Configuration of Industrial Automation Solutions Using Multi-relational Recommender Systems

Marcel Hildebrandt ✉[1,2⋆], Swathi Shyam Sunder[1,3⋆], Serghei Mogoreanu[1], Ingo Thon[1], Volker Tresp[1,2], and Thomas Runkler[1,3]

[1] Siemens AG, Corporate Technology, Munich, Germany
{marcel.hildebrandt, swathi.sunder, serghei.mogoreanu, ingo.thon, volker.tresp, thomas.runkler}@siemens.com
[2] Ludwig Maximilian University, Munich, Germany
[3] Technical University of Munich, Munich, Germany

Abstract. Building complex automation solutions, common to process industries and building automation, requires the selection of components early on in the engineering process. Typically, recommender systems guide the user in the selection of appropriate components and, in doing so, take into account various levels of context information. Many popular shopping basket recommender systems are based on collaborative filtering. While generating personalized recommendations, these methods rely solely on observed user behavior and are usually context free. Moreover, their limited expressiveness makes them less valuable when used for setting up complex engineering solutions. Product configurators based on deterministic, handcrafted rules may better tackle these use cases. However, besides being rather static and inflexible, such systems are laborious to develop and require domain expertise. In this work, we study various approaches to generate recommendations when building complex engineering solutions. Our aim is to exploit statistical patterns in the data that contain a lot of predictive power and are considerably more flexible than strict, deterministic rules. To achieve this, we propose a generic recommendation method for complex, industrial solutions that incorporates both past user behavior and semantic information in a joint knowledge base. This results in a graph-structured, multi-relational data description – commonly referred to as a knowledge graph. In this setting, predicting user preference towards an item corresponds to predicting an edge in this graph. Despite its simplicity concerning data preparation and maintenance, our recommender system proves to be powerful, as shown in extensive experiments with real-world data where our model outperforms several state-of-the-art methods. Furthermore, once our model is trained, recommending new items can be performed efficiently. This ensures that our method can operate in real time when assisting users in configuring new solutions.

Keywords: Recommender system · Cold start · Knowledge graph · Link prediction · Tensor factorization

⋆ These authors contributed equally to this work

1 Introduction

Industrial automation systems consist of a wide variety of components – in general, a combination of mechanical, hydraulic, and electric devices – described in a plan. For instance, a plan for a control cabinet might specify the model of the controller, the number of digital input ports, the number of sensors to be connected, and so on. Based on the plan, the user incrementally selects products which in combination fulfill all required functionalities. The goal of our solution is to support the user in this process by reordering product lists, such that products which most likely correspond to the user needs and preferences are on top.

A good ordering of the products may depend on the properties of the components, such as the line voltage common in the country (110V in US, 230V in Europe), marine certification, or whether or not the customer has a preference towards premium or budget products. While some of this information is explicitly modeled as product features – exploited by means of a knowledge graph in our work – a big part of it is implicit information. In our system, the current partial solution, which consists of the already selected components, acts as a source of information about the actual requirements of the user.

The main advantage of our proposed approach, for both the end customer and the sales team, is the reduction of time required to select the right product. Moreover, the system guides the customer towards a typical, and therefore most likely useful, combination of products. In addition, the learned information allows vendors to optimize the portfolio as the learned knowledge about item combinations makes implicit interdependencies transparent.

The problem of finding the right products to fulfill the costumer needs can be addressed from two directions. The first direction is towards general recommender systems that typically find products based on customer preferences. However, in our domain, the customer preferences are not fixed, but rather depend on implicit requirements of the solution currently being built. The second direction focuses on configuration systems which find a combination of products that fulfill all requirements specified in the plan. The practical use of the latter, however, is subject to the precise specification of the requirements. This is rarely the case, especially in industry-based applications such as the one we are dealing with. The use case addressed in this paper concerns an internal R&D project at Siemens. One of the sources of data for this project is the Siemens product database,[1] which presents a diversity-rich environment – offering services related to 135,000 products and systems with more than 30 million variants. Such massive amounts of data make it increasingly difficult to specify all the requirements explicitly and directly add to the computational overhead of the recommender system. Nevertheless, it is crucial to deliver instantaneous results without compromising quality in industrial use cases like ours.

Since our method is intended for real-world usage, it needs to satisfy the following constraints:

[1] https://mall.industry.siemens.com

– Reasonable effort to set up the system in terms of data preparation, maintenance, and integration of new data. The last point is particularly important when new products are being launched.
– Computing recommendations efficiently is crucial because the system must work in real time when the user is configuring new solutions.
– The method must be able to make recommendations for items without or with only little historical customer data.

In this paper, we present RESCOM. The key idea is to employ a modified version of RESCAL [16] for the recommendation task. We not only show that RESCOM outperforms all baseline methods but also demonstrate that we achieved all of the goals stated above.

