POLITECNICO DI MILANO
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

POLITECNICO
MILANO 1863

A quantum approach to a Learning-based Collaborative Filtering method in Recommender Systems

Supervisor: Prof. Paolo Cremonesi
Co-Supervisor: Dott. Maurizio Ferrari Dacrema

Thesis by:
Tang-Tang Zhou, 913087

Academic Year 2019-2020

Abstract

Recommender systems are information filtering systems used to recommends movies, books, or many other products and services to users. Nowadays, due to information overload, the daily usage of recommender systems is growing. For instance, when users visit *Amazon*, they are provided with a list of recommended items based on their preferences. Moreover, recommender systems can be, generally, based on all kinds of technologies: information retrieval algorithms, statistics, machine learning algorithms, etc.

With the new era, new promising technologies are available. One of them is the quantum computing; especially quantum annealing computers. They are machines that follow natural physics processes where the system evolves towards the lowest energy. This allows them to solve specific combinatorial optimization problems.

In general, many optimization problems can be already solved with exact or heuristic algorithms executed on classical computers. But nobody tried to use quantum annealing computers to solve them in recommender systems. Therefore, the objective of this research is to exploit the new quantum technology to develop a new model. Specifically, to be solved by quantum computers, the core algorithm of the model should be composed of specific optimization problems.

In the end, the results of the experiments show that the new quantum technology has performance nowhere near state-of-the-art models. However, the new technology might have a promising future because the new model solved with a heuristic algorithm achieved performance similar to state-of-the-art models. Therefore, with improved quantum computers, it might be possible to obtain better performance.

Sommario

I sistemi di raccomandazione sono sistemi di filtraggio delle informazioni usato per raccomandare film, libri, o molti altri prodotti e servizi a utenti. Al giorno d'oggi, a causa del sovraccarico delle informazioni, l'utilizzo giornaliero dei sistemi di raccomandazioni sta aumentando. Per esempio, quando utenti visitano *Amazon*, vengono forniti con una lista di prodotti raccomandati basato sulle loro preferenze. Inoltre, i sistemi di raccomandazione possono essere, generalmente, basati su tutti i tipi di tecnologie: algoritmi di recupero delle informazioni, statistica, algoritmi di machine learning, etc.

Con la nuova era, nuove promettenti tecnologie sono disponibili. Una di queste è la computazione quantistica; particolarmente i computer di ricottura quantistica. Sono macchine che seguono processi naturali fisici dove il sistema evolve verso l'energia più bassa. Questo permette a loro di risolvere specifici problemi di ottimizzazione combinatoria.

In generale, molti problemi di ottimizzazione possono essere già risolti con algoritmi esatti o euristici eseguiti su computer classici. Ma nessuno ha provato ad usare i computer di ricottura quantistica per risolverli nei sistemi di raccomandazione. Quindi, l'obiettivo di questa ricerca è di sfruttare la nuova tecnologia quantistica per sviluppare un nuovo modello. Nello specifico, per essere risolto da un computer quantistico, il cuore dell'algoritmo del modello dovrebbe essere composto da specifici problemi di ottimizzazione.

Alla fine, i risultati degli esperimenti mostrano che la nuova tecnologia ha prestazioni ben lontani da modelli *state-of-the-art*. Tuttavia, la nuova tecnologia potrebbe avere un futuro promettente poiché il nuovo modello risolto con un algoritmo euristico ha raggiunto prestazioni simili ai modelli *state-of-the-art*. Dunque, con computer quantistici migliorati, potrebbe essere possibile ottenere prestazioni migliori.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recommender systems are information filtering systems used in Web application systems to help users by recommending products or services (e.g. books, movies, videos, etc.). With the explosive growth of information on the Web, recommender systems are developed to cope with the information overload problem on users. For instance, *Amazon* suggests a list of items to users based on their history. Among all types of recommender systems, in this research, the focus is on the most used category, called Collaborative Filtering. Specifically, this type of technique exploits only the users' past interactions on items to provide recommendations. For instance, a specific type of collaborative filtering technique uses the concept of similarity. In other words, it uses the collaborative information of these interactions to recommend either similar items based on the users' past interactions or items liked by similar users.

Furthermore, this research tackles, also, the new topic of quantum computers. Different from classical computers, these new machines represent information with a new unit, called quantum bit or "qubit". Thanks to a few special properties of the qubit, quantum computers can be in multiple states simultaneously and act on all qubit at the same time in constant time. Nowadays, a specific quantum computing technique, called quantum annealing, is the leading quantum technology. It is a natural quantum physics phenomenon where a system evolves from a general problem to a target problem while keeping the lowest energy. More specifically, quantum annealing computers are designed to solve combinatorial optimization problems. Obviously, in the literature, there exist already heuristic algorithms run on classical computers that can solve these problems.

In brief, the overall focus of this research is to merge the two fields of recommender systems and quantum computers. In particular, the objective

is to exploit quantum annealing computers in a learning-based collaborative filtering model, called Sparse Linear Model (SLIM), proposed by Ning et al.[41] in 2011. More specifically, the goal is to design a new algorithm similar to SLIM, but it requires solving specific optimization problems compatible with the quantum annealing computers. Thanks to a quantum annealing company offering remote quantum computing, called D-Wave, we were able to implement the new model. However, the actual quantum annealing technology has many issues. First, the resource in quantum computation time is limited at a point where only a few experiments can be done monthly. Second, the optimization problem that can be embedded in the quantum computer is limited in variable size; thus, the conducted experiments on the new model are focused on small datasets. Finally, the technology is affected by non-negligible noise that causes to have very sub-optimal solutions to the optimization problems. To sum it up, nowadays, the quantum annealing technology is still quite limited in its availability, scalability, and performance. However, its future is still promising since the same new model with classical heuristic solvers has promising performance w.r.t. state-of-the-art models.

Regarding the structure of the thesis, we, first, explain the current state-of-the-art and the necessary information to understand fully this research in Chapter 2. In particular, we outline the basics of recommender systems, their models and the functioning of quantum annealing computers. Then, we present the new model in Chapter 3. Specifically, we focus on its objective function, a variation of the model that helps in the size scalability issue, and various implementation details. Additionally, in Chapter 4, we fully describe all the conducted experiments executed on the new model. In particular, the experiments' goals are to check the new model's behavior, scalability in time, and performance w.r.t. other state-of-the-art models. Finally, we discuss the conclusive findings on our model, experiments and possible future works in Chapter 5.

Chapter 2

State of the art

In this chapter, the state of the art about Recommender Systems and Quantum Computing are briefly described.

First of all, in Section 2.1, we describe a few basic things about Recommender Systems. In particular, goals, the type of data, and the problem are outlined with more emphasis on a specific problem, called the top-k recommendation problem. Then, Section 2.2 describes briefly a specific type of methodology, called Collaborative Filtering focusing on a basic method based on similarity. Additionally, in Section 2.3, we expand the topic of Collaborative Filtering with the few modern advanced techniques; e.g., graph-based methods, factorization methods, and adaptive neighborhood learning methods. Finally, regarding Recommender Systems, we outline the common quality evaluation techniques and metrics in Section 2.4.

Concerning Quantum Computing, in Section 2.5, we briefly describe what quantum computing is and the type of quantum techniques with a focus on quantum annealing. Afterward, in Section 2.6, we describe one of the leading companies of quantum annealing, called D-Wave. In particular, we shortly outline its functioning and architecture. Moreover, we briefly describe D-Wave's actual challenges in its technology which are, also, general quantum annealing issues.

In the end, we briefly describe our work in Section 2.7. Specifically, we exploited D-Wave quantum annealing computers to solve specific Recommender Systems problems.

2.1 Basics of Recommender Systems

Recommender systems (RS) are information filtering technologies used to make suggestions about products or services to users [22]. They are, gen-

erally, used in Web application systems such as e-commerce, e-tourism, e-business, and e-social to help users in handling the information. For example, a popular one is the *Amazon* recommender system that proposes a list of products to buy for users based on their history [37].

With the explosive growth of information on the Web in recent years, RS grew out of information retrieval and filtering research as an individual research field to cope with the information overload problem on users [45, 37]. Of course, since it is a field highly correlated to business, the primary objective is to increase the profit of the company (i.e. owner of the Web service). Meanwhile, other common technical goals [3] are:

- **Relevance**: its goal is to recommend relevant items to the user (i.e. those that are likely to be consumed);
- **Novelty**: it refers to the aim of recommending new items that users have not seen in the past;
- **Serendipity**: this goal is to recommend items that surprise the user which might lead to a discovery of a latent interest. The benefit is that it increases sales diversity, but there is also a tradeoff with the goal of Relevance.

2.1.1 Data of a Recommender System

The recommendation problem depends on different variables. One of them is the type of data that is available [9]. In general, collected data refers to three concepts: users, items, and transactions [45].

In the first place, **users** are individuals the RS provides recommendation to. Every user has a different goal and characteristics; thus, the RS needs to suggest to users in a personalized way based on their tastes. Regarding recommendation techniques, there are some, called Demographic, that exploit user attributes to recommend users in a different way. For instance, there is a distinction of items recommended to young users w.r.t. old users [45].

About **items**, it is the general term used to denote what the system recommends to users [45]. They can be products or services with different value and complexity. For example, on *Netflix*, the movies are the objects that are recommended. Also, in this case, they are characterized by attributes that are used in some techniques, called Content-based methods.

Finally, the **interactions** are recorded transactions between user and system [45]. They are divided into explicit and implicit feedback. The former includes explicit inputs by users regarding their interest in the items.

The most popular form of explicit feedback is the rating, i.e. feedback that indicates somehow the value of an item for a user. In general, ratings are collected with a discrete categorization such as 1 to 5 stars rating. The latter are inputs gathered transparently based, for example, on user click behaviour or navigation pattern [30]. For instance, on *Amazon*, when a user buys an item the system records this action as a sort of preference towards that item.

Generally, explicit feedback or implicit feedback between users and items are collected into a rating matrix R of size $|U| \times |I|$ where U and I are respectively the set of users and the set of items. For these interactions, a family of methods, called Collaborative filtering, uses them as its unique information to find good items to recommend.

Additionally, there are also context information about the interactions such as weather of the day, time of the day, and etc. These information are exploited by other techniques [3]. Overall, based on the type of data that is available, some recommendation techniques are more appropriate than others. Regarding the interactions, almost every technique uses them.

2.1.2 Recommendation problem

An important variable to the recommendation generation process is the type of problem that has to be solved. Overall, there are two versions of the recommendation problem [3]:

- **prediction** version: the goal is to predict the rating value for all user-item combination missing in the rating matrix. In fact, this problem is also referred to as the matrix completion problem.

- **ranking** version: in many applications, it is not necessary to predict ratings in order to recommend items to users. Sometimes, it is enough to recommend a list of good items. In fact, the goal in this ranking recommendation problem is just to recommend the top-k items to each user. This problem is also called the **top-k recommendation** problem.

Based on the type of problem, the evaluation methods are different. These are discussed in more detail in Section 2.4.

Basic models

A classical taxonomy of recommendation approaches presented in [11] divides RS models into six different classes. However, in this section, the focus

is on the two most popular models: the **collaborative filtering** method and the **content-based** method.

Collaborative Filtering methods are the earliest developed and most used techniques among all personalized recommendation methods. They only require users' past ratings to recommend new items that are similar to that of another user, if both users have rated other items in a similar way [45]. In Section 2.2, more details about these methods are described.

Regarding content-based approaches, they are methods focusing primarily on items. They make use of item attributes, also called features, and ratings to recommend similar items w.r.t. users' past liked items [45]. For example, consider a movie recommender system; a user that liked a horror movie with a specific producer might like another movie with the same genre and producer.

Moreover, there are also other models which are more complex. For instance, in most real applications, hybrid methods are the most used. They are, simply, a combination of all the other recommendation techniques that exploit all kinds of data [45].

2.2 Basics of Collaborative Filtering

As discussed before, Collaborative Filtering (CF) algorithms use the collaborative information to provide recommendations [3]. Most of the CF algorithms are based on a heuristic similarity measure that indicates either item similarity or user similarity. In the former case, the similarity is used to recommend similar items to those liked in the past. Meanwhile, in the latter case, the similarity is used to recommend items liked by similar users. For the input, they only require collaborative feedback that is the ratings of the user-item interactions. These ratings are often highly correlated across various users and items; therefore, they can be used to impute the missing ratings since the past ratings provide a way to find similar tastes in users.

Among CF models, most of them focus on leveraging either item or user inter-correlations for the prediction process. Meanwhile, others use optimization techniques to train models similarly w.r.t. how a classifier or regressor is trained. In general, CF models can be divided into two main categories:

- **Neighborhood methods**: these methods predict ratings based on neighborhoods of users or items [3]. Moreover, the recommendations methods are either user-based or item-based. The former recommends items to a target user that are rated by similar users called neighbors.

While, the latter suggest, for a specific user, items that are similar to those that are rated in the past.

- **Machine Learning methods**: these techniques use mainly machine learning and data mining parameterized models in the context of predictive models [3]. They, generally, require a learning phase to find optimal parameters, and then, they can predict ratings of users quickly [14]. However, there are cases where the prediction takes much more times due to the high complexity of the model. Some examples are decision trees, rule-based models, Bayesian methods, and latent factor models [45].

2.2.1 Neighborhood item-based CF methods

The most common approach to CF is based on neighborhood models [30]. As stated before, they are divided into user-based and item-based methods. Originally, most of the systems were user-oriented where ratings are estimated using known ratings provided by the same user on similar items. Afterward, item-oriented methods prevailed because they were more interpretable. This is also due to the fact that users are familiar with items liked in the past, but do not know about those preferred by other users [30].

Both methods greatly depend on a reference algorithm, called **k Nearest Neighbors** (KNN), used to find neighborhoods for either users or items. Focusing on item-based approaches, this algorithm finds the set of k nearest items w.r.t. a target item that will contribute to the prediction of the rating of a user and the target item [45]. The contribution of each item in the set depends on a weight that is, generally, calculated with heuristic methods such as similarity measure. This measure determines the similarity between each pair of items which, overall, defines a matrix of weights, generally denoted as S [9]. Moreover, these measures require only the ratings to be computed; some examples are the cosine similarity, the Jaccard similarity, and the Pearson correlation coefficient.

