Matrix Factorization based Heuristics for Constraint-based Recommenders

Seda Polat Erdeniz
Graz University of Technology
Graz, Austria
spolater@ist.tugraz.at

Alexander Felfernig
Graz University of Technology
Graz, Austria
alexander.felfernig@ist.tugraz.at

Ralph Samer
Graz University of Technology
Graz, Austria
rsamer@ist.tugraz.at

Muesluem Atas
Graz University of Technology
Graz, Austria
muesluem.atas@ist.tugraz.at

ABSTRACT

The main challenges for recommender systems are: producing high quality recommendations and performing many real-time recommendations per second for millions of customers and products. This paper addresses both challenges in the context of constraint-based recommenders where users specify their requirements and the system recommends a solution. We propose a novel approach to determine value ordering heuristics on the basis of matrix factorization. As far as we are aware, no researches exist in constraint-based recommendation domain which exploit matrix factorization techniques. The main idea of our approach consists in the prediction of *value ordering heuristics* based on historical transactions which can either represent past customer purchases or requirements. Thereby, *value ordering heuristics* are computed which are specific to each user's requirements. A series of experiments on real-world datasets for calculating constraint-based recommendations has shown that our approach outperforms compared methods in terms of *runtime efficiency* and *prediction quality*.

CCS CONCEPTS

• **Computing methodologies → Heuristic function construction**;

KEYWORDS

Artificial Intelligence, Recommendation Systems, Knowledge-based Recommendation, Constraint-based Recommendation, Value Ordering Heuristics, Matrix Factorization.

ACM Reference Format:
Seda Polat Erdeniz, Alexander Felfernig, Ralph Samer, and Muesluem Atas. 2019. Matrix Factorization based Heuristics for Constraint-based Recommenders. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3297280.3297441

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297441

1 INTRODUCTION

Due to the new century's information overload problem, recommender systems [13] are becoming more important in various domains. Knowledge-based recommendation [3] attempts to suggest objects based on inferences about a user's needs and preferences. One type of knowledge-based recommender systems are *constraint-based recommenders* [19] in which the recommendation tasks can be composed of many variables and constraints. In these settings, achieving an acceptable runtime performance of recommending suitable items to users can quickly become a very challenging. Therefore, search should be guided by intelligent search strategies, so-called *heuristics* [8]. Among the existing heuristics, variable and value ordering heuristics are widely adopted in the context of constraint-based recommenders.

Determining accurate heuristics for constraint-based recommenders is challenging due to the following reasons:

- *Complexity.* Solving a constraint-based recommendation task on a finite domain is an NP-complete problem with respect to the domain size and the complexity of the given constraints.
- *Accuracy.* A consistent recommendation result should have a high probability to be *accepted by the user.*

In this paper, we propose a novel search heuristic-based approach for constraint-based recommenders to overcome the aforementioned two challenges. Our approach uses (historical) transactions of the configuration system to calculate value ordering heuristics which are specific to the requirements of a user. Thereafter, these heuristics are used to solve the recommendation task with a high *prediction accuracy* in a *short runtime.*

2 PRELIMINARIES

In this section, we provide a brief overview of definitions for constraint-based recommenders.

Constraint-based Recommenders: In the context of complex products such as cars, computers, real-estate, or financial services, constraint-based recommenders represent the standard solution [7]. Constraint-based recommenders exploit explicit user requirements as well as deep knowledge about the underlying product domain for the computation of recommendations. A constraint-based recommendation task can be defined as a constraint satisfaction problem (see Definition 2.1) and can be solved using so-called constraint satisfaction algorithms and heuristics.

Definition 2.1. **(Constraint-based Recommendation Task.)** In general, a recommendation task (RT) can be defined as a constraint satisfaction problem (V, C) where V is a set of variables, C is a set of system constraints which may also include a set of unary constraints representing concrete customer requirements REQ. An assignment of the variables in V is denoted as consistent recommendation result (REC) for a recommendation task (V, C) if REC does not violate any of the constraints in C.

Constraint Satisfaction Algorithms: Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used techniques are variants of *search-based*, *consistency-based*, and their hybrid methods [10]. However, solving a constraint satisfaction problem on a finite domain is an NP-complete problem with respect to the domain size and the complexity of the given constraints. Consequently, the *direct* use of one of these techniques can not help to increase the runtime performance. Therefore, more suitable heuristics are needed in order to take full advantage of the underlying power these algorithms can provide.

Constraint Satisfaction Heuristics: Heuristics stand for strategies using readily accessible information to control problem-solving processes. The constraint satisfaction community has developed a number of different heuristics. There exist several search heuristics which are applied in the context of constraint solving.

Searching for a solution of a given CSP consists of techniques for the systematic exploration of the space of all solutions. The basic brute force algorithm *generate and test* (also referred to as *trial and error*) search is solely based on the idea of testing every possible combination of values to obtain a solution of a CSP. The most popular heuristics for constraint solving are the variable and value ordering heuristics [9, 11, 14]. These heuristics guide the search to decide on a variable and a value to try in the next step.

3 RELATED WORK

Constraint based recommenders use constraint solving algorithms and heuristics to improve the overall runtime performance. However, these methods do not take the performance criterion *accuracy* into account. There exist some approaches which are based on collaborative filtering that aim to improve the accuracy of recommenders but lacks of runtime performance improvement at the same time.