This paper is organized into six sections. Section 2 introduces the notation and reviews some of the basic principles of knowledge graphs. In Section 3, we proceed by investigating methods that are traditionally used for recommendation tasks and outline the challenges arising from the real-world nature of our data. Furthermore, we review methods that are commonly employed for the link prediction task on knowledge graphs. In Section 4, we present RESCOM, a generic recommendation method for complex, industrial solutions, that incorporates both past user behavior and semantic information in a joint knowledge base. The results of our extensive real-world experiments are presented in Section 5. In particular, we show that RESCOM outperforms several state-of-the-art methods in standard measures. Finally, Section 6 concludes our work by discussing the results and proposing some directions for future research.

2 Notation and Background

Before proceeding, we first define the mathematical notation that we use throughout this work and provide the necessary background on knowledge graphs.

Scalars are given by lower case letters ($x \in \mathbb{R}$), column vectors by bold lower case letters ($\mathbf{x} \in \mathbb{R}^n$), and matrices by upper case letters ($X \in \mathbb{R}^{n_1 \times n_2}$). Then $X_{i,:} \in \mathbb{R}^{n_2}$ and $X_{:,j} \in \mathbb{R}^{n_1}$ denote the i-th row and j-th column of X, respectively. Third-order tensors are given by bold upper case letters ($\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$). Further, slices of a tensor (i.e., two-dimensional sections obtained by fixing one index) are denoted by $\mathbf{X}_{i,:,:} \in \mathbb{R}^{n_2 \times n_3}$, $\mathbf{X}_{:,j,:} \in \mathbb{R}^{n_1 \times n_3}$, and $\mathbf{X}_{:,:,k} \in \mathbb{R}^{n_1 \times n_2}$, respectively.

In Section 4, we propose a recommender system that is based on constructing a knowledge base that contains not only historical data about configured solutions, but also descriptive features of the items available in the marketplace. This leads to a graph-structured, multi-relational data description. Loosely speaking, such kind of data is commonly referred to as a knowledge graph. The basic idea is to represent all entities under consideration as vertices in a graph and link those vertices that interact with each other via typed, directed edges.

More formally, let $\mathcal{E} = \{e_1, e_2, \ldots, e_{n_E}\}$ and $\mathcal{R} = \{r_1, r_2, \ldots, r_{n_R}\}$ denote the set of entities and the set of relations under consideration, respectively. In our setting, entities correspond to either solutions, items, or technical attributes of items. Relations specify the interconnectedness of entities. Pertaining to our use case, the *contains* relation is of particular interest. It links solutions and items by indicating which items were configured in each solution. The remaining relations connect items to their technical attributes. In this work, a knowledge graph is defined as a collection of triples $S \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ which are interpreted as known facts. Each member of S is of the form (e_i, r_k, e_j), where e_i is commonly referred to as subject, r_k as predicate, and e_j as object. Figure 1 depicts an example of a knowledge graph in the context of an industrial purchasing system and the kinds of entities and relations we consider in this paper.

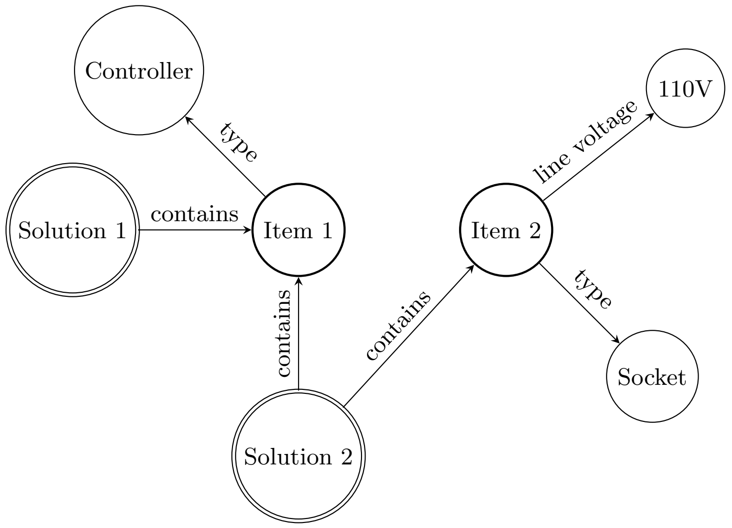


Fig. 1. A knowledge graph in the context of an industrial purchasing system. The nodes correspond either to solutions, items, or technical properties of items. The relationships between the different entities are determined by edges that come in multiple types.

A knowledge graph has a natural representation in terms of an adjacency tensor $\mathbf{X} \in \mathbb{R}^{n_E \times n_E \times n_R}$ with entries

$$\mathbf{X}_{i,j,k} = \begin{cases} 1\,, & \text{if the triple } (e_i, r_k, e_j) \in S\,, \\ 0\,, & \text{otherwise}\,. \end{cases} \tag{1}$$

Usually, a positive entry in the adjacency tensor is interpreted as a known fact. Under the so-called closed-world assumption, the absence of a triple indicates a false relationship. Alternatively, one can make the open world assumption which states that a zero entry is not interpreted as false but rather as unknown. The approach that we present in Section 4 is based on the local closed-world assumption. Thereby, a knowledge graph is assumed to be only locally complete in the sense that $(e_i, r_k, e_j) \notin S$ is assumed to be false if and only if there exists an entity $e_l \in \mathcal{E}$ such that $(e_i, r_k, e_l) \in S$. Otherwise, the triple is interpreted as unknown.