Overall, the prediction function in an item-based CF with implicit feedback for each user $u \in U$ and for each target item $i \in I$ is defined as [3, 30]:

$$\hat{r}_{ui} = \frac{\sum_{j \in \text{KNN}(i)} s_{ji} \cdot r_{uj}}{\sum_{j \in \text{KNN}(i)} |s_{ji}|}, \ \forall u \in U, i \in I \tag{2.1}$$

The set KNN(i) represents the set of k closest items, computed by KNN algorithm, to target item i. While, r_{uj} represents the value of the rating matrix R in row u and column j and s_{ji} is the similarity weight measure between item j and target item i. In particular, s_{ji} is the value of the

similarity matrix S. These similarity measures are, in general, calculated by a function that returns the similarity between two items i and j. For instance, in cosine similarity, the similarity measures are calculated with:

$$s_{ji} = \frac{\sum_{k \in I} r_{jk} \cdot r_{ik}}{\sqrt{\sum_{k \in I} r_{jk}^2} \cdot \sqrt{\sum_{k \in I} r_{jk}^2}} \tag{2.2}$$

Overall, the prediction function in Equation 2.1 is a linear combination of the user's own ratings on similar items w.r.t. the target item. From a matrix perspective, the similarity measures used represent a column of S; more specifically, it is the column of the target item.

To summarize, these methods are executed in 3 core steps [14]:

1. Compute the weights of similarity matrix between items according to the rating matrix
2. Select top-k similar neighbor items for each column of the weight similarity by using the KNN algorithm
3. Predict missing ratings of the rating matrix using the formula in Equation 2.1 and generate a recommendation list for every user

For instance, a general heuristic model that uses these 3 steps is the **Item-based KNN CF** model. In particular, in the first step of this model, the weights of the similarity matrix are computed with a similarity measure, like the cosine similarity.

2.3 Advanced CF methods

Among the CF methods, there are also more advanced techniques that do not follows the general neighborhood method.

First of all, in general, CF methods present two main drawbacks: **popularity bias** and **sensitivity to sparse data** [45]. The former means that these methods tends to highly suggest items that are popular; this means that unpopular items are almost never recommended. This issue is more prominent towards items that have no interaction or a few of them. The latter is about the fact that the accuracy of neighborhood-based methods suffers from the lack of available ratings, i.e. sparsity, which is a common problem to most applications of RS.

These advanced CF methods try to overcome in some way these disadvantages by having robustness to outliers, ability to capture high-level patterns in the data, better generalization and so on [45]. However, some of

these methods still have the previous explained drawbacks. In the following section, three categories of advanced CF methods are presented: graph-based methods, factorization methods and adaptive neighborhood learning methods.

- **graph-based methods**: these techniques determine the items to be recommended by traversing a graph in a specific way. In general, it is a bipartite graph where the two sets of nodes represent respectively the users and the items.

- **factorization methods**: they projects users and items into a reduced latent space that tries to capture high level attributes of users and items. There are two ways in which the factorization is applied: factorization of a sparse similarity matrix or factorization of a user-item rating matrix.

- **adaptive neighborhood learning methods**: they determine the neighborhood directly from data using predefined similarity measures. For instance, SLIM, described in Section 2.3.3, uses techniques related to linear regression where the similarity weights are learned automatically from the data with a learning process.

2.3.1 Graph-based methods

As stated before, in common graph-based methods, data are represented in the form of a bipartite graph where the two sets of nodes are the users and the items [45]. The rating of a user u over an item i is represented with an edge connecting the user u and item i. Moreover, an edge can also have other properties such as the weight that, generally, represents the rating value assigned by the user to the item.

The fundamental process, in graph-based methods, is how the rating prediction is computed. A common method, called $\mathbf{P}^3_\alpha$ uses the concept of **random walk** where an hypothetical walker traverses the graph starting from a node and jumps randomly to other neighbor nodes and so on [42]. Formally, the random walk can also be described with a Markov process defined by a set of n states and a $n \times n$ transition probability matrix P [45].

In P^3_α[42], the rating prediction of an item i of a specific user u is seen as the probability of reaching that item i through 3 jumps starting from the user u. Therefore, the predicted rating matrix is computed as follows:

$$\hat{R} = P \cdot P^T \cdot P \tag{2.3}$$

where P is the transition probability matrix of the bipartite graph (V, E) where the users and items represent the nodes, and the user-item interactions represent the edges. In particular, each probability of P from node i to j is defined as the α-exponential of the normalized weight of the edge of the two nodes:

$$P_{ij} = \left(\frac{e_{ij}}{\sum_{k \in V} e_{ik}} \right)^{\alpha} \tag{2.4}$$

where the notation e_{ij} represents the weight of the edge that goes from i to j and α is a hyper-parameter that might increase or decrease the probability.

$\mathbf{RP}^3_\beta$

RP^3_β[42] is graph-based method based on P^3_α. Specifically, due to the strong influence of item popularity on the transition probability matrix P, RP^3_β tries to make recommendations less popularity-driven. This is done by introducing a re-ranking procedure based on item popularity.

From the predicted rating matrix of P^3_α, a further step is applied as follows:

$$\hat{r}^*_{ui} = \frac{\hat{r}_{ui}}{D_i^\beta} \tag{2.5}$$

where D_i is the in-degree, i.e. the number of incoming edges of the node related to item i, and β is a strictly positive hyper-parameter value that controls the effect of the re-ranking process (e.g., $\beta = 0$ indicates that there is no re-ranking; thus equivalent to P^3_α). In case two items i and j with the same transition probabilities from a user u and $D_i < D_j$ (i.e. item j more popular than item i), then this model re-rank item i higher than j.

2.3.2 Matrix Factorization

As explained before, among the advanced CF methods there are factorization methods. One of these methods is called **Matrix Factorization** (MF) that focus on factorizing the rating matrix into the products of two smaller matrices. In particular, in CF, the two matrices represent latent factors (i.e. features of the reduced latent space) of the users and the items; in this way, the model tries to explain the rating matrix with synthetic features of users and items [35].

Let k be the number of latent factor, then the matrix factorization model tries to create two matrices, **user-factor matrix X** of size $|U| \times k$ and **item-factor matrix Y** of size $|I| \times k$, in a way such that $R \approx XY^T$ [3]. Every rows of U and V represents respectively a user and item in the latent factors. While every column of U and V represents a latent vector.

Given these two matrices, U and V, the rating prediction problem is trivial as follows:

$$\hat{r}_{ui} = X_u \cdot Y_i^T = \sum_{j \in I} X_{uj} \cdot Y_{ij} \tag{2.6}$$

where X_u and Y_i represent respectively the u-th row vector of matrix X and i-th row vector of matrix Y; while X_{uj} and Y_{ij} is respectively the value at row u and column j of matrix X and the value at row i and column j of matrix Y. However, the major challenge of matrix factorization models is the computation of the mapping of each user and item to the latent vector X_u and Y_i [35].

Matrix Factorization with BPR

Many ways to compute the mapping has been proposed in RS; a popular computation method is to learn the latent vectors of X and Y through a gradient-descent algorithm. In implicit feedback and top-k recommendation task problem, a general way to solve this problem is to use **Matrix Factorization with Bayesian Personalized Ranking** (MF BPR) [44].

This model uses the **BPR learning algorithm**, a stochastic gradient-descent algorithm based on bootstrapping samples from training triples. In particular, a triple is composed by a user, a "positive" item for that user, and a "negative" item for that user. More specifically, in implicit feedback, the "positivity" of an item is when the item is liked by the specific user, while the "negativity" is when it is disliked. Thus, based on these samples, the model learns the weights of the matrices X and Y by following the inverse direction of the gradient of the BPR loss, i.e. the general optimization criterion for personalized ranking.

2.3.3 SLIM

Sparse LInear Models (SLIM[41]) is a family of models belonging to the adaptive neighborhood learning-based category of item CF approaches. It is a model that encourages sparsity in the similarity weights with the use of regularization methods [3]. This approach is, especially, designed to work with non-negative ratings where users have no mechanism to specify dislikes. Hence, it is most appropriate for implicit feedback matrices. Unlike the method of regression stated previously, SLIM does not restrict the regression coefficients to only the neighborhood. Therefore, the prediction function is defined as follows [3, 45]:

$$\hat{r}_{ui} = \sum_{j \in I} w_{ji} \cdot r_{uj}, \ \forall i \in I \tag{2.7}$$

It is important to exclude the target item from the summation. Otherwise, the problem becomes trivial since it becomes possible to predict the rating using the original rating itself. This can be excluded by setting either $s_{ii} = 0$ or $r_{ui} = 0$. Meanwhile, In the matrix notation, the prediction problem is defined as:

$$\hat{R} = R \cdot W \text{ with Diagonal}(W) = 0 \tag{2.8}$$

Finally, the main goal of SLIM is to minimize the Frobenius norm $||R - RW||^2$ along with regularization terms. Moreover, this entire problem can be resolved by minimizing small independent optimization problems for each target item i. Each of them is defined as: [3, 45]:

$$\begin{aligned} &\min \sum_{u \in U} \left(r_{ui} - \sum_{j \in I} s_{ji} \cdot r_{uj} \right)^2 + \alpha \cdot \sum_{j \in I} s_{ji}^2 + \beta \cdot \sum_{j \in I} |s_{ji}| \\ &\text{subject to :} \\ &\quad s_{ji} \geq 0 \;\; \forall j \in I \\ &\quad s_{ii} = 0 \end{aligned} \tag{2.9}$$

This is possible because the similarity weights to be learned are not overlapping between different item formulations. In addition, to create a more interpretable solution, the weights are constrained to be non-negative. Overall, this final formula is called, in general, **SLIM Elastic Net** because the regularization used are L_1 and L_2 norm (i.e. Lasso and Ridge). These sparsity regulators ensure that each predicted rating can be expressed as a linear combination of ratings of a small number of related items. For practicality, sometimes, SLIM is implemented with the restriction of a neighborhood in the prediction function to add another layer of sparsity regularization.

2.4 Evaluation of Recommender Systems

Recommender systems, similarly to other research fields such as machine learning, require the evaluation of the quality of the system at different stages of its life cycle [45]. For CF, especially, the evaluation shares many similarities with the one of classification since CF can be seen as a generalization of the classification and regression modeling problem [3]. However, there are still many differences. One of the main differences is the experimental setting, i.e. how the evaluation experiments are executed. In RS, there are 3 types of experimental settings: offline experiments, online experiments, and user studies.

Offline experiments are the easiest to conduct because they do not require interaction with real users. They are, generally, evaluation of the system using metrics on public datasets [45].

Instead, **user studies** and **online experiments** require interactions with real users. The main difference between them is the fact that user studies ask a small group of users to perform tasks on the RS in a controlled environment, while online experiments are, generally, executed on a bigger pool of real users, unaware of the experiment [45]. In general, these two experiments are more trustworthy and closer to reality than offline experiments, but during the design phase of an algorithm, it is not always possible to apply them. For instance, A/B testing is an online experiment where users are divided into two groups, A and B, and for each group, a different RS is used [3]; at the end of the test, the quality of the two groups are compared.

Overall, in this section, the focus is on the partitioning of the data and the metrics of offline experiments. In particular, regarding the metrics, the accuracy metrics, which are those related to how good a recommendation is, are described in this section. Then, the focus is on the beyond-accuracy metrics. These metrics measure the quality of the recommendations on secondary goals such as coverage, diversity and item popularity.

2.4.1 Offline experiments

Offline experiments are performed by using pre-collected datasets of user-item ratings [45]. In CF, the only required data to run this experiment is the rating matrix. Assuming that the user behavior of pre-collected data will be similar enough to the user behavior when RS is deployed, the procedure of this experiment is to try to simulate the behavior of users that interact with the RS in an offline situation without the interaction with real users. This is why offline experiments are attractive. Moreover, it allows comparing a wide number of candidate algorithms at a low cost. But, the downside is that it can answer only a very narrow set of questions (e.g. accuracy of the predictions).

However, there are some design issues to consider. In offline experiments, it is crucial to avoid overestimating or underestimating the quality of RS [3]. A common approach, applied even in ML, is to split the data into two parts: training set, and test set. The training set is the only part that is visible by the RS, while the test set is hidden and is used for evaluating the system with metrics. This is what happens when the goal is to evaluate a single set of hyper-parameters of an algorithm, but when the objective is to optimize

the hyper-parameters of an algorithm, it is necessary to divide the data into three parts [3]:

- **training** set: this portion of data, which is generally bigger than the other two, is exclusively used for the training of the model;
- **validation** set: this set is used for parameter tuning, i.e. to optimize the hyper-parameters of a model. After this evaluation, the "best" model with tuned parameters is selected;
- **test** set: after the optimization of the hyper-parameters, the test set is used to evaluate the final model. This set must not be leaked to the other previous sets; otherwise, there will be an overestimation of the true accuracy metric. This step is necessary because the tuned model might be overfitting the validation set.

The difference w.r.t ML is in how the data is sampled. Rating matrices are, typically sampled in an entry-wise fashion, i.e. a subset of entries are used for training and the remaining for evaluation. For segmenting rating matrices, there are two common division methods [3]:

- **hold-out**: it divides the rating matrix in multiple parts by random sampling in an entry-wise fashion. In case the final goal is to obtain three parts, then this method hides two fractions of entries of the rating matrix as test set and validation set, and uses the remaining part as the training set. The evaluation of metrics, such as accuracy, is done as explained previously.
- **cross-validation**: it divides the rating matrix into k sets of equal size by random sampling in an entry-wise fashion. For each single k-set, the system is trained on the $k-1$ remaining sets and tested on the single k-set. This yields to k metrics results that are reported with their average value. Moreover, this method estimates more accurate evaluation metrics when k is large. However, in case the problem requires three parts of data, a hold-out division method is applied to divide the dataset into two parts in which one of them is the test set, while the other one is divided with cross-validation. An extreme case is when k is chosen as equal to the size of the number of observed entries in the rating matrix, which is called leave-one-out cross-validation.

Notice that these two are not the only methods, but there are others that try to imitate better the user behavior in a real online RS.

2.4.2 Accuracy metrics of offline experiments

Similar to any ML algorithm, RS is evaluated with metrics on a set of data that are hidden from the training process, e.g. validation set and test set [3].