Ardissono et al. [1] presented a framework for the personalized configuration in business-oriented domain. They classified the system users according to their expertise in the domain. The personalization of the interaction and dialogues are adapted according to this user modeling. They have applied this framework to a telecommunication switches domain. They experienced successful results (speed up in configuration process) in user configurations. However, this approach uses explicit personalization rules whereas we learn search heuristics from historical transactions.

Felfernig et al. [6] presented an overview of several techniques to solve constraint based recommendation tasks. An open research issue in this particular context is to efficiently guide the solution search towards the relevant items.

Sandvig et. al [15] adapted association rule mining to collaborative filtering in order to discover valuable patterns between items that have similar ratings. Using item ratings such as *like* or *dislike*, the authors could achieve a good level of accuracy in the results. Although the achieved accuracy comes very close to the compared method *k-nearest neighbor*, the paper does not cover the performance aspect in more detail.

Zanker [18] proposed a system that learns rule-based preferences from successful interactions in historic transaction data. The author uses collaborative filtering to derive preferences from a user's nearest neighbors. The paper demonstrates that this approach can improve the overall prediction accuracy of the recommender system. However, no stronger focus is put on the runtime performance aspect in this work.

Another work of Zanker [20] presented an approach that ranks the recommended items according to their degree of constraint fulfillment. They used historical transactions to evaluate the prediction quality of the recommendation results. This work focuses on relaxation of low weighted constraints whereas we satisfy all the user constraints in our approach. Even the constraint relaxation may help to reduce the runtime, there is no explicit runtime performance evaluation of the proposed approach in [20].

In conclusion, all discussed related work and all of the mentioned state-of-the-art techniques lack of providing improvements in terms of *runtime performance* and *prediction accuracy* at the same time. Therefore, we introduce our proposed method to improve constraint-based recommendation in terms of both *runtime performance* as well as *prediction accuracy*.

4 A MOTIVATING EXAMPLE

In order to demonstrate our proposed method, we present an example of an online personalized bike shop. We use a bike recommendation task (according to Definition 2.1) which is a small part of a real world knowledge base[1].

The concrete recommendation task of our bike shop example is denoted as RT_{bike}. Thereby, the task is composed of variables and constraints of a personalized bike shop (see Table 1). Furthermore, the bike shop does not offer a fixed product catalog. Instead, it offers any kind of personalized bike which complies with the constraints C_{bike} listed in Table 1.

	RT_{bike}
V_{bike}	$frame\_biketype = \{0, .., 4\}$, $frame\_internal = \{0, .., 1\}$, $extra\_propstand = \{0, .., 2\}$, $gear\_internal = \{0, .., 1\}$
C_{bike}	$c_1 : (frame\_biketype = (1 \vee 2)) \implies (frame\_internal = 1)$, $c_2 : (frame\_biketype = 4) \implies (gear\_internal = 0)$, $c_3 : (frame\_internal = 0) \implies (gear\_internal = 0)$

Table 1: A constraint-based recommendation task RT_{bike}

Table 2 shows an example of different transactions in the context of the given recommendation task RT_{bike}. In this example, there are six users: Alice, Bob, Tom, Ray, Joe and Lisa. The first five users interacted with the online personalized bike shop in the past. The last user Lisa is an active user who has not left the online shop yet and has defined own preferences. In this table, the transaction type is denoted as TRX and explained in the remainder of this section.

[1] https://www.itu.dk/research/cla/externals/clib/bike2.cp

User name:	Alice	Bob	Tom	Ray	Joe	Lisa
TRX ID:	TRX_{Alice}	TRX_{Bob}	TRX_{Tom}	TRX_{Ray}	TRX_{Joe}	TRX_{Lisa}
$frame\_biketype$	3	4	0		2	2
$frame\_internal$	1	0		1		1
$extra\_propstand$	1	1	1			
$gear\_internal$	1	0	1	1	1	
TRX type:	HT1	HT1	HT2	HT2	HT3	AT

Table 2: Transactions based on RT_{bike}

We categorize the transactions into four subtypes as follows:

Historical Transaction 1 (HT1): HT1 represents a complete and historical transaction (i.e., a complete past purchase). A transaction of this type can later serve as valuable input for recommendations in the future. Table 2 shows some examples of HT1 transactions. The purchases of Alice and Bob represent complete transactions.

Historical Transaction 2 (HT2): Another possible scenario refers to the situation where a user may leave the online shop without having purchased a recommended product. Such transactions are incomplete and denoted as HT2. These transactions represent stored incomplete transactions which only contain user requirements but not a complete specification of a product. In Table 2, Tom and Ray left the system without having finished to configure a bike. In other words, the configuration session was aborted before the transaction was completed.

Historical Transaction 3 (HT3): Another scenario refers to situations where new product features/services are introduced to existing products for which some complete and incomplete transactions of type HT1 and HT2 already exist. With the introduction of new product features/services new columns are added to the *Transaction Matrix*. These new columns cause new blank entries in the matrix for all existing transactions (of type 1 and 2). Consequently, all existing transactions of type 1 and 2 are converted into transactions of type HT3. In Table 2, Joe had purchased a personalized bike at a time when both product features $frame\_internal$ and $extra\_propstand$ did not exist. After these two features were introduced the former HT1 transaction of Joe was converted to a transaction of type HT3.

Active Transaction (AT): Whenever a new user starts a new configuration session, he or she will receive instant recommendations while he or she is defining requirements in the online bike shop. This scenario is referred to as an active transaction. Likewise HT2 and HT3, an active transaction appears as an incomplete row in the matrix (see example in Table 2. In sharp contrast to HT2 and HT3, the transaction is active (i.e., ongoing) and the user can receive instant recommendations based on the requirements provided by him or her.