3 Related Methods

Here we cover both well-established general recommender systems and knowledge graph models. Given that knowledge graphs offer a natural representation for semantically structured information (see Figure 1), they are closely related to the domain under consideration.

3.1 General Recommender Systems

Traditional approaches to recommender systems are based on collaborative filtering, where the problem is formulated as that of matrix completion pertaining to the user-item matrix. The winning solution to the Netflix challenge [2] and Amazon's item-to-item collaborative filtering method [12] are two prominent examples. A general class of collaborative filtering methods is based on factorizing the user-item matrix $X \in \mathbb{R}^{n_U \times n_I}$, where n_U and n_I correspond to the number of user and items, respectively. Further, the entries of X are only well-defined on an index set $\mathcal{I}$ that correspond to observed user-item interactions. The underlying optimization problem consists of factorizing X into the product of two low-dimensional matrices, i.e.,

$$\min_{U,V} \sum_{(i,j)\in\mathcal{I}} (X_{i,j} - U_{i,:}V_{j,:}^T)^2 + \lambda(||U||_F^2 + ||V||_F^2)\,, \tag{2}$$

where $||\cdot||_F$ denotes the Frobenius norm. Moreover, the factor matrices $U \in \mathbb{R}^{n_U \times d}$ and $V \in \mathbb{R}^{n_I \times d}$, where d denotes the number of latent factors, represent the strength of the associations between the latent features of users and items. So the product UV^T essentially performs a sum of the features weighted by the corresponding coefficients. Imposing elementwise non-negativity constraints on the factor matrices U and V in Equation (2) leads to the non-negative matrix factorization (NMF). Similar to most factorization-based approaches, NMF is most effective when there is a high availability of user behavior information.

However, sparsity, which stems from the fact that users typically will have rated only few of the items, constitutes a challenge to collaborative filtering methods. Further, collaborative filtering recommender systems are also prone to the cold start issue, i.e., the problem of dealing with new users or novel items, while suffering from a lack of preferences or content information. This is caused by the context information not being taken into account in collaborative filtering approaches.

Although context-aware recommender systems (CARS) adapt recommendations to the specific contextual situations of users, they fail to address the issue with user/product cold start. Additionally, as described in [1], most of the existing work in this area require all of the context information to be known a priori and to be modeled explicitly. This is in contrast to our use case of industry automation, where a majority of the context information is rather implicit.

While content-based filtering does consider context, recommendations are primarily generated based on a comparison between the content of items and the user profile. This causes only those items to be recommended that are similar to the items already rated by the user, resulting in a lack of diversity and novelty. While discussing the shortcomings of content-based systems, [13] also states that this problem of over-specialization limits the range of applications where the method could be useful.

Contrary to pure approaches, hybrid recommender systems are based on combining multiple models in one of the several possible ways described in [6]. While weighted hybrid recommenders (e.g., [7] and [18]) manually assign weights to the collaborative and content-based recommendation components, switching hybrid recommender systems such as [3] choose among the different recommendation components based on some criteria. Further, there also exist feature-based techniques where a set of features is first computed and subsequently fed into the recommendation algorithm. Although hybrid methods can provide more accurate recommendations than any method on its own, they are still required to strike the right balance between collaborative approaches and knowledge-based methods. The strength of our method lies in its ability to automatically learn this balance without heavy reliance on either deterministic rules or user behavior.

Together with the multitude of practical applications trying to help users cope with information overload, the increased development of new approaches to recommender systems – including this work – may also be attributed to tackling the existing challenges discussed in this section.

3.2 Knowledge Graph Methods

Most knowledge graphs that are currently in practical use are far from being complete in the sense that they are missing many true facts about the entities at hand. Therefore, one of the most important machine learning tasks related to knowledge graphs consists of predicting new links (i.e., facts) given the remaining knowledge graph. This problem is sometimes also referred to as knowledge graph completion. In our setting, the recommendation task is equivalent to predicting new links of a certain type based on the other items that are currently configured.

Many link prediction methods belong to the class of latent feature models. These methods approach the knowledge graph completion task by modeling the score of a candidate triple as a function of learned latent representations of the involved entities and relations. The method that we propose in this paper is based on RESCAL [16]. RESCAL constitutes a collective tensor factorization model that scores triples based on a set of bilinear forms. A more detailed description of RESCAL along with our modifications to make it applicable in our setting is presented in the next section.

Another popular class of methods against which we compare our approach is called translational methods. Here, the key idea is to find embeddings for the entities in a low-dimensional vector space and model their relatedness through translations via relation-specific vectors. TransE, introduced by [4], was the first among this class of methods. It aims at learning representations such that $\mathbf{e}_i + \mathbf{r}_k \approx \mathbf{e}_j$ if $(e_i, r_k, e_j) \in S$, where the bold letters refer to the vector space embeddings of the corresponding entities and relations.