The choice of accuracy metrics depends on the type of recommendation problem. In general, in the case of the prediction problem, the metrics used measure how well the predicted ratings compare with the hidden ratings in the test set; these are, generally, prediction metrics. While in the top-k recommendation problem, the most popular metrics measure how relevant the list of items is for the user by comparing it with the hidden rated items in the test set [3]. Overall, accuracy metrics can be broadly classified into three categories: prediction accuracy metrics, precision metrics, and ranking accuracy matrics [28].

Prediction accuracy metrics

Prediction accuracy metrics measure the accuracy of the rating prediction through error metrics [3]. For instance, the most popular error metrics, even in the ML field, is the mean squared error (MSE). However, in the top-k recommendation problem where the focus is on the top-k items, these metrics provide a less realistic perspective of the true usefulness of the RS because, in general, users only view few items from a list of k items. For this reason, the focus in this section is more on the classification and ranking accuracy.

Precision metrics

Precision metrics measure the frequency with which a RS suggests relevant items [28]. For instance, the two most popular metrics for information retrieval systems are **precision** and **recall**. In a top-k recommendation problem, the precision and recall metrics depend on the length of the list of items; thus they are referred to as $precision@k$ and $recall@k$. For each user u, by considering the suggested list of k items as $T_u@k$ and the true set of relevant items as G_u, the precision and recall are defined as [3]:

- $precision@k$: it is defined as the percentage of recommended items that are truly relevant over all the recommended items

$$Precision@k = \frac{|T_u@k \cap G_u|}{|T_u@k|} \tag{2.10}$$

- *recall@k*: it is defined as the percentage of ground-truth positives that have been recommended as positive for a list of size k

$$Recall@k = \frac{|T_u@k \cap G_u|}{|G_u|} \tag{2.11}$$

In the end, the precision and recall are reported as the arithmetic mean over all test users. When using classification accuracy metrics, one cannot simply focus on either precision or recall because there is a trade-off between them. For instance, a longer recommendation list typically improves recall but reduces precision. Therefore, both need to be considered at the same time, e.g. by computing precision-recall curves. Moreover, there are also metrics that summarize these curves, such as F1-score and Area Under the Curve (AUC) [45].

Ranking accuracy metrics

Regarding ranking accuracy metrics, these focus on the accuracy of only the ranks of the top-k items rather than all the items [3]. It is clearly meant for the top-k recommendation problem. In general, ranking accuracy metrics can be divided into two approaches:

- **reference ranking**: the goal is to try to find the correct order of a set of items for each user and measure how close it is to the correct one (i.e. reference) [45]. Obviously, a reference ranking for each user is needed. In general, the items can be ranked in decreasing order of ratings. The most used reference ranking accuracy metric is the Mean Average Precision (**MAP**[6]), which comes from information retrieval. It computes the overall precision of the system at different lengths of the recommendation list [40]. Moreover, it is defined as the mean of the Average Precision (AP[15]) values over all the users in the test set.

$$MAP@k = \frac{1}{|U|} \sum_{u \in U} AP_u@k \tag{2.12}$$

$$AP@k = \frac{\sum_{i=1}^{k} Precision@i \cdot Rel@i}{Number\ of\ relevant\ items} \tag{2.13}$$

  where k is the length of the recommended list of items, U is the set of all users in the test set and $Rel@i$ is a binary number that indicates if the item in position i is relevant.

- **utility-based ranking**: this approach uses the ground truth rating in combination with the RS ranking [3]. The goal is to try to quantify how useful the user might find its recommended ranking based on

the order of the recommended items. A popular utility-based ranking accuracy metric is the Normalized Discounted Cumulative Gain (**NDCG**), which is a measure from information retrieval where positions are discounted logarithmically [45]. Assuming each user $u \in U$ has a "gain" $g_{u,i}$ from being recommended item i, then, the average DCG for a top-k recommendation task is defined as:

$$DCG@k = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{g_{u,i_j}}{\log_b(j+1)} \tag{2.14}$$

where i_j is the item in position j of the list of recommendations and b is a parameter for the base of the logarithm (typically, $b \in [2, 10]$). In general, with implicit feedback, the gain $g_{u,i}$ is a binary value which indicates if the item i is relevant for the user u. Finally, NDCG is defined as the normalized version of DCG:

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{2.15}$$

$$IDCG@k = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{1}{\log_b(j+1)} \tag{2.16}$$

where IDCG is the ideal DCG.

2.4.3 Beyond-accuracy metrics of offline experiments

As stated in Section 2.1, accuracy is not the only goal in building a RS. For example, recent research works suggests that having more diverse recommendations in the list of recommended items at the cost of accuracy can be beneficial [43].

The following section describes some of the beyond-accuracy metrics such as item coverage, diversity and average item popularity.

Item coverage

The item coverage of a RS measures, in general, how many items are covered in the recommendations [28]. In many systems, especially those that use a CF approach, they may provide good recommendations but their item coverage measures are very small [45]. More specifically, most recommended items are popular items, i.e. those selected by most of the users. A common concept of item coverage is defined as the fraction of items that are recommended to at least one user; this is referred to as the **catalog coverage**

[26, 3]. This coverage notion is specifically suited to recommendation lists and is formulated as follows:

$$CC@k = \frac{|\cup_{u \in U} T_u@k|}{n} \tag{2.17}$$

Notice that n is the number of items and $T_u@k$ is the list of top-k items recommended to user u. The catalog coverage is somehow a trade-off; Most of the times, the higher is the coverage, the lower is the accuracy metric. This happens because, in the definition, each item is counted even if the recommendation of the item is irrelevant. As a solution, it is possible to apply a correction which counts only relevant items:

$$CC_{correct}@k = \frac{|\cup_{u \in U} TC_u@k|}{n} \tag{2.18}$$

where $TC_u@k$ is the list of relevant recommended items to user u.

Diversity

Diversity is seen as the opposite of similarity. As stated in the goals of RS, increasing diversity might increase the chance to find an item relevant to a user. The definition of diversity is not standard; there are a lot of different definitions and evaluations of diversity [36]. Among them, an easily interpretable definition is the **Mean Inter-List Diversity** [48]. This diversity measures the uniqueness of different user recommendation lists [21]. Specifically, it measures how "diversified" the recommendations of different users are in a top-k recommendation problem:

$$MIL@k = \frac{1}{|U|^2 - |U|} \sum_{u \in U, v \in U, u \neq v} 1 - \frac{q(u,v)@k}{k} \tag{2.19}$$

where $q(u,v)@k$ is the number of common items between the recommendation list of u and v. Notice that the length of the recommendation lists of the two users has to be k. This is an equivalent formulation proved in [23].

Average item popularity

Average item popularity measures how "popular" the list of item recommendations to all users is. It is quite correlated with the item coverage since, in general, low item coverage and high accuracy indicates a system that recommends most of the time popular items. Hence, the item popularity may be high in this situation. The **Average Item Popularity** is defined as:

$$AIP@k = \frac{1}{|U|} \sum_{u \in U} \frac{\sum_{j=1}^{k} p(i_{u,j})}{k} \tag{2.20}$$

where $p(i_{u,j})$ is the $[0,1]$-normalized popularity value of the item in position j of the recommendation list $T_u@k$. Notice that the popularity value of an item is the number of times an item has been rated or selected by a user up to now [2].

2.5 Quantum computing

Quantum computers, originally proposed by Paul Benioff[7] and Richard Feynman[25] in the early 1980s, are special computing machines that exploit quantum physics phenomenal to perform quantum computations. [38, 4]. Unlike a classical computer that operates with electric signals to represent information as bits, quantum computers operate with a quantum bit called "**qubit**" [32] represented by quantum elements. For instance, trapped ions, atoms or electrons are examples of quantum elements that are exploited to represent a qubit. Besides, there are other fundamental differences between them.

A different property of qubit w.r.t. bit is the **superposition**, which means that it can be in both states (i.e. 0 and 1) simultaneously with a certain probability [38]. A key fact is that superposition states cannot be directly observed, but when a qubit is measured, it "collapses" to a classical state (i.e. 0 or 1) with a certain probability. Another fundamental property is the fact that qubits can interact between them in a way that they are no more considered independent [38]. This property is called **entanglement**. In other words, if two qubits are entangled together, no matter how far they are put apart, they are still coupled together in a dependent relationship. Therefore, by measuring one of the entangled qubits it is possible to deduce the other one. This is because a variation on the state of one of the entangled qubits also influence the other one. In addition, qubits are very fragile. In fact, they present a high error in the reading operation of a qubit caused, generally, by electromagnetic interference. Meanwhile, in classical computers, bits of electric signals are much more robust against noise.

Moreover, quantum computers provide an advantage in terms of time, called quantum speedup. This is because they are able to act on all qubits in constant time thanks to the special characteristics of the qubit [38]. Indeed, one of the most famous applications/examples is the Shor's algorithm [46] which demonstrates the power of quantum computers by factorizing an n-bit in polynomial time on a specific quantum computer, called gate model [1]. Meanwhile, on classical computers, there is no actual algorithm that can factor numbers in polynomial time.

Nowadays, regarding the different types of quantum computers, there are mainly two leading quantum computing technologies [17]:

- **Gate-model**: this type of quantum computing is universal. In the sense that it is able to solve any kind of problem. In particular, it is developed around the concept of a register of n qubits that is controlled by in-parallel or in-series quantum circuits. Each circuit is composed of different quantum gates which are similar to the logic gates of a classical computer [38]. Specifically, similarly to electronic circuits, an algorithm on gate-model is modeled by building a sort of circuit that works on the register of qubits. For example, in a parallel level, qubits can be entangled, while in series level, qubits are altered by quantum gates or read by a reading component.

- **Quantum annealing**: it is a natural quantum-mechanical evolution where quantum systems evolves naturally from a general problem to a target problem. Unlike the gate-model, this method works specifically for combinatorial optimization problems. In addition, it is commonly confused by the metaheuristic algorithm implemented by Apolloni et al. in 1988[5], also called quantum annealing. Specifically, the algorithm models particle fluctuations in quantum dynamics, while the quantum annealing, executed on quantum computers, is a natural phenomenon.

The following sections focus primarily on quantum annealing and its implementation and formulation.

2.5.1 Quantum annealing

Quantum Annealing (QA) is a natural quantum physics process where a system evolves naturally from a general problem to a target problem. In particular, the system evolves naturally while keeping the ground state, i.e. lowest energy. As stated previously, quantum annealing is commonly confused with a metaheuristic algorithm for combinatorial optimization problems that mimics the the natural quantum process in quantum physics [38]. For clarity, in the following sections, the metaheuristic algorithm is referred as classical QA (CQA) while the natural phenomenon as QA.

Additionally, QA can be also seen as a variation of another classical metaheuristic algorithm, called Simulated Annealing (SA [34]). Differently from the CQA, SA mimics the thermal annealing process used in metallurgy. Overall, both CQA and SA mimics natural physical processes to solve combinatorial optimization problems. Moreover, both are optimization methods

that are based on heuristic local search that continuously iterates from a solution to another in order to head towards lower-cost solutions[38, 4].

Annealing problem formulation

The physical system used in QA can be described by a **Hamiltonian** $\mathcal{H}(t)$, a mathematical description of a system in terms of energies [17]. A generic Hamiltonian of QA is a function that maps certain states to energies defined as follows:

$$\mathcal{H}(t) = s(t)\mathcal{H}_I + (1 - s(t))\mathcal{H}_F \tag{2.21}$$

where $\mathcal{H}_I$ is the initial Hamiltonian from which the system starts its annealing process, $\mathcal{H}_F$ is the final Hamiltonian, also called problem Hamiltonian, which is the goal of the problem to be solved, and $s(t)$ is an adiabatic evolution path function that decreases from 1 to 0 as $t : 0 \rightarrow t_f$, for some elapsed time t_f [38]. In theory, the system starts from the ground state of the initial Hamiltonian that is easy to reach, then the annealing process starts by introducing little by little the problem Hamiltonian depending on the evolution path $s(t)$. In the end, when the process is finished, the system reaches the ground state of the problem Hamiltonian if the process was slow enough. Otherwise, it might end up in an excited state which does not have minimum energy. The speed of the process highly depends on the problem, more particularly, in its energy spectrum. For example, problems with a small energy gap between the ground state and the first excited state are more difficult to cope with.

In QA, assuming to have n qubits connected by a graph $G = (V, E)$, then a classical way to define the problem Hamiltonian is

$$\mathcal{H}_F = \sum_{i \in V} h_i \sigma_i^z + \sum_{(i,j) \in E} J_{ij} \sigma_i^z \sigma_j^z \tag{2.22}$$

where the notation σ_i^z is the Pauli-z operator applied on the qubit i, while h_i and J_{ij} are qubit biases and couplers to apply respectively biases on the superposition "collapse" probabilities and entanglement between qubits [38]. More specifically, qubit biases modify the probabilities of a qubit to "collapse" at a certain classical state and qubit couplers entangle a qubit to another which removes the independency property [17].

The problem Hamiltonian is very similar to the **Ising model**, a mathematical model used for studying phase transitions of physical systems [38]. Given n particles arranged on connected graph $G = (V, E)$ where each particle can have either positive or negative spin s, then the energy $H(s)$ of the

Ising model is defined as:

$$H(s) = \sum_{i \in V} h_i s_i + \sum_{i,j \in E, i<j} J_{ij} s_i s_j \tag{2.23}$$

where s_i is a spin variable of particle i that can assume as values either $+1$ or -1. Meanwhile, in Ising formulation, h_i is the strength of the applied field at particle i and J_{ij} acts as interaction field between neighboring spins i and j [8]. These parameters h_i and J_{ij} are equivalent to those in the problem Hamiltonian. Overall, this is why the previously defined problem Hamiltonian is a classical implementation in QA.