5 THE PROPOSED METHOD

The mentioned two main challenges for constraint-based recommender systems are *producing high quality recommendations* and *delivering instant recommendations for online recommendation tasks*. All discussed related work and state-of-the-art techniques lack of providing performance improvements in terms of *runtime performance* and *prediction accuracy* at the same time. The main objective of our approach is to decrease the runtime in accordance with the increase of prediction quality for constraint-based recommender systems.

Our proposed method calculates value ordering heuristics with respect to the formal definition of the recommendation task (see Formula 2.1). These heuristics are calculated based on the historical transactions data. We gather the data into a matrix. This transactional data matrix is sparse because of the variants of the transactions as mentioned in Section 4. Therefore, we apply matrix factorization to obtain a dense matrix. Using the pre-calculated dense matrix, we calculate value ordering heuristics for a recommendation task which decreases the runtime and increases the prediction quality.

In this section, we show how our approach can be applied in such a way that it can find a consistent recommendation for each recommendation task. Our approach consists of two phases: an *offline phase* and an *online phase*. In the *offline phase*, the prerequisites of the calculations are completed. This phase is applied only once, not before solving each recommendation task. In the *online phase*, based on a given user requirement set *REQ*, *REQ*-specific value ordering heuristics are calculated and used in the recommendation search.

5.1 Offline Phase

Prerequisites of the calculations in our approach are gathering transactional data into a sparse bitmap matrix and using it to obtain a dense bitmap matrix. The dense bitmap matrix is calculated using matrix factorization techniques.

5.1.1 Building the bitmap matrix. In the first step of the offline phase, a sparse bitmap matrix, which holds all types of transactions, is generated. In our approach, we even take advantage of HT2/HT3 type (incomplete) transactions by including them besides the list of HT1 type (complete) transactions. We find predicted products for HT2/HT3 transactions in the offline phase and directly recommend those products in the online phase when they come back to the online shop. Furthermore, by including this valuable extra information, the prediction quality of our approach can be further increased.

We use a bitmap matrix [4, 12] to hold the transactions. During the online phase, this bitmap matrix is used in order to obtain the value ordering heuristics. Each row of the matrix represents a *transaction*. Each column of the matrix represents a domain value of a variable. The bitmap matrix contains only 0s and 1s (i.e., bits). In a row (transaction), 1s mean that the variables hold as values the *corresponding* values of these columns. In return, all of the other domain values of these variables are set to 0.

For example, as shown in Table 4, the transaction of Alice is mapped into bits where all bits represents a value of the domain of the variable. For example, $frame\_biketype$ has a domain with five values (0,..,4), so its bitmap value has five bits (1s and 0s). $frame\_biketype$ is set to 3 by Alice, so in the bitmap matrix the corresponding bit (3rd bit) is set to 1 and others (1st, 2nd, and 4th bits) to 0.

In the working example, we convert all transactions in Table 2 to their corresponding bitmaps as shown in Table 4. For example, the transaction of Alice is mapped into bits where all bits represents a value of the domain of the variable. For example, $frame\_biketype$

	frame_biketype					frame_internal		extra_propstand			gear_internal	
	0	1	2	3	4	0	1	0	1	2	0	1
	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Alice	0	0	0	1	0	0	1	0	1	0	0	1
Bob	0	0	0	0	1	1	0	0	1	0	1	0
Tom	1	0	0	0	0			0	1	0		
Ray						0	1				0	1
Joe	0	0	1	0	0						0	1

Table 4: The sparse matrix M_TRX_{bike}.

has a domain with five values (0,..,4), so its bitmap value has five bits (1s and 0s). $frame\_biketype$ is set to 3 by Alice so in the bitmap version the corresponding bit (3rd bit) is set to 1 and others (1st, 2nd, and 4th bits) to 0.

5.1.2 Decomposing the sparse bitmap matrix. In this step, we decompose the sparse bitmap matrix which consists of all types (HT1, HT2, and HT3) of transactions.

In our working example, M_TRX_{bike} is a sparse bitmap matrix composed of transactions of RT_{bike} where each row represents a transaction of a user and each column represents a value of a variable. In terms of *matrix factorization*, the sparse matrix M_TRX_{bike} is decomposed into a $m \times k$ *user-feature matrix* and a $k \times n$ *value-feature matrix* which both contain the relevant information of the sparse matrix. Thereby, k is a variable parameter which needs to be adapted accordingly depending on the internal structure of the given data.

As shown in Table 3, an estimated dense matrix $M\_TRX'_{\text{bike}}$, a user-feature and a value-feature matrices are calculated by applying matrix factorization based on singular-value decomposition (SVD) [2][2]. The matrices M_UF_{bike} and M_VF_{bike} will be used in the online phase to calculate the recommendation task specific value ordering heuristics.

5.2 Online Phase

In the online phase, the personalized bike shop recommender system receives the requirements of a user and searches for a consistent recommendation. In order to guide the search, we calculate recommendation task specific value ordering heuristics using the dense bitmap matrix which is calculated in the offline phase. At the end of this phase, a consistent recommendation result is provided to the user.