After realizing that TransE is quite limited when dealing with multi-cardinality relations, several other methods that extend the idea of TransE and add more expressiveness have been developed: TransH [20], TransR [11], and TransD [9]. All these methods learn linear mappings specific to the different relations and then map the embeddings of entities into relation-specific vector spaces where they again apply TransE.

One of the ways to apply this translation-based approach to recommender systems was introduced in [8] where personalized recommendations are generated by capturing long-term dynamics of users. However, the focus is geared more towards sequential prediction, i.e., predicting the next item for a user, knowing the item previously configured in the solution. While it does not consider possible interdependencies among all different items previously configured in the solution, it also does not deal with as much semantic information of the items, as this work.

4 Our Method

In analogy to existing collaborative filtering approaches which are based on matrix factorizations, RESCOM relies on exploiting the sparsity pattern of $\mathbf{X}$ by finding a low-rank approximation via tensor factorization.

Our method, RESCOM, is based on a modified version of RESCAL [16] which is a three-way-tensor factorization that has shown excellent results in various relational learning tasks (e.g., in [17]). The key idea is to approximate the adjacency tensor $\mathbf{X}$ by a bilinear product of the factor matrix $E \in \mathbb{R}^{d \times n_E}$ and a core tensor $\mathbf{R} \in \mathbb{R}^{d \times d \times n_R}$, where d corresponds to the number of latent dimensions. More concretely, we impose

$$\mathbf{X}_{:,:,r} \approx E^T \mathbf{R}_{:,:,r} E \quad , \forall r = 1, 2, \ldots, n_R \,. \tag{3}$$

Thus, after fitting the model, the columns of E denoted by $(\mathbf{e}_i)_{i=1,2,\ldots,n_E} \subset \mathbb{R}^d$ contain latent representations of the entities in $\mathcal{E}$. Similarly, each frontal slice of $\mathbf{R}$ contains the corresponding latent representations of the different relations in $\mathcal{R}$. These embeddings preserve local proximities of entities in the sense that, if a large proportion of the neighborhood of two entities overlaps, their latent representations are also similar. Hence, we obtain that, if items are similar from a technical point of view or if they are often configured within the same solution, they will have similar latent representations.

In its original form, the parameters of RESCAL are obtained by minimizing the squared distance between the observed and the predicted entries of $\mathbf{X}$. Hence, the objective function is given by

$$\operatorname{loss}(\mathbf{X}, E, \mathbf{R}) = \sum_{r=1}^{n_R} ||\mathbf{X}_{:,:,r} - E^T \mathbf{R}_{:,:,r} E||_F^2 \,. \tag{4}$$

While logistic extensions of RESCAL that model the entries of $\mathbf{X}$ explicitly as Bernoulli random variables were proposed (see [15]), we stick to the formulation given by Equation (4). This is mainly due to the fact that when using the squared error loss, recommending new items for partial solutions can be performed efficiently via an orthogonal projection into the latent feature space (see below for more details). We also ran experiments with DistMult [21], which can be obtained as a special case of RESCAL when the frontal slices of the core tensor $\mathbf{R}$ are restricted to be diagonal. However, since this did not lead to a better performance, we do not report the results in this work.

Usually Equation (4) is minimized via alternating least squares (ALS). However, the sparsity in our data (only about $6 \cdot 10^{-6}\%$ of the entries of $\mathbf{X}$ are nonzero) caused numerical instability when computing the least-squares projections for ALS. In particular, we tested two different implementations of RESCAL[2] that are based on ALS and found that they were both infeasible in our setting. Therefore, we implemented our own version of RESCAL that aims to minimize a modified loss function. In order to obtain an approximate version of Equation (4) that does not require to take all the entries of $\mathbf{X}$ into account, we sample negative examples from the set of unknown triples $S' \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E} \setminus S$. More specifically, we employ the following loss function during training

$$\text{loss}(\mathbf{X}, E, \mathbf{R}) = \sum_{(e_i, r_k, e_j) \in S} (1 - \mathbf{e}_i^T \mathbf{R}_{:,:,k} \mathbf{e}_j)^2 + (\mathbf{e}_i^T \mathbf{R}_{:,:,k} \tilde{\mathbf{e}})^2 \,, \tag{5}$$

where $\tilde{\mathbf{e}}$ corresponds to the latent representation of a randomly sampled entity $\tilde{e}$ such that $(e_i, r_k, \tilde{e}) \in S'$. Further, we impose the additional constraint that $\tilde{e}$ appears as object in a known triple with respect to the k-th relation (i.e., there exists an entity e so that $(e, r_k, \tilde{e}) \in S$). This sampling procedure can be interpreted as an implicit type constraint which teaches the algorithm to discriminate between known triples on one side and unknown but plausible (i.e., semantically correct) triples on the other side. Conceptually, our sampling mechanism is related to the local closed-world assumption proposed in [10], where the goal is to infer the domain and range of relations based on observed triples. While [10] proposes a type-constrained ALS procedure, our training process is based on generating negative samples and is carried out by minimizing Equation (5) using different versions of stochastic gradient descent.