QUBO

A more popular and equivalent formulation to the Ising model is the Quadratic Unconstrained Binary Optimization (**QUBO**). Given an $n \times n$ matrix of weights Q, the QUBO problem is to minimize the following quadratic function based on n binary variables:

$$\min_x x^T Q x \tag{2.24}$$

where $x = [x_1, x_2, ..., x_n]^T$ is a column vector containing binary variables [38]. In index notation, it is clear that the QUBO problem is equivalent to the Ising fomulation:

$$\min \sum_{i,j} Q_{ij} x_i x_j = \min \sum_i Q_{ii} x_i^2 + \sum_{i \neq j} Q_{ij} x_i x_j \tag{2.25}$$

where the diagonal of the matrix Q reflects the parameters h_i of the Ising model and the other values are related to the parameters J_{ij}. In particular, Ising problems can be transformed into QUBO with a simple substitution $s = 2x - 1$. Due to these characteristics, there are two "standard" forms: symmetric matrix and upper triangular matrix [27]. Both cases are equivalent w.r.t. the problem to be optimized. In the field of combinatorial optimization, QUBO problems describe many important problems. For instance, SAT, Knapsack, and Map Coloring are problems that can be reformulated into a QUBO. These problems are **NP-hard** problems, which means that QUBO belongs to this category [27].

QUBO is the ground foundation of quantum annealing because it can describe NP-hard problems and it is equivalent to an Ising problem [27]. For this reason, in the quantum annealing field, it is very common to formulate problems in QUBO and then, embed them in the specific architecture structure of the quantum computer.

In case problems are constrained, there is a general approach that uses penalty models to handle these problems [27]. For equality constraints $Ax = b$, they can be introduced into the QUBO models by adding a penalty to the objective function:

$$P \cdot (Ax - b)^T \cdot (Ax - b) \tag{2.26}$$

where P is a penalty constant value that depends on the original optimization problem. In general, the penalty value should be large enough such that low energy solutions do not violate this constraint [16]. In the case of inequalities, it is possible to transform it first into an equality constraint by introducing slack variables, but it is a little different w.r.t. traditional transformation of the optimization research field.

In addition to the penalty model, there are many other ways to obtain a QUBO formulation. For instance, problems can be described with Constraint Satisfaction Problem (CSP) and then, transformed into QUBO with a specific procedure. Another equivalent formulation is **graph formulation** where weights on edges are considered as qubit couplers and weights on vertices are considered as qubit biases.

Simulated Annealing (SA)

As explained previously, SA is a metaheuristic algorithm that mimics the thermal annealing process to solve combinatorial optimization problems. This algorithm, proposed by Kirkpatrick et al. [34], is based on applying a heuristic search to find the minimum of an objective function.

More specifically, it uses an iterative strategy from an initial state x having cost c [38]. At each iteration, it selects a random neighbor of the current state x; thus, if the neighbor state has lower cost, then it becomes the next state. Otherwise, in case of higher cost, it can become the next state based on a probability defined as: that depends on a value, called **temperature** T, and on the difference between the new cost $cnew$ and the old cost c:

$$P = \min\left(1, e^{-\frac{cnew-c}{T}}\right) \tag{2.27}$$

where e is the base of the natural logarithm and $cnew$ is the cost of the neighbor. Meanwhile, the temperature T follows a cooling schedule, similar to the cooling of metals in metallurgy. In particular, it starts from a specific high temperature and decreases by following its schedule.

Regarding the problems that SA can solve, the QUBO problem is also one of them. As stated before, SA is not an exact algorithm, which means that it is not guaranteed to find the optimal solution.

2.5.2 Applications of quantum annealing

Universal quantum computing can solve every Turing-machine problem, but in QA, the focus is on optimization problems [38]. Nowadays, quantum ML is one of the most promising areas from the aspect of quantum computing [29]. Most algorithms of ML are optimization problems that are ideal for QA. Overall, there are 2 general categories of applications in QA: optimization problems and sampling problems. In the former case, they can be solved by converting them into energy minimization problems, e.g. QUBO and, then solved them with the annealing process of the Hamiltonian. For the latter problem, they are useful for ML problems where the goal is to build a probabilistic model of reality [17]. This works well with QA because the sampling from qubits in superposition is a natural way to obtain values in a probabilistic way.

Concerning optimization problems, there are many which are not about ML. For example, some classical optimization problems are Minimum Vertex cover, Set Partitioning, Graph Coloring, SAT, etc. These are all problems that can be solved with QA. In the business field, QA is already helpful. In fact, there are already researches done by big companies. For example, Toyota developed a method for controlling traffic signals [31]. Obviously, also in quantum ML, there are researches on generated datasets. For instance, in [20], there is a study on quantum linear regression by changing the number of data point or the number of parameters of the model.

2.6 D-Wave

Nowadays, one of the leading quantum annealing companies is D-Wave which, in 2011, started the first quantum computer with 128 qubits. Then, almost every two years, D-Wave doubled its qubits until reaching more than five thousand qubits in 2020. Unlike gate-model quantum computers that still have few qubits, D-wave machines' qubits can reach such a high number of qubits because there is a specific architecture in which qubits are connected.

From the user's point of view, D-Wave machines are composed of two components: the quantum processing unit (QPU) and the front-end server. The quantum processor chip inside the QPU works in an environment with a temperature below 20mK and it is shielded from electromagnetic interferences to keep the system as closed as possible [38]. Moreover, since it is a quantum annealing computer, the phenomenon can be described by a

Hamiltonian. In D-Wave, it is defined as:

$$\mathcal{H}(t) = -\frac{A(s)}{2}\left(\sum_i \sigma_i^x\right) + \frac{B(s)}{2}\left(\sum_i h_i\sigma_i^z + \sum_{i,j} J_{ij}\sigma_i^z\sigma_j^z\right) \tag{2.28}$$

where σ_i^x is the Pauli-x operator and, $A(s)$ and $B(s)$ are a pair of envelope functions that define the annealing path [17, 38]. Moreover, the annealing path of $A(s)$ and $B(s)$ depends on $s(t) : 0 \to 1$ as $t : 0 \to t_f$ for some elapsed time t_f. The D-Wave problem Hamiltonian is the same as the classical implementation of QA defined before. Therefore, it is possible to use the formulation of Ising model or QUBO as objective functions.

The second component of D-Wave computers is the front-end server with which users can interact to send problems to be solved remotely by quantum computers [38]. Once the problems are sent, they need to wait in a queue until a quantum computer becomes available. Nowadays, the front-end server offers two types of computation:

- **Quantum computing**: it is a type of computation that only consists of using the QPU. D-Wave offers two machines: D-Wave2000Q[17] and Advantage system[39]. The former is the older one with 2048 qubits allocated on an architecture, called Chimera. Meanwhile, the latter one is the most recent, introduced publicly in 2020, with more than five thousand qubits on a different architecture, named Pegasus. Moreover, when using these quantum computers, there is flexibility in the evolution path $s(t)$. In particular, it let users change how the annealing process evolves. For instance, the standard evolution process is a linear evolution that goes from 0 to 1 as $t_f : 0 \to t_f$.

- **Hybrid computing**: it consists of a merge of classical computation and quantum computation. The most recent one, described in [13], can handle up to one million variables with some restrictions on the connectivity of the variables between each other. The algorithm used by the hybrid system is not disclosed, but only an abstract structure and process is described in [13]. In particular, the hybrid system is composed by a portfolio server used as interface with the user. Then, when a problem is received, it combines classical solvers and quantum solvers to solve it.

2.6.1 Architecture

As stated before, D-Wave allocates its qubits on a specific architecture. Nowadays, there are two available architectures: **Chimera** and **Pegasus**.

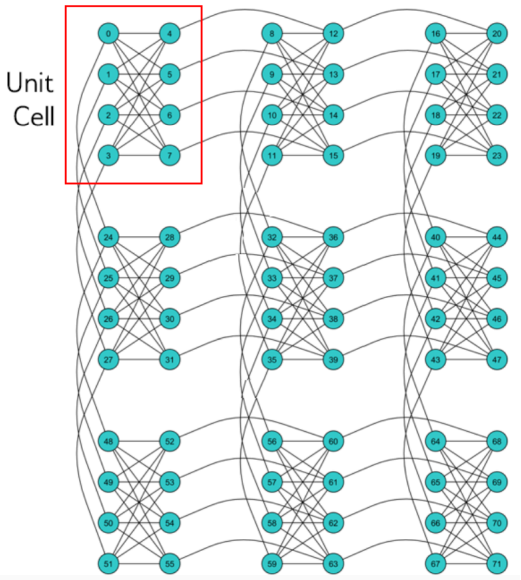

Figure 2.1: Chimera graph architecture C_3. One bipartite graph of 4 node per side is a unit cell. The connection inside the unit cell are "internal" couplers, while the others are "external" couplers (Source: D-Wave Systems Inc.)

The most recent one is Pegasus used by Advantage system with much more connectivity and qubits. These two architectures differentiate between each other by their qubits structure.

The Chimera architecture, denoted by C_N, consists of a graph formatted with a grid of $N \times N$ unit cells [17]. Each unit cell is a connected bipartite graph of 8 nodes, 4 on the left side and 4 on the right side as shown in Figure 2.1. Moreover, each node of the unit cell is connected with another node of an adjacent unit cell. In particular, the right side nodes of a unit cell are connected to the right side nodes of the right adjacent unit cell, while the left-side nodes are connected to the lower adjacent unit cell. Overall, each node of the Chimera graph has a degree of 6 [33]. Regarding D-Wave2000Q, it uses a C_{16} Chimera architecture which has 16×16 unit cells totaling 2048 qubits.

With the recent Pegasus architecture, the degree of each node in the graph becomes 15 [33]. Similar to Chimera, Pegasus architecture notation is denoted by P_N. In Chimera, qubits are connected with 2 types of couplers: internal and external. The internal couplers are the connection inside the unit cell, while the external ones are the connection between different unit cells. Unlike Chimera, Pegasus has a third type of coupler, called odd couplers. These connect parallel qubit pairs in adjacent rows or columns. The

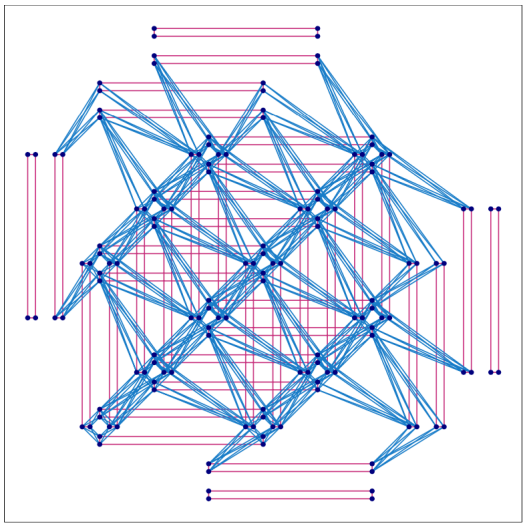

Figure 2.2: Pegasus graph architecture P_3. Blue lines are "internal" couplers, long red lines are "external" couplers, and short red lines are "odd" couplers (Source: D-Wave Systems Inc.)

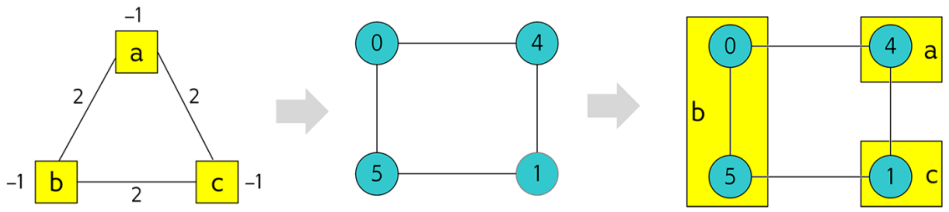

Figure 2.3: Example of the embedding of a triangular graph into a section of a Chimera unit cell by using a chain. (Source: D-Wave Systems Inc.)

structure of Pegasus is quite complex, similar to a 3D extension of blocks of qubits as shown in Figure 2.2. Specifically, a P_N graph contains $24N(N-1)$ qubits. For instance, Advantage system's architecture is P_{16} which contains 5760 qubits [33].

In the end, Pegasus has been introduced because it offers benefits against Chimera. One of them is the higher degree of graph connectivity which allows bigger problems to be solved. Another benefit is the fact that the new type of coupler helps in error correction schemes [33].

2.6.2 Minor embedding

Since D-Wave machines' architectures utilize a graph architecture, QUBO problems have to be embedded on the graph in order to be solved. Moreover,

due to the limitation in degrees of connectivity of D-Wave's architectures, logical qubits of QUBO problems, i.e. the variables, cannot be mapped in a one-to-one correspondence way with the physical qubits. For these reasons, each logical qubit of the objective function is mapped to one or more physical qubits [17]. Overall, this mapping process is called **Minor Embedding**. In particular, the process creates groups of **chains**, i.e. physical qubits representing the same logical qubit. The qubits in the same chain are entangled together in a way that forces the qubits to assume the same classical state at the end of the annealing. Then, to preserve the same graph of the objective function, every chain, considered as a single logical qubit, is connected to other chains as defined in the objective function [47]. It is necessary that qubits of different chains are not in both chains. For example, as shown in Figure 2.3, a triangular graph is embedded into a section of a Chimera unit cell by using a chain; in particular, the node 0 and 5 are in a chain which represents the single logical qubit b. Additionally, the entanglement of qubits of a chain is exactly a equality constraint in the QUBO [16]. Therefore, it is important to choose a "good" penalty value to avoid breaking the entanglement.

The process of minor embedding presents an issue, which is the fact that finding a single minor embedding is an NP-complete problem except for some fixed pattern graphs [47]. For this reason, D-Wave offers a heuristic embedding algorithm, called **MinorMiner** algorithm [12], that tries to find quickly a good sub-optimal embedding within a certain number of steps. Specifically, it is composed of two phases. Firstly, it starts from an initial chain mapping (usually empty), then it loops over the vertices of QUBO graph to obtain a semi-valid embedding, i.e. embedding in which chains might still share qubits. Secondly, the algorithm continuously loops over the vertices to fix this semi-valid embedding. In case no progression has been found in the second phase, the algorithm is restarted.

Besides, D-Wave offers specific functions to find embedding in special graphs [47]. For instance, for clique graphs, i.e. graphs where every two distinct vertices is adjacent (fully connected graph), there is a specific function that returns an embedding with balanced chain length [10]. Due to the special structure, the problem is no more NP-complete. Therefore, it does only require a short time to find the embedding.

In general, the entanglement between qubits of the same chain does not work perfectly. In other words, after the annealing, chains can break, i.e. at least one physical qubit of the chain assume a different final state between the others. This problem is resolved during the inverse operation of embedding, named **unembedding** [17]. In this operation, the goal is to

bring back the problem into the original QUBO problem. Therefore, each chain has to be resolved regardless of its state. If the chain is not broken, i.e. all physical qubits assume the same value, then the original logical qubit of the problem is set to the unique value. Otherwise, there are some strategies used to handle the broken chain. The most popular method is to apply a majority voting, i.e. the logical qubit is assigned with the value that has the highest frequency in its corresponding chain. Moreover, discarding the solution with broken chains is another way to solve this issue.