[2]we set the parameters in SVD as $latentfactors = 3$ and $iterations = 1000$

5.2.1 Calculating the value ordering heuristics for HT2/HT3. Since HT1 is a complete (purchase) transaction, there is no need to predict a recommendation for such transactions. However, for the incomplete transactions HT2 and HT3, we can provide recommendations. When a user of incomplete historical transactions comes back to the online system again, the corresponding transaction in the dense matrix can be used as a value ordering heuristics to guide the search of consistent recommendation.

For example, when *Ray* revisits the online bike shop, the related value ordering heuristics is calculated using the estimated transaction of *Ray* in the dense matrix $M\_TRX'_{\text{Ray}}$. For frame_biketype in $M\_TRX'_{\text{Ray}}$, we have the probabilities as {0.6, -0.5, 0.5, 0.5, 0.1} where the third and fourth probabilities are the highest. Thus, for $M\_TRX'_{\text{Ray}}$, the value ordering heuristic for frame_biketype becomes an ordered set: {2, 3, 0, 4, 1}. The values in this set indicate the assignment order of variable values used by the constraint solver.

5.2.2 Calculating the value ordering heuristics for AT. In order to calculate a value ordering heuristics for AT, we can not use the dense matrix since Lisa's transaction does not yet exist in the transaction matrix. If we include Lisa in the dense matrix $M\_TRX'_{\text{bike}}$ and again decompose it, the runtime would not be feasible for a real-time recommendation task.

Therefore for AT, we select the k nearest neighbors [5] (k most similar historical transactions) from the dense matrix of transactions. Then the corresponding k user factors from M_UF are aggregated into one user factors which is called predicted user factors for AT ($M\_UF'_{\text{AT}}$). Then we multiply $M\_UF'_{\text{AT}}$ with the value features matrix M_VF to obtain a predicted transaction for AT ($M\_TRX'_{\text{AT}}$).

We use Euclidean Distance [17] based similarity to find k most similar historical transactions to AT as shown in Formula 1. M_TRX' is a historical transaction from the dense matrix and M_TRX_{new} is the bit-mapped requirements of the active user. b_{i} represents the ith bit of the bitmap matrix. The bits of M_TRX_{new} and M_TRX' are compared on the basis of the instantiated bits in the M_TRX_{new}. Therefore n stands for the total number of instantiated bits in M_TRX_{new} and i stands for the index of the instantiated bits in M_TRX_{new}.

$$\Delta(M\_TRX_{\text{new}}, M\_TRX') = \sqrt{\sum_{i=1}^{n} \|M\_TRX_{\text{new}}.b_{\text{i}} - M\_TRX'.b_{\text{i}}\|^2} \quad (1)$$

(a) User factors M_UF_{bike}

	$uf1$	$uf2$	$uf3$	$uf4$	$uf5$	$uf6$
Alice	0.1	0.3	0.3	-0.4	0.4	0.5
Bob	0.1	0.3	0.3	-0.1	-0.4	0.7
Tom	0.1	0.3	0.4	-0.1	0.2	0.6
Ray	0.2	0.3	0.5	-0.5	0.4	0.4
Joe	0.1	0.3	0.4	-0.7	-0.2	0.2

(b) Value factors M_VF_{bike}

	$vf1$	$vf2$	$vf3$	$vf4$	$vf5$	$vf6$
v1	1	1	0.2	0.3	0.3	0.1
v2	1	1	-1.5	0.2	0	-0.3
v3	1	1	0.1	-0.4	-0.2	-0.3
v4	1	1	-0.5	-0.1	0.5	-0.1
v5	1	1	-0.2	0.3	-0.5	0.2
v6	1	1	0.4	0.2	-0.6	0.3
v7	1	1	1.3	-0.4	0.7	0.2
v8	1	1	-1.3	0.2	-0	-0.3
v9	1	1	3.4	-0.4	0.1	0.8
v10	1	1	-1.3	0.2	-0	-0.3
v11	1	1	-1.5	0.2	-0	-0.3
v12	1	1	3.7	-0.6	-0	0.7

(c) Dense matrix $M\_TRX'_{\text{bike}}$

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Alice	0.5	-0.3	0.3	0.4	0.1	0.3	1.3	-0.2	2	-0.2	-0.3	2
Bob	0.4	-0.3	0.3	0	0.6	0.9	0.6	-0.2	1.9	-0.2	-0.3	2
Tom	0.5	-0.3	0.2	0.3	0.3	0.5	1.1	-0.3	2.1	-0.3	-0.4	2.1
Ray	0.6	-0.5	0.5	0.5	0.1	0.5	1.7	-0.4	2.7	-0.4	-0.5	2.9
Joe	0.3	-0.4	0.6	0.2	0.2	0.6	1.1	-0.3	2.1	-0.3	-0.4	2.3

Table 3: Estimating the dense matrix $M\_TRX'_{\text{bike}}$ where latent factor=6. User factors (uf1..uf6) and value factors (vf1..vf6) are representing user and value features according to the defined latent factor.

