: In this paper, we discuss job recommendations, which are often discrimination-prone. We did our best to remove any feature related to gender or ethnicity. Our solution can also have a societal impact as it automatizes part of the recruitment process. However, we want to stress that it should be used to assist recruiters in finding the best position for a given person, as we do not provide any strong guarantee of the results.

Due to the nature of the data we manipulated, we used a private dataset in our experiment. As there is no equivalent public dataset, applying our approach to another real-life dataset on job recommendation is hard, limiting our evaluation's scope. However, our study gives valuable insights into how recruiting works and how it can be improved and assisted. In particular, although our data is of relatively high quality due to the annotation by professional recruiters, it is still subject to noise from the annotation process, mainly introduced by junior recruiters. In future work, we would like to introduce the full result of the recruiting process (until the contract is signed) to analyze the candidates better and provide ways to improve the recruiting process by giving feedback.

As our ablation study shows, including features in the graph is not necessarily trivial and might require further consideration. For example, although the company is irrelevant, its sector may be interesting. That would allow the inclusion of previous positions occupied by a candidate in the graph. For the sparse features, it might be worth finding a way to fill them. However, it would require significant human labor. A possible future work would be to have a human-in-the-loop system in which we suggest candidates and ask for further information. Concerning the graph sampling strategy, we observed that sampling all the nodes at a given depth (four in our case) and following only certain edge types gave the best results while still keeping reasonable computation times. If execution time is critical or computation resources are limited, we could use more advanced sampling strategies like PASS [72] that sample the most important nodes for our task. It will raise the question of adapting such a sampling for temporal heterogeneous graphs. Finally, we assumed that a candidate's preference does not change, impacting how we model time. Although it is true in most cases, future work could try to include the previous experiences of a candidate to model their career path better.

REFERENCES

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

[2] C. Qin, H. Zhu, T. Xu, C. Zhu, C. Ma, E. Chen, and H. Xiong, "An enhanced neural network approach to person-job fit in talent recruitment," *TOIS*, 2020.

[3] P. K. Roy, S. S. Chowdhary, and R. Bhatia, "A machine learning approach for automation of resume recommendation system," *Procedia Computer Science*, 2020.

[4] M. N. Freire and L. N. de Castro, "e-recruitment recommender systems: a systematic review," *Knowledge and Information Systems*, 2021.

[5] A. Giabelli, L. Malandri, F. Mercorio, M. Mezzanzanica, and A. Seveso, "Skills2job: A recommender system that encodes job offer embeddings on graph databases," *Applied Soft Computing*, 2021.

[6] C. Yang, Y. Hou, Y. Song, T. Zhang, J.-R. Wen, and W. X. Zhao, "Modeling two-way selection preference for person-job fit," in *RecSys*, 2022.

[7] L. Wu, Z. Qiu, Z. Zheng, H. Zhu, and E. Chen, "Exploring large language model for graph data understanding in online job recommendations," 2023.

[8] M. Dong, F. Yuan, L. Yao, X. Xu, and L. Zhu, "Mamo: Memory-augmented meta-optimization for cold-start recommendation," 2020.

[9] Y. Lu, Y. Fang, and C. Shi, "Meta-learning on heterogeneous information networks for cold-start recommendation," in *SIGKDD*, 2020.

[10] Y. Zhu, R. Xie, F. Zhuang, K. Ge, Y. Sun, X. Zhang, L. Lin, and J. Cao, "Learning to warm up cold item embeddings for cold-start recommendation with meta scaling and shifting networks," in *SIGIR*, 2021.

[11] Y. Wei, X. Wang, Q. Li, L. Nie, Y. Li, X. Li, and T.-S. Chua, "Contrastive learning for cold-start recommendation," 2021.

[12] D. Cai, S. Qian, Q. Fang, J. Hu, and C. Xu, "User cold-start recommendation via inductive heterogeneous graph neural network," *TOIS*, 2023.

[13] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," 2019.