2.6.3 Challenges

Nowadays, quantum computing still has a long way to go, the same for QA computers. A lot of challenges and issues are present in QA computers. Some of them are briefly described in the following paragraphs.

One of the biggest challenges for D-Wave's QA is the Integrated Control Errors (**ICE**) that refers to sources of infidelity in problem representation. [18]. Firstly, although qubit biases and couplers are specified as double-precision floats, some loss of fidelity occurs due to some sources connected to qubits [38, 18]. For example, DACs (digital-to-analog converters), utilized to apply quantization on biases and couplers, are one of the sources of infidelity. Therefore, ICE introduces noises in the problem formulation as follows:

$$H^{\delta}_{ising}(s) = \sum_{i \in V} (h_i + \delta h_i) s_i + \sum_{i,j \in E, i<j} (J_{ij} + \delta J_{ij}) s_i s_j \tag{2.29}$$

where δh_i and δJ_{ij} represent the errors in the parameters. These errors depend on biases and couplers of a certain qubit neighborhood [18]. Overall, this problem affects many more problems with parameters whose range of values is scattered on multiple high-density regions. For instance, a problem with a big positive density region and a big negative density region requires a lot of precision to distinguish intra-region values. Therefore, the problem solved is different from the submitted problem.

Another issue affecting the performance of the QPU is the **temperature** [18]. As stated before, the QPU, to works, needs to operate at a near 0K temperature. To ensure this, it requires to minimize the amount of energy deposited on the QPU, but during normal operations, some heat is dissipated. This can increase the temperature of qubits and thus affecting the solution quality [18].

In quantum computing, it is generally known that **decoherence**, i.e. the tendency of qubits to lose their fragile entangled states, is a common issue [38]. This happens especially at read-time since it is the instant when qubits

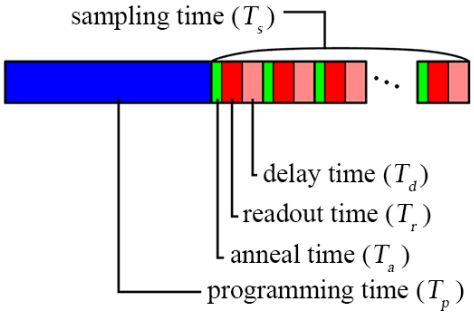

Figure 2.4: Breakdown of QPU access time (Source: D-Wave Systems Inc.)

are more affected by the outside environment. However, in QA, there is no such problem because, at the end of annealing, qubits end up in a classical state [38]. Still, there is readout infidelity of the QPU smaller than 1% [18]. This issue is not a problem because D-Wave offers the possibility to request multiple reads. In other words, the annealing process, also called **sampling**, is repeated multiple times. For instance, one out of 100 samples of a problem may be a solution that is completely wrong.

Just like every quantum computing, QA computers provide, as a benefit, the quantum speedup. However, there are still challenges ahead before it is well exploitable. First of all, a quantum annealing process considering only the anneal time is in the order of μs. However, due to noises and read errors, doing a single annealing is not enough. More specifically, by excluding the waiting time due to network and queue of other requests, the required time to solve a problem, called **QPU access time**, can be simplified into two parts: **programming time** and **sampling time** [19]. The former is the elapsed time to load the embedding problem into the quantum machine. The latter is the elapsed time to samples as many solutions as required. Moreover, the sampling time is further divided into anneal time, readout time and delay time for each sample. They are respectively the time to anneal from the initial Hamiltonian to the final Hamiltonian, the time to read the solution of the final Hamiltonian, and the time required to bring back the temperature to its initial value for the next annealing process. In Figure 2.4, it is shown how the QPU access time is broken down. Overall, since the programming time is still in the order of ms, the speedup is still present, but, nowadays, problems with more than hundreds of variables still cannot be embedded. Therefore, big data still cannot be handled by these quantum computers.

2.7 Our Work

Our work tries to merge RS with quantum computing. Nowadays, quantum computing technology has became more and more available for the public usage. Therefore, the goal of this work is to try to exploit this new technology. Moreover, noises due to the challenges of D-Wave are less problematic in RS; mainly because, in top-k recommendation task, a recommendation system is "good" even if the learning problem is not solved perfectly; especially because only the order counts in the evaluation of the system. For these reasons, we decided to opt for this research.

More in specific, the algorithm that we decided to implement with quantum computing is SLIM. The idea is to transform the item independent objective functions of Equation 2.9 in a QUBO formulation with only binary variables. The domain space of the variables becomes smaller and discrete, but post-processing operations can bring back the continuous values of the variables. Also, sparse regulators terms such as Lasso and Ridge are introduced in the new QUBO formulation in similar ways. Therefore, the new quantum SLIM problem is solved by optimizing as many objective functions as the number of items in the dataset on a quantum annealing computer.

Our focus is to analyze how the new technology behaves and compares with state-of-the-art techniques run on classical computers. Moreover, we want to analyze the actual time scalability of the quantum SLIM model on different size of synthetic datasets.

Since quantum computing is still in its infant phase, many datasets are still too huge for the most recent D-Wave machine. However, the goal of this work is to show the potential of quantum annealing computing combined with RS.

Chapter 3

Model

In this chapter, we focus on explaining all the details regarding our model starting from the initial SLIM model.

At first, in Section 3.1, we describe the QUBO formulation obtained from SLIM, the possible sparsity regulators and post-processing operations. More particularly, we start by describing the first formulation derived from SLIM (i.e. MSE loss function). Then, due to intrinsic issues, we changed the formulation by adding a normalization on the loss function. Additionally, we also described the new sparsity regulators that substitute the Lasso and Ridge terms of SLIM. Finally, due to binary variables in the formulation, we also introduced possible post-processing operations able to transform the results into floating values.

Then, in Section 3.2, we present a new variant of our model that is able to handle datasets with higher number of items by reducing the size of the QUBO. More specifically, for each independent optimization problem, we select a smaller set of items to be used in the QUBO.

Finally, in Section 3.3, we describe more in details the implementation of our model. In particular, we focus on the library used, the minor-embedding technique, and other small details.

3.1 Quantum SLIM model

Starting from the theoretical model of SLIM, the goal is to write a formulation of SLIM that is suitable for a QA computer. As described in Section 2.3.3, SLIM is a CF model that works well in implicit feedback ratings. Its optimization problem, defined in Equation 2.9, is an independent regression problem that has to be solved for each item in the rating matrix. Afterward, the similarity matrix is filled with each solution of the regression problems

in a column-wise fashion. In the end, using the prediction function defined in Equation 2.1, we obtain the predicted ratings of each combination of user and item.

In the following sections, we describe the new model that we designed, denominated as **Quantum SLIM** (QSLIM).

3.1.1 QUBO formulation

As explained in Section 2.5.1, a QA computer solve optimization problems, but they require a specific formulation of the problem. The most common formulation is the minimization of the QUBO problem. For the MSE loss function, which is already a minimization problem, it can be easily transformed in QUBO notation if the variables are considered binary. Let us first recall the SLIM optimization problem to be solved for each item i independently:

$$\begin{aligned} &\min \sum_{u \in U} \left(r_{ui} - \sum_{j \in I} s_{ji} \cdot r_{uj} \right)^2 + \alpha \cdot \sum_{j \in I} s_{ji}^2 + \beta \cdot \sum_{j \in I} |s_{ji}| \quad \forall i \in I \\ &\text{subject to :} \\ &s_{ji} \geq 0 \;\; \forall j \in I \\ &s_{ii} = 0 \end{aligned} \tag{3.1}$$

Then, by excluding the sparsity regularization terms, we obtain only the MSE portion of the objective function:

$$\sum_{u \in U} \left(r_{ui} - \sum_{j \in I} s_{ji} \cdot r_{uj} \right)^2 \quad \forall i \in I \tag{3.2}$$

By expanding the quadratic term inside the summation, we obtain:

$$\sum_{u \in U} \left(r_{ui}^2 + \left(\sum_{j \in I} s_{ji} \cdot r_{uj} \right)^2 - 2 \sum_{j \in I} r_{ui} r_{uj} \cdot s_{ji} \right) \quad \forall i \in I \tag{3.3}$$

Consequently, the first term of the internal summation (i.e $\sum_{u \in U} r_{ui}^2$) is a constant term which can be removed from the optimization problem since it is not affected by any variables. Moreover, on the second term inside the parenthesis, i.e. the square of a summation, its expansion turns out into two terms composed by the square of each term and the double products as follows:

$$\left(\sum_{j \in I} s_{ji} \cdot r_{uj} \right)^2 = \sum_{j \in I} r_{uj}^2 \cdot s_{ji}^2 + \sum_{\substack{j,k \in I \\ j \neq k}} r_{uj} r_{uk} \cdot s_{ji} s_{ki} \tag{3.4}$$

Notice that there is not a 2 in front of the double products because j and k can also exchange their values; thus, this already takes into consideration the double products. By considering all this, the independent optimization problem can be finally formulated in the QUBO notation defined in Equation 2.25 by moving the summations:

$$\sum_{j \in I} \left(\sum_{u \in U} r_{uj}^2 - 2 \cdot r_{uj} r_{ui} \right) \cdot s_{ji} + \sum_{\substack{j,k \in I \\ j \neq k}} \left(\sum_{u \in U} r_{uj} r_{uk} \right) \cdot s_{ji} s_{ki} \quad \forall i \in I \tag{3.5}$$

Notice that, since in QUBO the variables are binary, we substituted s_{ji}^2 with s_{ji} because they are equivalent for every value in $0, 1$. Additionally, for each item i, let $x = [s_{1i}, ..., s_{|I|i}]^T$ be the variables; then, in matrix notation $x^T Q x$, the square matrix Q of the previous QUBO problem is:

$$Q = R_i^{*T} \cdot R_i^* - 2 \cdot diag(R_i^{*T} \cdot R_i) \quad \forall i \in I \tag{3.6}$$

where R_i is the i-th column of the rating matrix R and R_i^* is a variant version of R where the i-th column is set to zero. Notice that the notation $diag(v)$ of vector v denotes a diagonal matrix of size $|v| \times |v|$ where the diagonal is composed by v. Additionally, the matrix notation is related to the Equation 3.5 in the following way:

- The first term $R_i^{*T} \cdot R_i^*$ covers the first term r_{uj}^2 of the first summation and the entire second summation;
- The second term $-2 \cdot diag(R_i^{*T} \cdot R_i)$ covers the second term of the first summation.

Notice that this QUBO problem takes into consideration the constraint of $s_{ii} = 0$ of the SLIM Equation 3.1 by using the rating matrix with a column of zeros, i.e. R_i^*.

Unfortunately, the MSE QUBO formulation contains in itself the rating prediction function in its not normalized form. In particular, due to having binary variables, this formulation, w.r.t. the original SLIM, has lost its expressiveness in the rating prediction function. In fact, this produces awful results in accuracy metrics. Therefore, to solve this issue, a common solution we found is to apply a normalization. In general, normalization is added to the predicted rating formula; however, this introduces non-linearity to the loss function. Hence, the solution is to add the normalization over the original rating to the Equation 3.2:

$$\sum_{u \in U} \left(\sum_{j \in I} s_{ji} \cdot r_{ui} - \sum_{j \in I} s_{ji} \cdot r_{uj} \right)^2 \quad \forall i \in I \tag{3.7}$$

Similar to what has been done with the previous formulation, if the quadratic term is expanded, then we obtain:

$$\begin{aligned}\sum_{u\in U}\left(\left(\sum_{j\in I} s_{ji}\cdot r_{ui}\right)^2+\left(\sum_{j\in I} s_{ji}\cdot r_{uj}\right)^2\right)\\ -2\sum_{u\in U}\left(\sum_{j\in I} s_{ji}\cdot r_{ui}\right)\left(\sum_{j\in I} s_{ji}\cdot r_{uj}\right)\quad \forall i\in I\end{aligned} \tag{3.8}$$

Given these three terms, we expand each of them in a similar way to what we did in the MSE formulation:

$$\begin{aligned}&(1)\quad \left(\sum_{j\in I} s_{ji}\cdot r_{ui}\right)^2=\sum_{j\in I} r_{ui}^2\cdot s_{ji}^2+\sum_{\substack{j,k\in I\\ j\neq k}} r_{ui}^2\cdot s_{ji}s_{ki}\\ &(2)\quad \left(\sum_{j\in I} s_{ji}\cdot r_{uj}\right)^2=\sum_{j\in I} r_{uj}^2\cdot s_{ji}^2+\sum_{\substack{j,k\in I\\ j\neq k}} r_{uj}r_{uk}\cdot s_{ji}s_{ki}\\ &(3)\quad \left(\sum_{j\in I} s_{ji}\cdot r_{ui}\right)\left(\sum_{j\in I} s_{ji}\cdot r_{uj}\right)=\sum_{j\in I} r_{ui}r_{uj}\cdot s_{ji}^2+\sum_{\substack{j,k\in I\\ j\neq k}} r_{ui}r_{uj}\cdot s_{ji}s_{ki}\end{aligned} \tag{3.9}$$

Thus, if we substitute these three expansions in the Equation 3.8, then, by moving some of the summations, we obtain the following equation in QUBO notation:

$$\begin{aligned}\sum_{j\in I}\left(\sum_{u\in U} r_{ui}^2+r_{uj}^2-2r_{ui}r_{uj}\right)\cdot s_{ji}^2+\\ \sum_{\substack{j,k\in I\\ j\neq k}}\left(\sum_{u\in U} r_{ui}^2-2r_{ui}r_{uj}+r_{uj}r_{uk}\right)\cdot s_{ji}s_{ki}\end{aligned} \tag{3.10}$$

Consequently, this equation, in matrix notation, represents a matrix Q defined as:

$$Q=R_i^{*T}\cdot R_i^*+1_{N\times N}\cdot(R_i^T\cdot R_i)-2\cdot(R_i^{*T}\cdot R_i)\cdot 1_{1\times N} \tag{3.11}$$

where the notation $1_{N\times M}$ refers to a matrix of size $N\times M$ containing all ones. Moreover, each term refers to different terms of the Equation 3.10:

- The first term $R_i^{*T}\cdot R_i^*$, just like in MSE formulation, refers to the term r_{uj}^2 of the first summation and the term $r_{uj}r_{uk}$ of the second summation

- The second term $1_{N\times N}\cdot(R_i^T\cdot R_i)$ refers to the term r_{ui}^2 in both summations. In particular, the scalar value $\sum_{u\in U} r_{ui}^2$ is computed by $R_i^T\cdot R_i$. Then, by multiplying it with a matrix $N\times N$ with all ones, we obtain a matrix filled completely by that scalar value.