For the working example, to find the most similar transactions, first we need to convert REQ_{Lisa} into a bitmap form (M_TRX_{new}) as shown in Table 1.

$$M\_TRX_{\text{Lisa}} = [\ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ \_\ \ \_\ \ \_\ \ \_\ \ \_\ ]$$

Figure 1: The active transaction of Lisa (M_TRX_{Lisa})

We need to select k most similar transaction of M_TRX_{Lisa} from the rows of the dense matrix $M\_TRX'_{\text{bike}}$. In this working example, we set $k = 3$ to make it easy to present in the paper. In order to find three most similar transactions to M_TRX_{Lisa}, we find euclidean distance between TRX_{Lisa} and each row of $M\_TRX'_{\text{bike}}$ based on the instantiated bits of M_TRX_{Lisa}.

transactions in $M\_TRX'_{\text{bike}}$	distances to $\text{M\_TRX}_{\text{Lisa}}$
$\text{M\_TRX'}_{\text{Alice}}$	**2.03**
$\text{M\_TRX'}_{\text{Bob}}$	**1.95**
$\text{M\_TRX'}_{\text{Tom}}$	2.07
$\text{M\_TRX'}_{\text{Ray}}$	2.43
$\text{M\_TRX'}_{\text{Joe}}$	**1.89**

Table 5: Euclidean distances to M_TRX_{Lisa}

According to the calculated distances in Table 5, the most similar transactions to M_TRX_{Lisa} are selected as M_TRX_{Alice}, M_TRX_{Bob}, and M_TRX_{Joe} since they have the shortest distances to M_TRX_{Lisa}. In order to find the predicted complete transaction $M\_TRX'_{\text{Lisa}}$, the user features of Alice, Bob, and Joe are taken from M_UF_{bike} to be aggregated then the aggregated matrix represents the predicted user features of Lisa $M\_UF'_{\text{Lisa}}$. Then $M\_UF'_{\text{Lisa}}$ is multiplied with M_VF_{Lisa} in order to find the predicted transaction of Lisa $M\_TRX'_{\text{Lisa}}$.

$$\begin{bmatrix} M\_UF_{\text{Alice}} \\ M\_UF_{\text{Bob}} \\ M\_UF_{\text{Joe}} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.3 & 0.3 & -0.4 & 0.4 & 0.5 \\ 0.1 & 0.3 & 0.3 & -0.1 & -0.4 & 0.7 \\ 0.1 & 0.3 & 0.4 & -0.7 & -0.2 & 0.2 \end{bmatrix}$$

$$M\_UF'_{\text{Lisa}} = [\ 0.1 \quad 0.3 \quad 0.3 \quad -0.4 \quad -0.1 \quad 0.5\ ]$$

Figure 2: The predicted user-features of Lisa ($M\_UF'_{\text{Lisa}}$)

The predicted user-feature matrix of Lisa $M\_UF'_{\text{Lisa}}$ is multiplied with the value-feature matrix M_VF_{bike} in order to find the value ordering heuristics for RT_{Lisa} as shown in Figure 3.

$$M\_TRX'_{\text{Lisa}} = M\_UF_{\text{Lisa}} x M\_VF_{\text{bike}}$$

$$[\ 0.4 \quad -0.2 \quad 0.7 \quad 0.3 \quad 0 \quad 0.3 \quad 1.2 \quad -0.1 \quad 1.7 \quad -0.1 \quad -0.2 \quad 1.9\ ]$$

Figure 3: The predicted transaction for Lisa ($M\_TRX'_{\text{Lisa}}$)

Using the predicted complete transaction for Lisa $M\_TRX'_{\text{Lisa}}$ as a matrix of probabilities for values (as used for Ray in Section 5.2.1), we obtain the value orderings to solve RT_{Lisa} as shown in Table 6.

value ordering for frame_biketype:	**2**, 0 ,3, 4, 1
value ordering for frame_internal:	**1**, 0
value ordering for extra_propstand:	**1**, 0, 2
value ordering for gear_internal:	**1**, 0

Table 6: Value ordering heuristics to solve RT_{Lisa}

5.2.3 Searching for a consistent recommendation. The predicted transaction (in the form of bitmap matrix) holds the probabilities of the domain values of each variable for RT_{new}. However, the highest probabilities in the predicted transaction matrix (*candidate recommendation*) may be inconsistent. Therefore, the highest values in the predicted transaction can not be recommended directly before checking their consistencies. A consistent recommendation can be found by a CSP solver. In order to guide the search of the CSP solver, we provide our calculated value ordering heuristics to solve the corresponding recommendation task.

Our approach calculates the consistent recommendation result using the probabilities in the predicted transaction matrix as value ordering heuristics. Using the given value ordering heuristics in Table 6, a consistent recommendation result is found by the CSP solver. As seen in Table 7, the consistent recommendation holds the first values from the value orderings in Table 6 which means the recommendation is found within a minimum search space. Moreover, if this recommendation is purchased by Lisa, the prediction accuracy becomes 1.

TRX_{Lisa}	AT	Consistent Recommendation
$frame\_biketype$	2	**2**
$frame\_internal$	1	**1**
$extra\_propstand$		**1**
$gear\_internal$		**1**

Table 7: Consistent recommendation result for TRX_{Lisa}

6 EXPERIMENTAL EVALUATION

In this section, we conduct a series of experiments to evaluate the effectiveness of the proposed approach for improving runtime performance and prediction accuracy for constraint-based recommendation. We first describe the experimental settings including the knowledge bases, evaluation criteria and parameters of tests. Then, we demonstrate the performance of our approach compared to the available methods.

6.1 Knowledge bases

We have used publicly available real-world knowledge based and datasets from *Configuration/Diagnosis Benchmarks in Choco (CDBC)*[3]. The number of historical and active transactions in each knowledge base is presented in detail in Table 8.

Bike knowledge base. We have used a real-world bike knowledge base from *CDBC* to evaluate the runtime performance of our approach. Using this knowledge base, there can be an online personalized bike shop where users can define their preferences on the 34 variables to generate their own personalized bike. After their

[3]https://github.com/CSPHeuristix/CDBC

specification, a personalized bike is recommended on the online system. Then, they can order this recommended bike or still make new changes on it then order the next recommendation.