In the context of industrial automation, generating recommendations is most relevant right at the time when new solutions are being configured, thus making it necessary to tune the previously trained model to make predictions on the new data. While being time consuming, re-training the model on the freshly available partial solutions is possible. However, the need for immediate real-time updates makes model re-building an infeasible option for the recommendation task. Hence we propose to achieve this through the *projection* step. This is a special case of the more general embedding mapping for tensor factorization models proposed in [22]. To ease the notation, let $\mathcal{E}_I \subset \mathcal{E}$ with $|\mathcal{E}_I| =: n_{E_I}$ denote the collection of entities that correspond to configurable items.

We start by constructing a binary vector $\mathbf{x} \in \mathbb{R}^{n_{E_I}}$, where a value of 1 at the i-th position indicates that the corresponding item was part of the partial solution.

[2] https://github.com/mnick/rescal.py
https://github.com/nzhiltsov/Ext-RESCAL

For the purpose of recommendations, we only consider the *contains* relation which represents the information linking solutions to the items configured in them. In order to ease the notation, let r_1 denote the *contains* relation.

Then, we perform an orthogonal projection of this partial solution into latent space guided by the following equation

$$\mathcal{P}(\mathbf{x}) = \left(\mathbf{R}_{:,:,1} E_I E_I^T \mathbf{R}_{:,:,1}^T\right)^{-1} \mathbf{R}_{:,:,1} E_I \mathbf{x}\,, \tag{6}$$

where $E_I \in \mathbb{R}^{d \times n_{E_I}}$ contains the latent representations of all items. Thus, we have that a completion of $\mathbf{x}$ is given by

$$\hat{\mathbf{x}} = \mathcal{P}(\mathbf{x})^T \mathbf{R}_{:,:,1} E_I\,. \tag{7}$$

The values of the entries of $\hat{\mathbf{x}}$ can be interpreted as scores indicating whether or not the corresponding items are likely to be configured in the particular solution. With this interpretation, the items may be reordered in decreasing order of their scores and the ranking can then be used for recommendations.

Note that the matrix representation of the linear map $\mathcal{P}(\cdot)$ can be precomputed and stored for later use. Thus the computational complexity of the whole *projection* step is given by $\mathcal{O}(dN_{E_I})$. To the best of our knowledge, there is no equivalent, simple procedure for the class of translational models. Hence, they require re-training when a novel solution is set up by the user. This constitutes a major limitation in their applicability to our use case.

5 Real-World Experimental Study

5.1 Data

The experiments have been conducted on real-world data collected from Siemens internal projects. It can be roughly divided into two categories:

1. *Historical solutions*: This data contains (anonymized) information about automation solutions that have been previously configured. Only considering this part of the data (as it is done by some of the traditional recommender systems) would result in recommending items that have been configured together most frequently in the past.
2. *Descriptive features of the items*: These features may either specify the type of an item (e.g., panel, controller) or a certain technical specification (e.g., line voltage, size). An example of an entry in this part of the dataset would be a Siemens SIMATIC S7-1500 controller (with a unique identifier 6ES7515-2FM01-0AB0) being fit for fail-safe applications.

Some pre-processing steps have been applied before conducting our experiments. These include:

1. Removing solutions that contain only one item.
2. Removing duplicate descriptive features that were named in different languages.

3. Removing descriptive features that have unreasonably long strings as possible values. In most cases, these are artifacts of an automatic documentation processing procedure for legacy items.
4. Removing items with no known descriptive features or descriptive features with just one possible value.

After applying these pre-processing steps, we obtained 426,710 facts about 25,473 solutions containing 3,003 different items that share 3,981 different technical features among themselves. Figure 2 illustrates the three-way adjacency tensor obtained from the data: both rows and columns correspond to the entities (i.e., solutions, items, as well as all possible values of the descriptive features of the items), while each slice corresponds to one of the relations (with the first one being the *contains* relation).

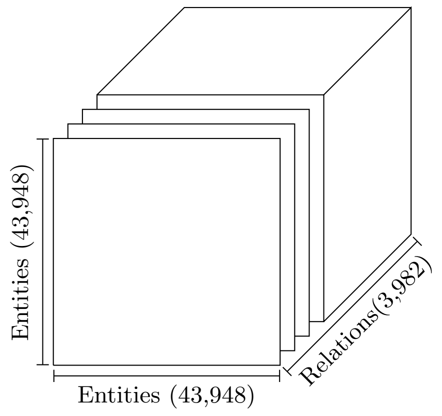


Fig. 2. The three-way adjacency tensor obtained after data pre-processing

5.2 Implementation

We compared RESCOM to state-of-the-art methods, namely TransE, TransR, TransD, and TransH, as well as NMF, considering its successful application in popular recommender systems. For experimental consistency, we re-implemented these methods within the same framework as our method, using Python and TensorFlow. All experiments were conducted on a Linux machine and the models were trained/tested on NVIDIA Tesla K80 GPUs. Training our model takes approximately 60 minutes for 50 epochs ($\sim$100 batches) on one GPU.