[14] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," 2020.

[15] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," 2020.

[16] ——, "A temporal kernel approach for deep learning with continuous-time information," 2021.

[17] E. Behar, J. Romero, A. Bouzeghoub, and K. Wegrzyn-Wolska, "Tackling cold start for Job recommendation with heterogeneous graphs," *CEUR Workshop Proceedings*, 2023.

[18] S. Yang, M. Korayem, K. AlJadda, T. Grainger, and S. Natarajan, "Combining content-based and collaborative filtering for job recommendation system: A cost-sensitive statistical relational learning approach," *Knowledge-Based Systems*, 2017.

[19] R. Mishra and S. Rathi, "Efficient and scalable job recommender system using collaborative filtering," in *ICDSMLA*, 2020.

[20] J. Dhameliya and N. Desai, "Job recommendation system using content and collaborative filtering based techniques," *Int J Soft Comput Eng*, 2019.

[21] D. Prince, K. Madhan, K. Vishwa, and D. Yamunathangam, "Job and course recommendation system using collaborative filtering and naive bayes algorithms," in *ICAECA*, 2023.

[22] F. Abel, A. Benczúr, D. Kohlsdorf, M. Larson, and R. Pálovics, "Recsys challenge 2016: Job recommendations," in *RecSys*, 2016.

[23] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *RecSys*, 2016.

[24] O. Barkan and N. Koenigstein, "Item2vec: neural item embedding for collaborative filtering," in *MLSP*, 2016.

[25] C. Daryani, G. S. Chhabra, H. Patel, I. K. Chhabra, and R. Patel, "An automated resume screening system using natural language processing and similarity," *ETHICS AND INFORMATION TECHNOLOGY*, 2020.

[26] A. K. Sinha, M. Amir Khusru Akhtar, and A. Kumar, "Resume screening using natural language processing and machine learning: A systematic review," *ICMLIP*, 2021.

[27] S. Bharadwaj, R. Varun, P. S. Aditya, M. Nikhil, and G. C. Babu, "Resume screening using nlp and lstm," in *ICICT*, 2022.

[28] H. Sajid, J. Kanwal, S. U. R. Bhatti, S. A. Qureshi, A. Basharat, S. Hussain, and K. U. Khan, "Resume parsing framework for e-recruitment," in *IMCOM*, 2022.

[29] S. Mohanty, A. Behera, S. Mishra, A. Alkhayyat, D. Gupta, and V. Sharma, "Resumate: A prototype to enhance recruitment process with nlp based resume parsing," in *ICIEM*, 2023.

[30] V. S. Tallapragada, V. S. Raj, U. Deepak, P. D. Sai, and T. Mallikarjuna, "Improved resume parsing based on contextual meaning extraction using bert," in *ICICCS*, 2023.

[31] Y. Li, Y. Zhang, and L. Sun, "Metaagents: Simulating interactions of human behaviors for llm-based task-oriented coordination via collaborative generative agents," 2023.

[32] P. Ghosh and V. Sadaphal, "Jobrecogpt–explainable job recommendations using llms," *arXiv preprint arXiv:2309.11805*, 2023.

[33] L. Wu, Z. Qiu, Z. Zheng, H. Zhu, and E. Chen, "Exploring large language model for graph data understanding in online job recommendations," in *AAAI*, 2024.

[34] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, 2022.

[35] Q. Guo, F. Zhuang, C. Qin, H. Zhu, X. Xie, H. Xiong, and Q. He, "A survey on knowledge graph-based recommender systems," *TKDE*, 2022.

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

[37] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: A survey," *ACM Comput. Surv.*, 2022.

[38] W. Shalaby, B. AlAila, M. Korayem, L. Pournajaf, K. AlJadda, S. Quinn, and W. Zadrozny, "Help me find a job: A graph-based approach for job recommendation at scale," in *BigData*, 2017.

[39] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, "Kgat: Knowledge graph attention network for recommendation," in *SIGKDD*, 2019.