PC knowledge base. We have used a real-world personal computer (PC) knowledge base from *CDBC*. Using this knowledge base, there can be an online personalized PC shop where users can define their preferences on the 45 variables to generate their own personalized PC. After their specification, a personalized PC is recommended on the online system. Then, they can order this recommended PC or still make new changes on it then order the next recommendation.

Camera knowledge base. To be able to evaluate the prediction accuracy, we used a real-world digital camera dataset (*CameraKB_ConfigurationDataset*) from *CDBC* which includes historical transactions based on the real product catalog from a digital camera online shop. This dataset contains 264 transactions which contains a list of user requirements and corresponding user's preferred product IDs from the product catalog. Among these 264 transaction, we have used 200 of them as historical transactions and 64 of them as active transactions. Even though the dataset is rather small, it is very important for our prediction accuracy tests since there is no other similar real-world dataset available publicly.

6.2 Evaluation Criteria

We compare the performance of our approach with other heuristics (given in Section 6.4) based on the following two performance criteria: time (τ) and accuracy (π).

Runtime (τ). Runtime, as one of our performance indicators, represents the time spent in between collecting user requirements and providing the recommendation result (n represents the number of recommendation tasks, see Formula 2).

$$\tau = \frac{1}{n} \times \sum_{i=1}^{n} runtime(RT_{\mathrm{i}}) \tag{2}$$

For the evaluation of our approach, runtime is calculated by dividing the time spent in the online phase (see Section 5.2) by the number of recommendation tasks.

Accuracy (π). The recommended product (*REC*) can be purchased by the user or not. If the purchased product (*PP*) has the same values as *REC*, then *REC* is considered as a correct recommendation *CR*, otherwise *REC* is not a correct recommendation *NR*. As shown in Formula 3, a correct recommendation has all the values equal to the corresponding values in the purchased product.

$$\pi = \frac{\#(\mathrm{CR})}{\#(\mathrm{NR})+\#(\mathrm{CR})} \tag{3}$$

We have used only the camera knowledge base for evaluating the prediction quality. In order to calculate π, we have compared the recommendation result with the selected product of the user.

6.3 Test Parameters

Our experiments are executed on a computer with an Intel Core i5-5200U, 2.20 GHz processor, 8 GB RAM and 64 bit Windows 7 Operating System and Java Run-time Environment 1.8.0. Constraint satisfaction is applied by a Java based CSP solver *choco-solver*[4]. For decomposing the bitmap matrix, we have used the *SVDRecommender* of Apache Mahout library [16] with a latent factor k=100, number of iterations =1000, and number of recommended items is set to number of total variables of the recommendation task. For finding the estimated transaction, three most similar transactions are selected from the historical transactions.

All parameters in the test cases are given in Table 8. *Size of search space* of each knowledge base is calculated by multiplying domain sizes of all variables. r is the number of unary constraints in the user requirements. Each unary constraint includes a variable assigned with a random value from the corresponding domain. *# of value ordering heuristics* includes also our approach besides compared six other value ordering heuristics: *IntDomainMin, IntDomainMax, IntDomainMedian, IntDomainMiddle, IntDomainRandom, IntDomainRandomBound*. In accuracy tests, only our approach is used as used value ordering heuristic. Therefore it is set to 7 in runtime tests. *# of similarity measures* are the number of compared methods in accuracy tests which are *Euclidean Distance Similarity, Pearson Correlation, Cosine Similarity, and Tanimoto Coefficient Similarity*. *# of test cases* for runtime tests are calculated by multiplying the number of variable ordering heuristics with the number of value ordering heuristics which gives all number of combinations. *# of test cases* for accuracy tests are calculated by multiplying the number of r with the summation of number of similarity measures and number of value ordering heuristics.

6.4 Comparative Approaches

Our approach is learning value ordering heuristics for constraint solvers to solve a constraint-based recommendation task efficiently in terms of *runtime* and *prediction accuracy*. It combines the techniques of constraint satisfaction and collaborative filtering. Therefore, we have compared our approach both with constraint satisfaction heuristics and collaborative filtering based approaches.

6.4.1 Baselines for runtime performance. In order to evaluate the *runtime performance* (τ), we compared our approach with six value ordering heuristics of *choco-solver*[5] (for integers) as following: **IntDomainMin, IntDomainMax, IntDomainMedian, IntDomainMiddle, IntDomainRandom, IntDomainRandomBound**. Choco solver has also 9 variable ordering heuristics: *Smallest, Largest, FirstFail, AntiFirstFail, Occurrence, InputOrder, DomOverWeg, GeneralizedMinDom, Random*. We have tested all combinations of value-variable ordering heuristics of choco-solver.

6.4.2 Baselines for prediction accuracy. To evaluate prediction accuracy (π), we have compared our approach with user neighborhood (*k nearest neighbor*) based recommenders which use **euclidean distance similarity, Pearson correlation, cosine similarity, and Tanimoto coefficient similarity** from the collaborative filtering library of Apache Mahout [16]. Comparative collaborative filtering approaches have calculated the recommendation results using the AT and HT1 transactions of the camera dataset. For each approach, the recommendation result is compared with the purchased product ID of the corresponding historical transaction of the active transaction. If they are same, the accuracy is set to 1, if not to 0.