- The third term $-2\cdot(R_i^{*T}\cdot R_i)\cdot 1_{1\times N}$ refers to the double negative terms $-2r_{ui}r_{uj}$ in both summations. Specifically, $R_i^{*T}\cdot R_i$ results in a matrix $N\times 1$ containing all the values of $\sum_{u\in U} r_{ui}r_{uj}$. Then, by doing a right matrix multiplication with $1_{1\times N}$, we obtain a final matrix $N\times N$ with the values of $R_i^{*T}\cdot R_i$ repeated along the columns.

In the end, by normalizing the predicted ratings, the issue related to the MSE formulation is fixed. This new formulation, denoted as **Normalized MSE**, presents a trivial minimum solution, which is $x = 0$. However, this can be resolved by some constraints or small changes in the objective function.

3.1.2 Sparsity regulators

As always just like in ML, an objective function requires **sparsity regulators** to avoid overfitting the problem. Another way to regulate sparsity, used more in the optimization research field, is to add constraint over the objective function.

Constraints

Fortunately, the QUBO model, even if unconstrained, can handle constraints by adding a specific penalty to the objective function that functions as a constraint. The constraint of our choice is one that fixes the number of selected variables in the solution:

$$\sum_{j\in I} s_{ji} = k \tag{3.12}$$

By using the transformation formula defined in Equation 2.26, the new term to be added to the QUBO model is:

$$P\cdot\left(\sum_{j\in I} s_{ji} - k\right)^2 \tag{3.13}$$

where P is a coefficient that specifies the strength of the constraint. Since QUBO model has additive property, i.e. the result of the addition of two

QUBO is still a QUBO, then the constraint term can be added to the problem QUBO formulation by addition. Basically, in matrix notation, the Q matrix of the constraint is:

$$Q_{constr} = P \cdot (1_{N \times N} - 2k \cdot I_{N \times N}) \tag{3.14}$$

where $I_{N \times N}$ is the identity matrix of size $N \times N$. With this constraint, denoted as **selection constraint**, solutions found from this objective function are limited in the number of selected variables. Moreover, this solves the trivial solution of the normalized MSE formulation. Notice that this formulation seen as a graph of N nodes is fully connected due to the fact that Q_{constr} has all non-zero values. Thus, if we add this constraint to the objective function, the graph representation is surely a clique.

Sparsity regularization terms

From ML point of view, common sparsity regularization terms are Lasso and Ridge. However, in binary variables, these two terms are equivalent:

$$\alpha \sum_{j \in I} s_{ji}^2 = \alpha \sum_{j \in I} |s_{ji}| \tag{3.15}$$

The penalty introduced by this sparsity term increases linearly as the number of selected variables increases. For this reason, it is denoted as a **linear sparsity** regularization term. In QUBO notation, this sparsity term affects only the diagonal of matrix Q:

$$Q_{lin\_sparsity} = \alpha \cdot I_{N \times N} \tag{3.16}$$

Due to the linearity of the previous sparsity regularization term, we also proposed a **quadratic** one:

$$\beta \left(\sum_{j \in I} s_{ji} \right)^2 \tag{3.17}$$

which affects the entire matrix Q in the following way:

$$Q_{quad\_sparsity} = \beta \cdot 1_{N \times N} \tag{3.18}$$

Differently from the linear sparsity term, this one increases in a quadratic way as more variables are selected.

Comparison between the constraint and the sparsity terms

Even if the constraint and the sparsity terms discussed previously comes from two different research fields, they have a clear tight relationship. Obviously, both of their goals are to increase the sparsity in the solution, but the addition they provide to the Q matrix is highly coupled. Basically, the constraint is equivalent to the linear combination of the two sparsity terms:

$$Q_{constraint} = -2Pk \cdot Q_{lin\_sparsity} + P \cdot Q_{quad\_sparsity} \tag{3.19}$$

with $\alpha = \beta = 1$. Obviously, a constraint with $k = 0$ removes the linear sparsity term from the relationship. Overall, the selection constraint can be seen as a specific regularization term that increases sparsity, but it limits the solution to have a certain k number of variables. Moreover, the penalty value P has to be high enough to penalize the low-energy solutions with low sparsity.

3.1.3 Post-processing

In QA, a critical limitation in solutions found from these optimization problems is the fact that results are all in binary space. Moreover, the solutions are also sub-optimal. Fortunately, as stated previously, solutions obtained from D-Wave are affected by some randomness which causes to have different solutions at each annealing. In addition, as discussed in Section 2.6.3, it is natural in sub-optimal solvers to require hundreds of samples for a problem. Therefore, our idea is to exploit all these different samples with a heuristic method to increase the expressiveness of the solutions which are the weights of the similarity matrix. Obviously, it is also possible to just choose the best sample, i.e. the one with minimum energy, and keep the binary values.

A common heuristic method that we decided to opt for is a simple aggregation of samples. Basically, this approach can be divided into 3 steps. Firstly, for each variable, it counts the number of times a variable is selected among all samples. Secondly, an operation is applied to each count value. Finally, each count value is normalized by the number of samples. Regarding the operation to be applied, it can be also a non-operation; in this case, the heuristic method is equivalent to a mean applied over all samples. Some other operations can be **logarithm** or **exponential** to respectively decrease or increase the importance of high frequent variables. Afterward, we also added another optional post-processing on top of the mean, logarithm or exponential operation . This keeps only the non-zero values about the original minimum energy sample. In this way, the solution uses the same selected variables of the minimum energy sample, but with floating values.

Obviously, there are many other possible heuristic approaches, but there is no utility in trying too many techniques. Because our purpose is to see if quantum can be applied to RS in a useful way.

3.2 Item selection method

In this section, we present a variant of the model that is able to deal with dataset whose number of items are more than the maximum number of variables embeddable in a clique graph of the D-Wave Advantage system. The idea for this variant revolves in the application of feature selection method on the items. In particular, the goal is to choose m most useful items in the rating matrix for a particular item optimization problem. In this way, the QUBO problem that has to be solved for each item is composed by only m variables instead of the total number of items in the dataset. Therefore, by selecting a number m that is less than the maximum number of variables, it is possible to embed the reduced QUBO problem into the quantum machine. However, there is still the need to solve as many QUBO problem as the total number of items in the dataset.

Regarding the feature selection method, we divided them into two categories: **absolute** methods and **relative** methods. This distinction comes from how the most useful items are chosen. The former one selects the m items to be kept in an absolute way for all the items; it means that, for all item optimization problems, these m items are the only ones used. Meanwhile, the latter methods select the m items based on the relation with the actual item optimization problem; thus, for each optimization problem, the items used in the reduced QUBO problem can be different.

For the absolute methods, we mostly considered methods from information retrieval field. In particular, we chose to select the items based on three possible measures: **absolute entropy**, **variance**, and **item popularity**.

- **Absolute entropy**: a common feature selection method is to look for the entropy, i.e. a measure of disorder. In this case, the entropy of a single item is computed by:

$$\begin{aligned} E_i &= entropy(R_i) = -\sum_{u \in U} p_{ui} \cdot \log p_{ui} \quad \forall i \in I \\ p_{ui} &= \frac{r_{ui}}{\sum_{u \in U} r_{ui}} \end{aligned} \tag{3.20}$$

  where R_i is the i-th column of the rating matrix R and r_{ui} is the rating value in row u and column i of rating matrix R.

- **Variance**: this is another feature selection method from information retrieval field, commonly known as Variance Thresholding. However, in this case, it is a selection of k features by variance:

$$\hat{\sigma}_i^2 = variance(R_i) = \frac{\sum_{u \in U} r_{ui}^2}{|U|} - \left(\frac{\sum_{u \in U} r_{ui}}{|U|}\right)^2 \quad \forall i \in I \tag{3.21}$$

- **Item popularity**: this is a common measure in RS that indicates how popular an item is; and, generally, a popular item is also highly recommended by RSs since they are mostly popular-oriented. In this case, the item popularity of a single item is computed by:

$$p_i = \sum_{\substack{u \in U \\ r_{ui} \neq 0}} 1 \quad \forall i \in I \tag{3.22}$$

  which is the count of the number of ratings for a specific item i.

Meanwhile, for the relative methods, we considered an hybrid method that uses the similarity of items to select the most useful items for a specific item. For example, first, we calculate the cosine similarity matrix of each items w.r.t. other items; then, for each item, we select a different set of m items with the highest value of similarity w.r.t. specific item. In this way, we reduced the size of the items and, possibly, chosen the best set of items.

3.3 Implementation details

Since D-Wave offers API libraries in Python, we implemented the model in the same programming language. In particular, the normalized MSE QUBO formulation and selection constraint has been implemented with some post-processing methods. Moreover, in regards to mathematical operations to prepare the QUBO formulation, we used two Python libraries, Numpy and Scipy, which provide data structures for efficient mathematical operations respectively on arrays/matrices and large sparse matrices. At last, we used also Pandas library to handle the postprocessing computation.

A critical implementation of the model is how minor-embedding is applied. As explained in Section 2.6.2, the QUBO problem has to be minor-embedded into the D-Wave quantum machine. Since the problem of minor-embedding is a NP-complete problem, we decided to use the standard heuristic function offered by D-Wave. However, there are still some details in the implementation due to two reasons. Firstly, the number of times that the QUBO problem has to be solved is as much as the number of items in

the dataset catalog. Lastly, the time complexity required to find a minor-embedding is very high. Fortunately, since all problems are fully-connected in its graph, the embedding can be shared among all problems. Therefore, we implemented an approach that fixes the embedding for all the problems. Moreover, we used the specific minor-embedding function for clique graphs.

Regarding the strength of constraints added to the QUBO model, we used a heuristic formula inspired by an initial common guess defined as:

$$Strength = C \cdot (\max Q - \min Q) \tag{3.23}$$

where C is a hyper-parameter and $\max Q - \min Q$ is the initial guess based on the matrix Q. This formula is used for the penalty value of both selection constraint and embedding chain constraint. Notice that the final strength value is different between the selection constraint and the chain constraint because the chain constraint is applied on top of the selection constraint. In other words, the Q matrix in the strength formula is different for the two constraints; therefore, the final strength value is different.

In addition, we implemented the possibility to choose the type of solvers to resolve the QUBO model. Obviously, quantum solvers such as D-Wave2000Q and Advantage system have been selected, but there are also two different types of solver: hybrid solver and classical solver. Firstly, as stated in Section 2.6, D-Wave offers also hybrid computation that solves QUBO problems with the use of both classical and quantum techniques. Lastly, for a comparison point of view, we used also a classical solver, i.e. simulated annealing, offered by D-Wave [1].

Regarding the variant of the model that reduced the number of items in the optimization problem, we implemented the three absolute methods described previously and the cosine similarity method.

[1] https://github.com/dwavesystems/dwave-neal/

Chapter 4

Results

In this chapter, we describe, firstly, the dataset we chose for our experiments and, then the results from the experiments that we conducted. In particular, we divided the chapter into 4 sections. First, the Section 4.1 explains the reason behind the choice of the datasets and some detail information about them. Second, in Section 4.2, we analyzed the behavior of our model when some hyper-parameters change to see if its behavior is as we expected. Then, in Section 4.3, we analyzed the scalability in time of the model related to the fit training process in order to see how well it fares in terms of computation speed. Finally, in the last Section 4.4, we compared the performance of the model and its variant with state-of-the-art models. In particular, we choose one non-personalized model and four other personalized models.

4.1 Datasets

In RS, as explained in Section 2.4, one of the simplest methodology to test the quality of a model is to carry out offline experiments. Therefore, we need to select public datasets to test our model. More specifically, since the model is a CF type, we require only datasets containing ratings. Moreover, since our model is a variant of SLIM, it is optimal to have implicit feedback ratings. However, it is also possible to transform explicit feedback into an implicit one by some heuristics method that transforms the ratings into binary values.

Another parameter of the dataset to consider is the number of items in the catalog. This consideration has to be taken into account because D-Wave quantum machines are limited in the number of variables that can be embedded into the machine. For instance, considering the most recent D-Wave machine, Advantage system solver (Pegasus architecture P_N with

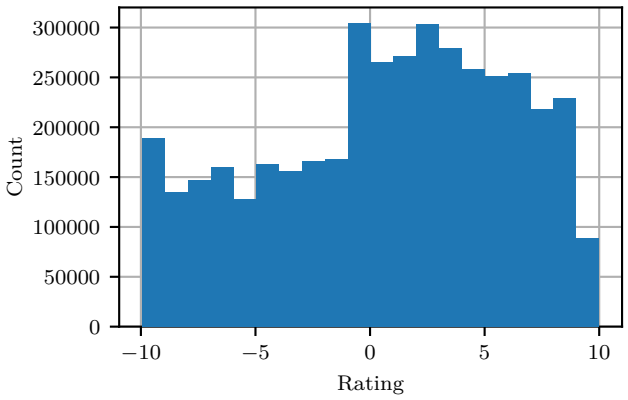

Figure 4.1: Histogram of Jester Jokes original ratings with 20 bins

$N = 16$), the highest number of variables that can be embedded theoretically is $12(N-1) = 180$, when the graph of the QUBO is a clique, i.e. fully-connected [33]. In our case, the normalized MSE formulation with the selection constraint is already a clique graph. However, the actual clique embedding function is only able to obtain an embedding of a clique of 120 variables at maximum.

For these reasons, we selected Jester Jokes and MovieLens100k. The first one is selected because it is the only one with the number of items lower than the highest number of variables that can be embedded in Advantage system solver. However, due to having only one dataset, we decided to consider also MovieLens100k even if its number of items is much larger than the maximum value for Advantage system. In particular, this second dataset is used for the variant of QSLIM that reduces the number of items in QUBO problems. Moreover, the decision came from the fact that MovieLens100k has much less items w.r.t. other datasets. In this way, the variant of QSLIM keeps about 10% of the items on this dataset.