To further facilitate a fair evaluation, we generated the training (70%), validation (20%) and test (10%) sets once and used this fixed split in all experiments. In each case, the best set of hyperparameters were chosen based on the highest mean reciprocal rank on the validation set. The results are then reported for the chosen parameters on the test set.

Having tried optimizers such as stochastic gradient descent, Adam, and AdaGrad, we found that the Adam optimizer resulted in the best convergence in our case. The dimensionality of the embeddings was tuned from the range $\{10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. We also investigated the influence of the number of negative triples per positive training sample by choosing among $\{1, 2, 3, 4, 5, 10, 20\}$. The regularization parameter was chosen among $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. All experiments were executed for a maximum of 500 epochs, with a check for early stopping every 50 iterations.

5.3 Evaluation Scheme

The particular metrics used for the evaluation of recommender systems depend greatly on the characteristics of the data set and the ultimate purpose of recommendation. In the current setting, the goal is to evaluate the predictive accuracy, i.e., how well the recommender system can predict the likelihood of occurrence of items in partial solutions, in the context of the already configured items.

For evaluation of the various translational models, we followed the ranking procedure proposed by [5]. For each test triple (e_i, r_k, e_j), the object is removed and in turn replaced by each of the entities e from the set of entities $\mathcal{E}$. Dissimilarities of all those resulting triples (also referred to as corrupted triples) are first computed by the models and then sorted in ascending order; the rank of the correct entity is finally stored. Unlike [5], we do not repeat the procedure by replacing the subject entity. This is in accordance with the recommendation setting, where we are interested in obtaining a ranking for the items that only appear as object nodes in the *contains* relation.

To evaluate non-negative matrix factorization (NMF) and our own method, we employ a common setting where we construct an adjacency matrix of solutions and configured items with missing entries. We then perform an inverse transformation (in the case of NMF) or an orthogonal projection (for RESCOM - see Equations (6) and (7)) to obtain a completed matrix. The values in each row of this matrix are then interpreted as scores deciding whether or not the corresponding items are present in the particular solution, with a higher value implying a greater chance of the item being present. Next, the items in each row are reordered in decreasing order of their scores, yielding the ranking.

We report the mean of those predicted ranks, the mean reciprocal rank (MRR), and the hits@10%, i.e., the proportion of correct entities ranked in the top 10%. We consider the MRR owing to its robustness towards outliers, which is not the case with the mean rank. Further, a trend analysis indicating that users typically look at the top 10% of the displayed results before opting for a new search, forms the basis for choosing hits@10% for evaluation (see Section 5.4 for more details).

As noted by [4], these metrics can be flawed when some corrupted triples are in fact valid ones (i.e., are present in the training, validation, or test set), causing them to be ranked higher than the test triple. This however should not be counted as an error because both triples are actually true. To avoid such misleading behavior, we discard all those triples in the ranking process that appear either in the training, validation, or test set (except the test triple of interest). This ensures that none of the corrupted triples belong to the dataset and grants a clearer view of the ranking performance. We refer to this as the *filtered (filt.)* setting and the original (possibly flawed) one as the *raw* setting.

In Section 5.4, results are reported according to both raw and filtered settings.

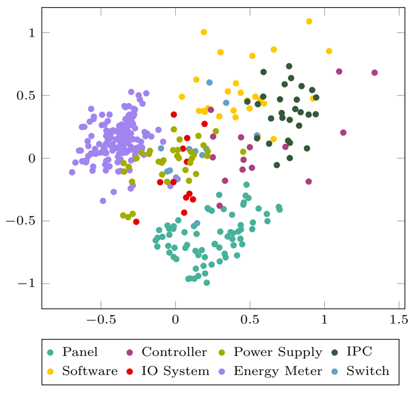
Evaluation Mechanism for Cold Start. We adapt the evaluation mechanism to specifically suit the scenario of the cold start problem. Experiments are run on the best hyperparameters chosen earlier, without the need for cross-validation. We introduce a set of new items into the dataset. In doing so, we only add technical features for these items, without including them in any solutions. Additionally, we ensure that items belonging to the same category as these new items are in fact available. This should allow the model to pick up features common among similar items. Thereafter, we carry out the same workflow as before for model training and testing and report the metrics on the test set.

5.4 Results

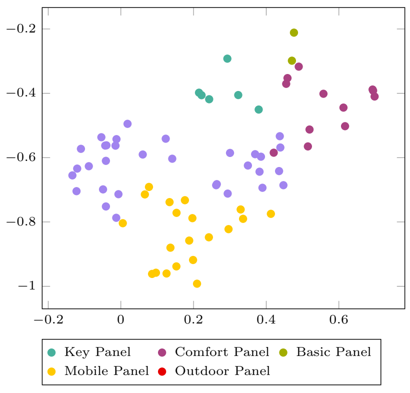
Table 1 displays the results of the recommendation task for all methods under consideration. The filtered setting produces lower mean rank and higher values of MRR and hits@10% for all the methods, as expected. As described in Section 5.3, this setting offers a better evaluation of the performance. RESCOM outperforms all methods in most of the standard metrics. However, NMF has the highest value of MRR in the filtering setting, even better than that of RESCOM. This may be explained by considering that MRR favours correct entities being retrieved exactly in a few cases, against correct entities being in the top-k ranks consistently. To verify if this affects our use case involving the industrial purchasing system, we further analyzed the number of items that users are willing to consider until they arrive at the desired product. This was done by looking at the filters applied by users at the time of solution configuration and noting the positions of selected items among the results. Trends show that users typically look at the top 10% of the displayed results before opting for a new search and since RESCOM still has the best hits@10%, we consider it to be superior to all the counterparts.