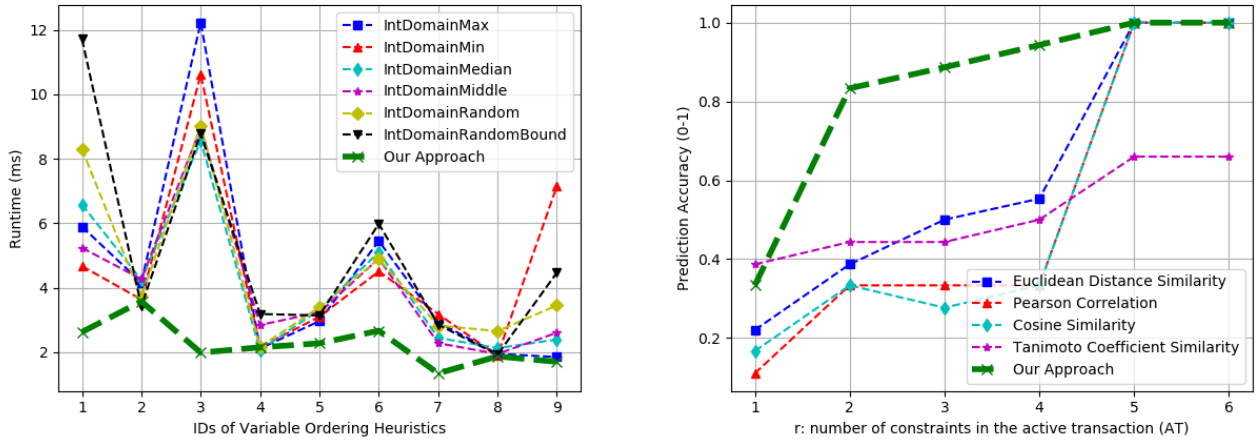
[4] http://www.choco-solver.org/

[5] http://www.choco-solver.org/apidocs/index.html

Parameters/Features	Bike KB* Runtime Tests	PC KB* Runtime Tests	Camera KB* Runtime Tests	Camera KB* Accuracy Tests
# of HT1	80**	50**	150***	150***
# of HT2/HT3	20**	50**	50***	50***
# of AT	50**	50**	64***	64***
# of variables	31	45	10	10
# of constraints	32	42	20	20
size of the search space	7.92×10^{24}	5.07×10^{34}	1.02×10^{7}	1.02×10^{7}
# of user constraints (r)	3	3	3	{1,2,3,4,5,6}
# of variable ordering heuristics	9	9	9	1
# of value ordering heuristics	7	7	7	1
# of similarity measures	-	-	-	4
# of test cases	63	63	63	35
# of runs on each case	50	50	50	50
# of total runs	3150	3150	3150	1750

*real world knowledge base, **synthetic transactions dataset, ***real world transactions dataset

Table 8: Parameters of the runtime performance and prediction accuracy tests.



(a) Runtime Performance (τ)

(b) Prediction Accuracy (π)

Figure 4: Performance results in terms of τ and π. In runtime performance comparisons (a), we have used the variable ordering heuristics with the following IDs: #1: Smallest, #2: Largest, #3: FirstFail, #4: AntiFirstFail, #5: Occurance, #6: InputOrder, #7: DomOverWDeg, #8: GeneralizedMinDomain, #9:Random. For the runtime performance evaluation, baseline approaches are the mentioned six value ordering heuristics, whereas for the prediction accuracy evaluation, the baseline approaches are the KNN recommenders based on four different similarity techniques. In both graphs, our approach is shown in a bold line.

6.5 Results

We have used real-world knowledge bases and transactions datasets in various test cases as shown in Table 8. Thereby, we have compared our approach with constraint satisfaction and collaborative filtering based approaches. We show the evaluated performance results (see Figure 4) in terms of two performance indicators runtime (τ), and prediction quality (π).

6.5.1 Evaluation of runtime performance. Figure 4(a) presents a comparison between our approach and six value ordering heuristics. Represented runtime values are the averages of the runtime results of all the tests on each of three datasets.

Each value ordering heuristic is combined with nine different variable-ordering heuristics (as described in Section 6.4) to set the search strategy of choco-solver. As observed, in some combinations of heuristics, the runtime results are very similar with our approach, especially when the variable ordering heuristic is set as Largest, Antifirstfail and GeneralizedMinDomain.

However, our approach outperforms all built-in value-ordering heuristics of Choco-Solver on the basis of all variable ordering heuristics combinations. The best performance of our approach is observed as 1.34 ms when it is combined with the variable ordering heuristic *DomOverWDeg* (#7).

6.5.2 Evaluation of accuracy. In Figure 4-b, we have compared the prediction quality of the recommendation results of our approach and mentioned collaborative filtering approaches in Section 6.4. In accuracy evaluations, we have used the camera knowledge base since the only real transactions dataset is collected based on this knowledge base.

We have tested each approach on the basis of r values where r represents the number of constraints in AT. According to the results, we have observed that when r increases the accuracy also increases. When r is set to 10, the user requirements itself is the same with the recommendation result. That is because there are only 10 variables in the camera knowledge base and r=10 means the user specifies 10 different variables.

As observed in our real transactions dataset, user requirements include in average r=3 constraints and the most of the them are with r=(2, 3, 4). For these most common cases r=(2, 3, 4), our approach finds recommendations with the best prediction qualities than all of the compared collaborative filtering approaches. For example, when r=2, the accuracy of our approach is 0.83 where the next best approach (Tanimoto coefficient similarity) has the accuracy 0.44.