4.1.1 Jester Jokes

The Jester Jokes[1] dataset contains rating information about jokes rated by users who visited a website created by a university in California, UC Berkeley. Actually, we decided to use a portion of the dataset that contains only 100 jokes rated by about 73 thousand users totaling 4.1 million ratings. These ratings were collected between April 1999 and May 2003. Moreover, the ratings are discrete values ranging from -10.0 to +10.0 with step of 0.1,

[1] http://eigentaste.berkeley.edu/dataset/

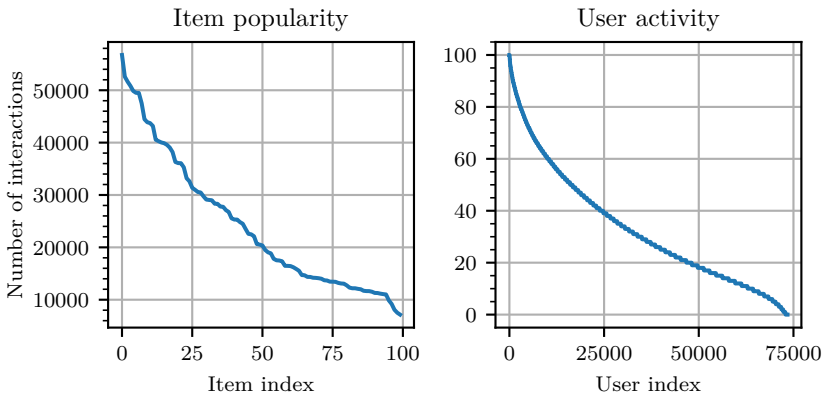

Figure 4.2: Item popularity and user activity of the Jester Jokes dataset after the transformation from discrete explicit feedback to implicit feedback

as shown in Figure 4.1.

For our model, we transformed the discrete ratings into implicit feedback based on a threshold. In particular, everything that was rated above 0 has been transformed into a positive rating with the same value. From a different point of view, we considered every joke non-rated as a "bad" joke for a certain user, while everything that was rated positively is considered as a "good" joke. This transformation also decreases the noise in the various ratings due to the high range and flexibility in the ratings.

After this transformation, the rating matrix contains about 2.4 million non-zero interactions with a sparsity level of 33%. Looking at the item popularity of the rating matrix, shown in Figure 4.2, every item is rated a huge amount of times, but some items are more popular than others. Another observation is in the user activity, shown in Figure 4.2, where few users rated all items. In other words, some users voted positively to all jokes.

4.1.2 MovieLens100k

The MovieLens[2] dataset is one of the most used benchmark dataset in RS. It is related to the MovieLens web recommender system which recommends movies to users. Moreover, it was created in 1997 by GroupLens, a research lab at the University of Minnesota. Regarding our experiments, we decided to use the MovieLens100k dataset version, containing 100k ratings from 1000 users on 1700 movies. In particular, the ratings were collected with a discrete categorization of $\{1, 2, 3, 4, 5\}$ before 1998. In Figure 4.3, it shows

[2]https://grouplens.org/datasets/movielens/

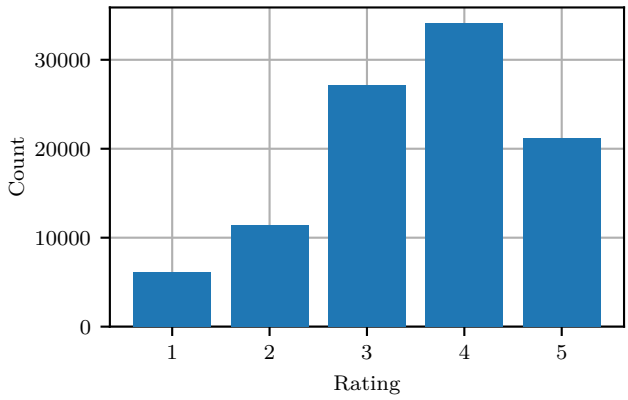

Figure 4.3: Bar plot of MovieLens100k original ratings

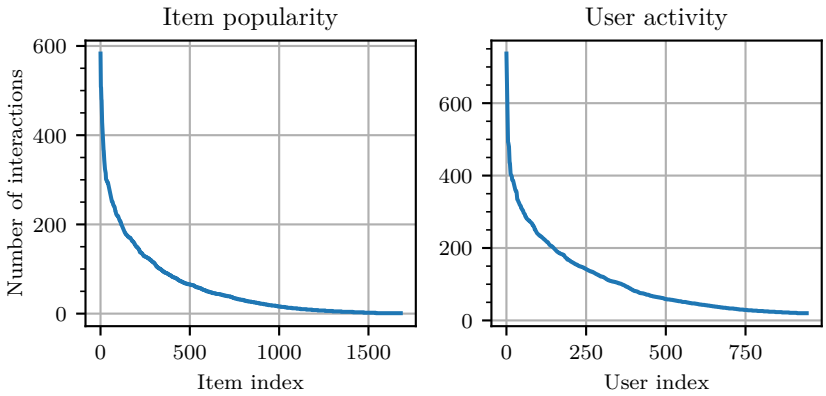

Figure 4.4: Item popularity and user activity of the MovieLens100k dataset

the original ratings of the MovieLens100k dataset, where most of them are ratings greater or equal than 3.

Since our model is, specifically, designed for implicit dataset, we transformed the explicit feedback into implicit feedback. In particular, we consider every ratings as a positive feedback of value 1. While everything that is not in the dataset is considered as negative feedback of value 0. Therefore, the final rating matrix contains ratings that are either 0 or 1. Moreover, in Figure 4.4, it shows the item popularity and the user activity of the rating matrix. These plots are quite different from the Jester Jokes one. In particular, for the user activity, MovieLens' users has no one that has rated all items. While, for the item popularity, there are items that has not been rated at all.

4.1.3 Dataset splitting

As described in Section 2.4.1, in order to evaluate correctly a RS model, we need to test it on a set of data unseen in the training procedure. Therefore, we have to split the dataset into multiple parts. For all the datasets that we used, a hold-out division method is applied because of two problems. Firstly, D-Wave offers limited computation time resources. Specifically, as a free computation time resource, it offers monthly one minute of quantum solvers computation or 20 minutes of hybrid solvers computation. Secondly, an experiment of QSLIM requires a lot of computation since it needs to solve as many QUBO problem as the number of items. Therefore, we decided to discard the cross-validation method which is, generally, less noisy.

In more detail, our division method splits the rating matrix into k-folds of rating matrices by random sampling. More specifically, for each user, it splits their ratings into the different folds by ensuring to have at least one rating for each fold. In case a user has fewer ratings than the number of folds, the user is discarded from the final k-folds. Among these k-folds, we merge some of them to obtain 3 total sets: training set, validation set, and test set. In general, the biggest set is the training set; then followed by the validation and the test set. As explained in Section 2.4.1, the training set is used in the training procedure of the model. In our case, it is the optimization of the QUBO model with one of the solver. Then, the validation set is used for comparing all the models with different parameters. Finally, the test set is used to evaluate the quality of the best model among all the ones compared in the validation set. More particularly, for our datasets, the number of folds in which they are divided is 5: 3 folds used by the training set, 1 fold for the validation set, and 1 fold for the test set.

4.2 Model analysis

In this section, we present an analysis on our model, more specifically, when it uses a quantum solver. Basically, since the model is new and uses a new technology, the goal is to see how the model behaves w.r.t. classical solvers while there are changes of some important hyper-parameters. Due to the size limitation of the quantum solver, the model is analyzed only over the Jester Jokes dataset.

To study the model behavior we applied some changes to important hyper-parameters, and then, we analyzed what happens on the samples obtained from the solvers. In particular, we focused on the chain constraint multiplier and the selection constraint multiplier explained in Section 3.3.

4.2.1 Chain constraint multiplier

As explained in Section 2.6.2, the minor embedding process introduces an equality constraint related to the chains. This means that in order to be added in the objective function, it has to add a penalty value regulated by a strength value. In our implementation, the strength value is regulated by the Equation 3.23, reported here for clarity:

$$Strength = C \cdot (\max Q - \min Q) \tag{4.1}$$

Basically, C is the **chain constraint multiplier** which is also a hyper-parameter of the model, only for quantum solvers.

In this analysis, we did a grid search on different values of the chain constraint multiplier. More specifically, the values ranges from 1.0 to 2.0 with a step of 0.2. Regarding the other model parameters, we used:

- the normalized MSE as the objective function with a selection constraint of $k = 5$ and selection constraint multiplier of 1.
- the Advantage system as the solver of QUBOs. Since this is the only QPU solver we used in all experiments, in the following sections, it will be referred as **QPU solver**;
- the number of samples acquired, for each QUBO problem, is 100.
- the post-processing aggregation of samples, simply, chooses the sample with the minimum energy. However, since the following analysis is related to samples, this parameter is irrelevant to our analysis.

From this experiment, our goal is to analyze what happens to the samples. In particular, there are two characteristic we want to analyze: the **energy** and the **chain break fraction**. The former one, as stated in Section 2.5.1, is the measure of the quality of a sample. In particular, it is the the value of the objective function, that has been minimized, related to the specific sample. Meanwhile, the latter one is the fraction of broken chains w.r.t. the total number of chains in the minor-embedding for a specific sample. Therefore, its range varies from 0 to 1, where 0 refers to the extreme case when there are no broken chains and 1 refers to the case when all chains are broken.

Chain break fraction analysis

First of all, we show the histogram plots related to different chain break fraction values. As shown in Figure 4.5, the chain break fraction values of

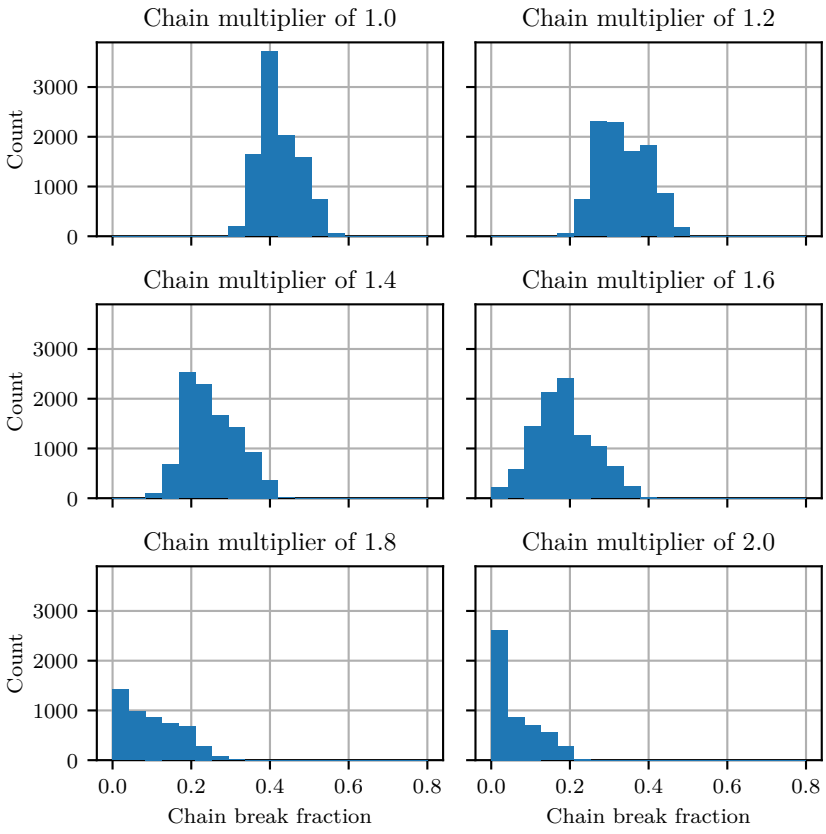

Figure 4.5: 20-bin histograms of chain break fraction values with different values of chain constraint multiplier regarding all samples of all independent QUBO problems.

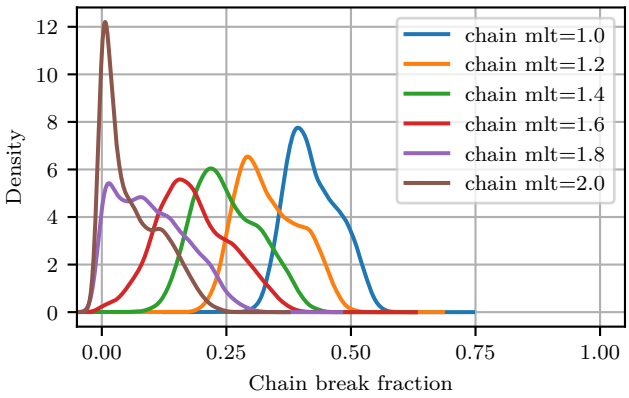

Figure 4.6: Density distribution plot of chain break fraction values with different values of chain constraint multiplier regarding all samples of all independent QUBO problems.

the samples ranges from 0.3 to 0.6 in the first plot where the chain constraint multiplier is set to 1.0. This means that, in the average, half of the chains in the minor-embedding is broken. However, as we increase the multiplier, there is a shift in the mean of the chain break fraction values, as shown in Figure 4.6 which represents six density distributions of the chain break fraction for each chain constraint multiplier. In fact, in the last plot of Figure 4.5, where the multiplier is set to 2.0, it has chain break fraction values that ranges from 0 to 0.2.

Overall, this is what we expected from the experiment: increasing the strength of the constraint decreases the probability that chains, inside the minor-embedding, breaks. However, this does not directly imply that the samples obtained from the model with the multiplier of 2.0 are better than the models with lower multiplier.

Energy analysis

Hence, we show the energies of all samples related to all QUBO problems. In order to compare the energies of samples related to different QUBO problems, we chose to use a normalized energy value. In particular, the normalization is a min-max normalization applied to each QUBO problems independently; i.e. each QUBO problem uses a different minimum and maximum energy value. Moreover, to compare it with different chain constraint multiplier models, the minimum and the maximum are calculated considering all the QUBO from all models.