Table 1. Results of all evaluated methods on the test set

Metric	Mean Rank		Mean Reciprocal Rank		Hits@10%	
Eval. Setting	Raw	Filt.	Raw	Filt.	Raw	Filt.
TransR	898.42	894.58	0.02	0.02	0.29	0.32
TransE	322.40	318.58	0.09	0.10	0.70	0.72
TransD	332.62	328.82	0.09	0.12	0.68	0.71
TransH	316.04	312.22	0.09	0.10	0.69	0.72
NMF	182.38	177.84	0.12	**0.22**	0.82	0.87
RESCOM	**81.32**	**76.76**	**0.13**	0.18	**0.93**	**0.95**



(a) Items coloured by category



(b) Panels coloured by subcategory

Fig. 3. Item embeddings obtained using RESCOM after PCA (with 33.42% variance explained)

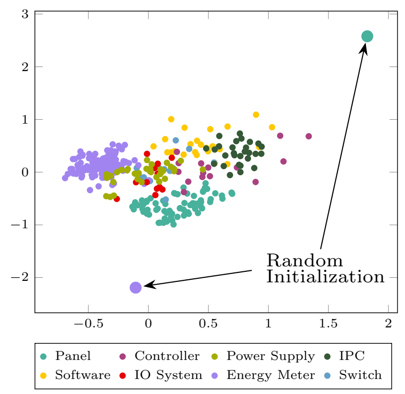
Furthermore, Figure 3 depicts a scatter plot of the item embeddings. After training a model using RESCOM (with 15 latent dimensions), the resulting embeddings were further subjected to dimensionality reduction via principal component analysis (PCA) to arrive at the 2-dimensional embeddings shown. In Figure 3(a), the items are coloured by *category*, which is one of the available technical features. As indicated by a few dense regions resembling clusters, it may be seen that the model learns the associations among the items for most categories. Figure 3(b) provides a more detailed view of the items in the Panel category. The items are coloured by subcategory and the values correspond to the subcategories within the Panel category. This demonstrates that our model is capable of learning the subspace embeddings as well, although we do not encode this hierarchical structure explicitly.

Cold Start: Model Evaluation Results. As explained in Section 5.3, we perform an explicit evaluation of the methods for the cold start scenario. Since it is not possible to test for previously unseen data in the case of NMF, we exclude it from consideration. The results are as shown in Table 2. We can clearly see that RESCOM performs better than all of the other methods, proving that it is capable of recommending even those items that were not previously configured.

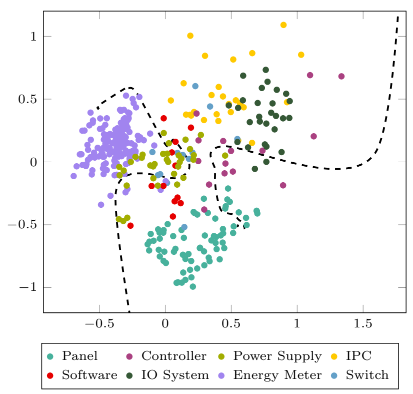
Further, we conducted an experiment aimed at testing the ability of the model to learn the embeddings for newly added items, while using the previously trained embeddings for old items. We added two new items – one belonging to the Panel category and another belonging to the Switch category. The model was then trained while keeping the embeddings of the previously trained items fixed. Figure 4(a) shows the two new randomly initialized items while the previous embeddings remain the same as in Figure 3(a). The new items were then trained in the usual way and we observed their embeddings during the training. After each

Table 2. Cold start: Results of all evaluated methods on the test set

Metric	Mean Rank		Mean Reciprocal Rank		Hits@10%	
Eval. Setting	Raw	Filt.	Raw	Filt.	Raw	Filt.
TransR	766.7216	766.1959	0.0044	0.0036	0.284	0.289
TransE	486.7938	482.2165	0.0983	0.1073	0.655	0.686
TransD	533.8299	529.3041	0.0982	0.1066	0.649	0.655
TransH	473.2113	468.7423	0.0993	0.1081	0.65	0.658
RESCOM	**127.5792**	**122.3405**	**0.1423**	**0.1801**	**0.884**	**0.887**



(a) Pre-trained embeddings for old items and 2 new items randomly initialized



(b) After training the new items for 50 epochs

Fig. 4. Cold start: Item embeddings obtained using RESCOM after PCA (with 33.42% variance explained)

epoch, we tracked these items to see if they moved closer to the clusters formed by the other items in the same categories. Figure 4(b) illustrates the embeddings obtained after training for 50 epochs. As evident from the trajectories shown, the model learns the associations among items of the same category quite well.