7 CONCLUSIONS AND FUTURE WORK

In particular, constraint-based recommenders are applicable in situations where there are large and potentially complex product assortments and/or cold-starting users. In such situations standard collaborative and content-based filtering techniques are known to encounter serious limitations. For such recommenders, the quick generation of recommendations within a reasonable time (i.e., before users leave the web page) can be very challenging if recommendation results should be in a high prediction quality as well.

In this paper, we have proposed a novel value-ordering heuristics for constraint-based recommenders which employs matrix factorization techniques and historical transactions. As far as we are aware, no researches exist in constraint-based recommendation domain which exploit matrix factorization techniques. Our approach calculates a historical transactions matrix in the offline phase. This matrix is used to estimate a dense transaction for each new active transactions. Estimated dense transaction is used as a value ordering heuristics for searching a recommendation result in the online phase. According to our experimental results on the basis of real-world constraint-based knowledge bases (PC, bike, and camera), our approach outperforms the compared value ordering heuristics in terms of *runtime efficiency*. Moreover, to be able to evaluate *prediction quality*, we used real historical transactions based on camera knowledge base and we have observed that our approach outperforms nearest neighbor based collaborative filtering methods.

We have employed nearest neighbor based collaborative filtering approaches as prediction accuracy baselines but could not use other constraint-based recommendation approaches since these implementations are not yet available in commonly used recommendation libraries like Apache-Mahout. However, as future work, we would like to compare our approach with other constraint-based recommenders as well.

We have used user constraints which include exact value assignments (e.g. $frame\_biketype$=1). However, in many real world cases, user requirements do not have exact value assignments but a group of preferred values (e.g. $frame\_biketype$= 1 *or* 2). Moreover, there can be also combined constraints in user requirements (e.g. *if* $frame\_biketype$=1 *then* $gear\_internal$=1). As a future work, we plan to focus on covering these kind of constraints as well. By this way, we can provide a full solution for real-world configuration systems which assist its users online with high quality recommendation results within a reasonable response time.

ACKNOWLEDGMENTS

The work presented in this paper has been conducted within the scope of the Horizon 2020 projects OpenReq (Grant Nr. 732463) and AGILE (Grant Nr. 688088).

REFERENCES

[1] Liliana Ardissono, Alexander Felfernig, Gerhard Friedrich, Anna Goy, Dietmar Jannach, Markus Meyer, Giovanna Petrone, Ralph Schäfer, Wilken Schuetz, Markus Zanker, et al. 2002. Personalising on-line configuration of products and services. In *ECAI*, Vol. 2. 225–229.

[2] Christos Boutsidis and Efstratios Gallopoulos. 2008. SVD based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition* 41, 4 (2008), 1350–1362.

[3] Robin Burke. 2002. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction* 12, 4 (2002), 331–370.

[4] Chee-Yong Chan and Yannis E Ioannidis. 1998. Bitmap index design and evaluation. In *ACM SIGMOD Record*, Vol. 27. ACM, 355–366.

[5] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[6] Alexander Felfernig and Robin Burke. 2008. Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th international conference on Electronic commerce*. ACM, 17–26.

[7] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. 2015. Constraint-based recommender systems. In *Recommender Systems Handbook*. Springer, 161–190.

[8] Chris Groër, Bruce Golden, and Edward Wasil. 2010. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation* 2, 2 (2010), 79–101.

[9] Dietmar Jannach. 2013. Toward Automatically Learned Search Heuristics for CSP-encoded Configuration Problems - Results from an Initial Experimental Analysis. In *Proceedings of the 15th International Configuration Workshop, Vienna, Austria, August 29-30, 2013*. 9–13.

[10] Vipin Kumar. 1992. Algorithms for constraint-satisfaction problems: A survey. *AI magazine* 13, 1 (1992), 32–44.

[11] Nina Narodytska and Toby Walsh. 2007. Constraint and Variable Ordering Heuristics for Compiling Configuration Problems.. In *IJCAI*. 149–154.

[12] Manos Papagelis and Dimitris Plexousakis. 2005. Qualitative analysis of user-based and item-based prediction algorithms for recommendation agents. *Engineering Applications of Artificial Intelligence* 18, 7 (2005), 781–789.

[13] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.

[14] Norman Sadeh and Mark S. Fox. 1996. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *AI Journal* 86, 1 (1996), 1–41. https://doi.org/10.1016/0004-3702(95)00098-4

[15] Jeff J Sandvig, Bamshad Mobasher, and Robin Burke. 2007. Robustness of collaborative recommendation based on association rule mining. In *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 105–112.

[16] Sebastian Schelter and Sean Owen. 2012. Collaborative filtering with apache mahout. *Proc. of ACM RecSys Challenge* (2012).

[17] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. 2006. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*. 1473–1480.

[18] Markus Zanker. 2008. A collaborative constraint-based meta-level recommender. In *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 139–146.

[19] Markus Zanker, Markus Aschinger, and Markus Jessenitschnig. 2007. Development of a collaborative and constraint-based web configuration system for personalized bundling of products and services. In *International Conference on Web Information Systems Engineering*. Springer, 273–284.

[20] Markus Zanker, Markus Jessenitschnig, and Wolfgang Schmid. 2010. Preference reasoning with soft constraints in constraint-based recommender systems. *Constraints* 15, 4 (2010), 574–595.