[40] Z. Yang and S. Dong, "Hagerec: Hierarchical attention graph convolutional network incorporating knowledge graph for explainable recommendation," *Knowledge-Based Systems*, 2020.

[41] E. Palumbo, G. Rizzo, and R. Troncy, "Entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation," in *RecSys*, 2017.

[42] E. Rich, "Users are individuals: individualizing user models," *International journal of man-machine studies*, 1983.

[43] V. Bogina, T. Kuflik, D. Jannach, M. Bielikova, M. Kompan, and C. Trattner, "Considering temporal aspects in recommender systems: a survey," *User Modeling and User-Adapted Interaction*, 2023.

[44] D. Kowald, S. Kopeinik, P. Seitlinger, T. Ley, D. Albert, and C. Trattner, "Refining frequency-based tag reuse predictions by means of time and semantic context," in *Workshop at MUSE*, 2015.

[45] Y. Song, A. M. Elkahky, and X. He, "Multi-rate deep learning for temporal recommendation," in *SIGIR*, 2016.

[46] X. Wang, Y. Wang, D. Hsu, and Y. Wang, "Exploration in interactive personalized music recommendation: a reinforcement learning approach," *TOMM*, 2014.

[47] R. W. White, A. Kapoor, and S. T. Dumais, "Modeling long-term search engine usage," in *UMAP*, 2010.

[48] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan, "A dynamic recurrent model for next basket recommendation," in *SIGIR*, 2016.

[49] R. He, C. Fang, Z. Wang, and J. McAuley, "Vista: A visually, socially, and temporally-aware model for artistic recommendation," in *RecSys*, 2016.

[50] Y. Luo and P. Li, "Neighborhood-aware scalable temporal network representation learning," 2022.

[51] L. Wu, S. Li, C.-J. Hsieh, and J. Sharpnack, "Sse-pt: Sequential recommendation via personalized transformer," in *RecSys*, 2020.

[52] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *CIKM*, 2019.

[53] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *ICDM*, 2018.

[54] Y. Yang, A. Kneip, and C. Frenkel, "Evgnn: An event-driven graph neural network accelerator for edge vision," *arXiv:2404.19489*, 2024.

[55] S. Schaefer, D. Gehrig, and D. Scaramuzza, "Aegnn: Asynchronous event-based graph neural networks," in *CVPR*, 2022.

[56] Y. Bi, A. Chadha, A. Abbas, E. Bourtsoulatze, and Y. Andreopoulos, "Graph-based spatio-temporal feature learning for neuromorphic vision sensing," *IEEE Transactions on Image Processing*, 2020.

[57] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *TIIS*, 2015.

[58] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *SIGKDD*, 2011.

[59] N. Asghar, "Yelp dataset challenge: Review rating prediction," *arXiv:1605.05362*, 2016.

[60] J. D. Smedt, M. le Vrang, and A. Papantoniou, "Esco: Towards a semantic web for the european labor market," in *LDOW@WWW*, 2015.

[61] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Commun. ACM*, 2014.

[62] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *NeurIPS*, 2020.

[63] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv:1607.06450*, 2016.

[64] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv:1606.08415*, 2016.

[65] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *ESWC*, 2018.

[66] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *SIGKDD*, 2019.

[67] Z. Fan, Z. Liu, J. Zhang, Y. Xiong, L. Zheng, and P. S. Yu, "Continuous-time sequential recommendation with temporal graph collaborative transformer," in *CIKM*. ACM, 2021.

[68] OpenAI, "New embedding models and api updates," https://openai.com/index/new-embedding-models-and-api-updates/, 2024.

[69] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," 2024.

[70] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv:1903.02428*, 2019.

[71] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[72] M. Yoon, T. Gervet, B. Shi, S. Niu, Q. He, and J. Yang, "Performance-adaptive sampling strategy towards fast and accurate graph neural networks," in *SIGKDD*, 2021.