6 Conclusion

We proposed RESCOM, a multi-relational recommender system, and applied it in the context of an industrial purchasing system for setting up engineering solutions. In this domain, the necessity of expressive recommendation methods arises because the functionality of each solution is highly dependent on the interplay of its components and their technical properties. RESCOM aims to take this complexity into account by embedding all past solutions, items, and technical properties into the same vector space and modelling their relatedness via a set of bilinear forms. We conducted extensive experiments based on real-world data from the Siemens product database. To sum up, the main findings of our real-world study are:

– RESCOM significantly outperforms all the considered baseline methods in most of the common performance measures.
– RESCOM offers a natural remedy to the cold start problem. Even in the absence of any historical data for particular items, our method is able to produce reasonable recommendations.
– RESCOM requires minimal effort in terms of data preparation and maintenance. Moreover, training the model and computing recommendations can be performed efficiently. Apart from sorting the items, we have shown that recommending items boils down to a sequence of matrix multiplications. This ensures that our method can operate in real time when guiding the user to set up new solutions.

So far, we have considered semantic annotations of the configurable items, but ignored contextual information about the solutions such as their area of applications or temporal aspects. We will explore this direction in future work when the relevant data becomes available. Other possibilities for future research include extending this approach from recommendations to pairwise preferences that can be represented in preference graphs [19] and incorporating semantic knowledge using reasoning mechanisms [14].

Finally, we would like to stress that this work evolved from a real-world industrial R&D project at Siemens. In this regard, we have also evaluated our approach together with domain experts from Siemens. More specifically, we considered a set of typical solutions and tested whether our model produces comprehensible recommendations from an engineer's point of view. After having passed this test as well, we intend to take further steps towards integrating our method into one of the major solution configurators at Siemens.

Acknowledgements. We would like to thank Siemens Digital Factory Division for providing the data and helping us to get a better understanding of the application area, our colleague Martin Ringsquandl for the insightful discussions and helpful remarks on early drafts, as well as the head of our research group, Steffen Lamparter, for providing all the necessary resources.

References

1. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Recommender Systems Handbook, pp. 191–226. Springer (2015)
2. Bell, R.M., Koren, Y., Volinsky, C.: The Bellkor 2008 solution to the Netflix prize. Statistics Research Department at AT&T Research **1** (2008)
3. Billsus, D., Pazzani, M.J.: User modeling for adaptive news access. User Modeling and User-Adapted Interaction **10**(2-3), 147–180 (2000)
4. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems. pp. 2787–2795 (2013)
5. Bordes, A., Weston, J., Collobert, R., Bengio, Y., et al.: Learning structured embeddings of knowledge bases. In: AAAI. vol. 6, p. 6 (2011)

6. Burke, R.: Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction **12**(4), 331–370 (2002)
7. Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., Sartin, M.: Combining content-based and collaborative filters in an online newspaper (1999)
8. He, R., Kang, W.C., McAuley, J.: Translation-based recommendation. In: Proceedings of the 11th ACM Conference on Recommender Systems. pp. 161–169. ACM (2017)
9. Ji, G., He, S., Xu, L., Liu, K., Zhao, J.: Knowledge graph embedding via dynamic mapping matrix. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). vol. 1, pp. 687–696 (2015)
10. Krompaß, D., Baier, S., Tresp, V.: Type-constrained representation learning in knowledge graphs. In: International Semantic Web Conference. pp. 640–655. Springer (2015)
11. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: AAAI. vol. 15, pp. 2181–2187 (2015)
12. Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing **7**(1), 76–80 (2003)
13. Lops, P., De Gemmis, M., Semeraro, G.: Content-based recommender systems: State of the art and trends. In: Recommender Systems Handbook, pp. 73–105. Springer (2011)
14. Mehdi, G., Brandt, S., Roshchin, M., Runkler, T.: Towards semantic reasoning in knowledge management systems. In: IFIP International Workshop on Artificial Intelligence for Knowledge Management. pp. 132–146. Springer (2016)
15. Nickel, M., Tresp, V.: Logistic tensor factorization for multi-relational data. arXiv preprint arXiv:1306.2084 (2013)
16. Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: ICML. vol. 11, pp. 809–816 (2011)
17. Nickel, M., Tresp, V., Kriegel, H.P.: Factorizing yago: scalable machine learning for linked data. In: Proceedings of the 21st International Conference on World Wide Web. pp. 271–280. ACM (2012)
18. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review **13**(5-6), 393–408 (1999)
19. Runkler, T.A.: Mapping utilities to transitive preferences. In: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems. pp. 127–139. Springer (2018)
20. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: AAAI. vol. 14, pp. 1112–1119 (2014)
21. Yang, B., Yih, W.t., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. arXiv preprint arXiv:1412.6575 (2014)
22. Yang, Y., Esteban, C., Tresp, V.: Embedding mapping approaches for tensor factorization and knowledge graph modelling. In: International Semantic Web Conference. pp. 199–213. Springer (2016)