The Figure 4.7 shows six density distributions of the normalized energies of all samples; each distribution is related to a different chain constraint

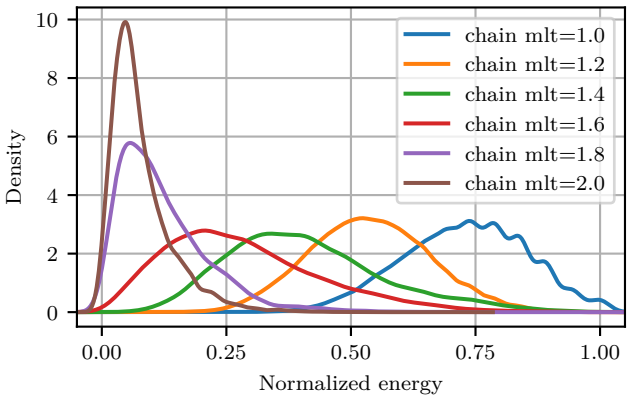

Figure 4.7: Density distribution plot of normalized energy values of all samples of all independent QUBO problems with values of chain constraint multiplier between 1 and 2.

multiplier. Similar to what we saw in the chain break fraction, there is a shift in the mean of the distribution while the multiplier decreases. This is clearly better because the energy is better when it is lower. However, we cannot exclude the possibility that continuously increasing the chain constraint strength always improves the energy. Therefore, we did further experiments with chain multiplier of 3, 4, and 5.

As shown in Figure 4.8 on the left plot, the chain break fraction values continuously decrease as the multiplier increases. However, the normalized energy on the right plot, reaches the smallest energy value when the multiplier is set to 3. But with higher multiplier, we can see that the distribution of normalized energy tends to move on the right which means that the energy values are increasing instead of decreasing. As stated in Section 2.6.3, this behavior might be related to the ICE issues of the quantum annealing technique. In particular, when the QUBO values has to be embedded on the QPU, there is a quantization that approximates the QUBO values which leads to embedding a different QUBO problem w.r.t. the one we requested.

Energy comparison with the classical solver SA

Up until now, we compared energy values only with the QPU solver; thus, we do not know how optimal the samples are. Therefore, we did another experiment using the SA classical solver for a comparison. The new experiment fits the QSLIM model by using the same parameters of the previous experiment with the exception of the solver and the chain constraint multiplier. In particular, in case the solver is not a QPU, then the minor-embedding is not

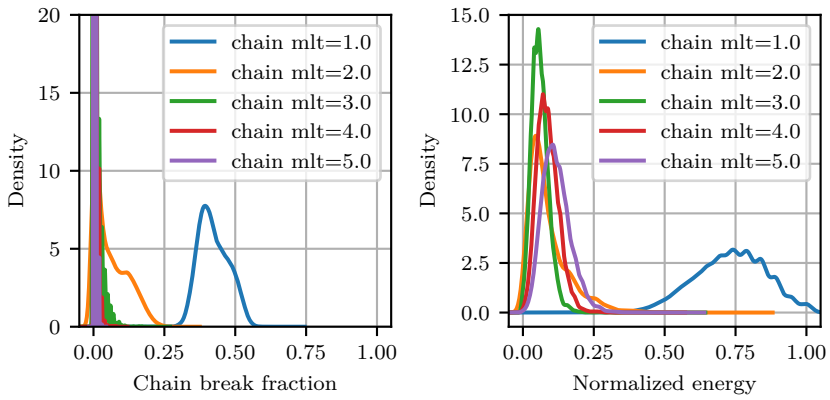

Figure 4.8: (Left): Density distribution plot of chain break fraction values with different values of chain constraint multiplier. The y-axis max value is cropped at 20 for a clearer image. (Right): Density distribution plot of normalized energy values of all samples from all independent QUBO problems with different values of chain constraint multiplier.

necessary; consequently, the chain constraint multiplier is also not needed. Similar to what we did before to compare the energy, here, we apply the same min-max normalization on all QUBO problems using minimum and maximum computed from both SA and QPU solvers.

As shown in Figure 4.9, differently from Figure 4.7, there is only the addition of the SA density distribution whose samples are all in the minimum energy value. Moreover, as we can see, the SA samples are lower in energy w.r.t. the QPU samples. From the plot, it seems that the energy values are very similar between SA and QPU. However, the values are normalized which means that there is no way to tell the real difference from the two cases. Unfortunately, looking at the real values, the ones related to SA have all negative energies in the -10^6 scale value, while the ones related to QPU are all positive values.

In conclusion, as we expected, changing the strength of the chain constraint reinforce the constraint; in fact, it decreases the chain break fraction of the samples. Furthermore, also, the energy of the samples improved. However, continuously increasing the strength of the constraint is not always better for the samples. Probably, due to the ICE issues of the QPU, e.g. quantization issue, the embedded problem in the quantum machine was completely different from the requested QUBO problem. Therefore, even if the chains are broken-free, the samples obtained are still bad w.r.t. requested QUBO problem. Moreover, we saw that the samples of the QPU solver are much worse than the ones of the SA solver.

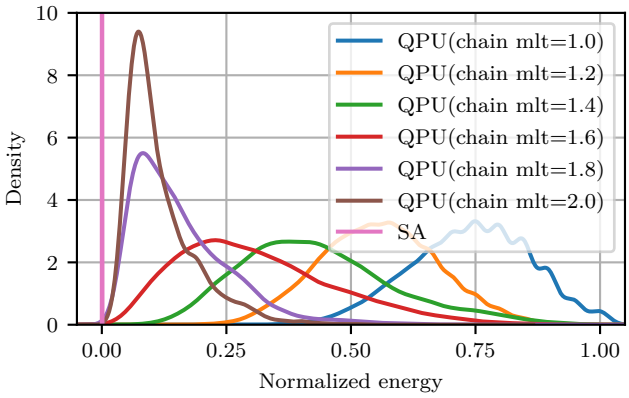

Figure 4.9: Comparison of density distributions plot of normalized energy values of all samples between SA and QPU solver. For the QPU solver, there are different distributions, each with a different value of chain constraint multiplier. The y-axis max value is cropped at 10 for a clearer image.

4.2.2 Selection constraint multiplier

The selection constraint multiplier, as explained in Section 3.1.2, is another important hyper-parameter that defines the strength of the selection constraint, which is the constraint that pushes the solution of the QUBO problem to keep a fixed number of variables selected. In particular, it acts as a sparsity term regulator. As stated in Section 3.1.2, the selection constraint can be seen as a linear combination of linear sparsity term and quadratic sparsity term. However, introducing more hyper-parameters in an environment with limited resources was too difficult to analyze and explore.

In this analysis, our goal is to see if the constraint is actually affecting the samples of the solvers and how well it is doing so. Therefore, we did some experiments by fitting many models with different selection constraint multipliers. The first experiment is characterized by:

- a normalized MSE objective function with the selection constraint of $k = 5$;
- the QPU solver, i.e. D-Wave Advantage system solver;
- a chain constraint multiplier of 2;
- a post-processing operation that selects the samples with the minimum energy for each QUBO problem;
- acquiring 100 number of samples for each QUBO problem;

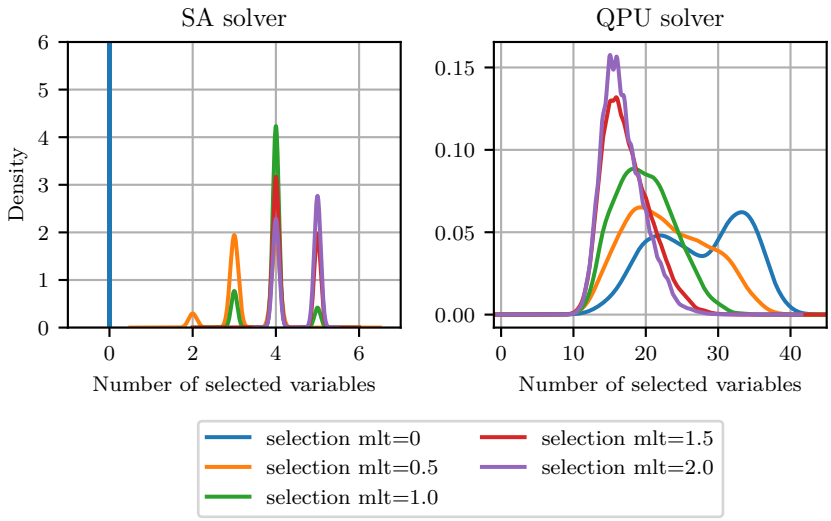

Figure 4.10: (Left): Density distributions plot of number of selected variables of all samples about all independent QUBO problems using SA solver while changing the selection constraint multiplier. When the selection constraint multiplier is set to 0, the objective function is composed by only the normalized MSE which has a trivial solution composed of all zeros. The y-axis maximum limit value is cropped at 6 for a clearer image. (Right): Density distributions plot of number of selected variables of all samples about all independent QUBO problem using QPU solver, while changing the selection constraint multiplier.

- a selection constraint multiplier that varies from 0 to 2 with a step of 0.5.

Then, for the second experiment, we did the same thing but with the SA solver. For analyzing the quality of the constraint, we focused on the number of selected variables of each sample for all QUBO problems.

In Figure 4.10, on the right-side, we can see the density distribution of the number of selected variables of the QPU solver regarding all the samples collected from each QUBO problem. In particular, the distribution evolves by shifting to the left towards the selected number of variables which is 5. However, it never reaches the selected number of variables, but it stops at about 15 as mean when the selection constraint multiplier is set to 2.0. Instead, on the left-side, there is the density distribution plot regarding the SA solver. In this case, the selection constraint works perfectly. In fact, as the selection constraint multiplier increases, the density distribution of the number of selected variables moves closer towards the selected number of variables in the constraint. Interestingly, we can notice that, for the SA

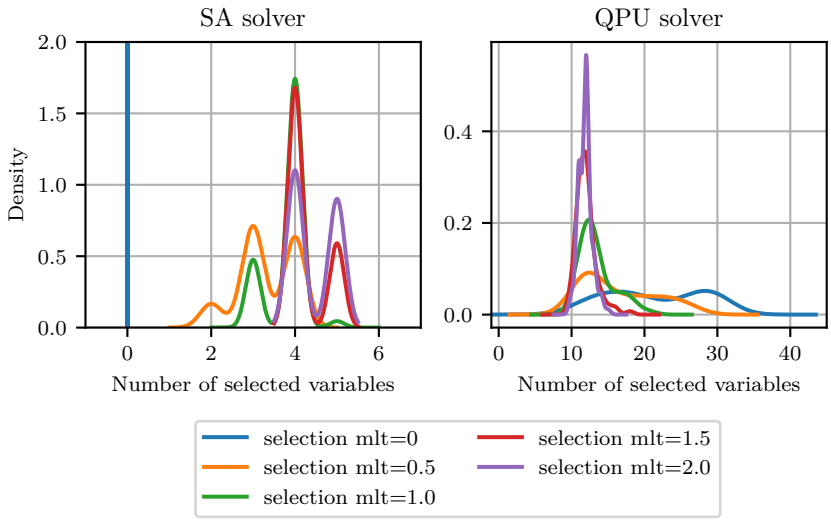

Figure 4.11: (Left): Density distributions plot of number of selected variables of the minimum energy sample of all independent QUBO problems using SA solver, while changing the selection constraint multiplier. When the selection constraint multiplier is set to 0, the objective function is composed by only the normalized MSE which has a trivial solution composed of all zeros. The y-axis maximum limit value is cropped at 6 for a clearer image. (Right): Density distributions plot of number of selected variables of the minimum energy sample of all independent QUBO problems using QPU solver, while changing the selection constraint multiplier.

solver, when the selection constraint multiplier is equal to 0, the number of selected variables is all zeros because of the trivial solution in the normalized MSE formulation explained in Section 3.1.1.

For completeness, we also report the density distribution of the minimum energy sample of each QUBO problem using both QPU and SA solver. In Figure 4.11, we can see that for the SA solver, the different density distributions did not change much. However, for the QPU solver, the density distribution of high selection constraint multiplier are less noisy in the values. In fact, for the case of selection constraint multiplier equal to 2.0, the density distribution mean is about at 11-12 with a smaller variance than before.

In conclusion, we saw that the selection constraint works as intended. In particular, for SA solver, its effect performs perfectly: by increasing the constraint multiplier, the number of selected variables in the samples is closer and closer to the selected value in the constraint. However, for QPU solver, this does not happen; at maximum, with higher strength value, the number of selected variables does not reach the selected one. Maybe, it is due to incorrect values of chain constraint multiplier or ICE issues.

4.3 Scalability analysis

In this section, we present an analysis on the scalability of the QSLIM model. In particular, the focus is on the time complexity and its scalability w.r.t. the size of the dataset. For a better analysis, we compared the QPU version with the SA version of the model and, also, with SLIM Elastic Net. First of all, the experiments are all characterized by the following features:

- **Machine**: every SA and SLIM Elastic Net experiment is executed on a 2.9 GHz dual-core computer. However, both algorithms are not implemented to exploit multi-cores. While, every QPU experiment is executed on the Advantage system solver.

- **Dataset**: since the scope of these experiments is to analyze the scalability in time w.r.t. dataset, all the datasets used are synthetic. In particular, they are randomly distributed rating matrices with a certain number of items, number of users and density of the matrix. More specifically, the standard parameters used are 50 number of items, 1000 number of users and 0.05 density (i.e. 5%). If the parameters are not specified in the following experiment description, then the values are the standard ones.

- **Model**: every experiment is executed with the following standard parameters with the exception of some of them because the experiment focus on the variation of them. First, the loss function of the QUBO is the **Normalized MSE** with a selection constraint of 5 variables. Then, the number of samples obtained from the solver for each QUBO problem is 50. Lastly, the post-processing, simply, choose the sample with the minimum energy.

- **Experiment**: for each experiment, we decided to run 5 times the fit algorithm with certain parameters. Then, we computed the mean and standard deviation of the time measured.

Moreover, the time measured in all the following experiments is the elapsed time required to solve all the QUBO optimization problems of the model, also called **fit time**. In particular, we distinguished the overall fit time into four categories:

- **Pre-processing time**: it covers all the operations related to every QUBO conversion from rating matrix to the optimization problem. In this phase, there are two fundamental operations: the computation of the Q matrix and the transformation from the Q matrix to a data

structure interpretable by the solvers. The former operation consists of a matrix multiplication of sparse matrices and other matrix operations such as summation. Meanwhile, the latter one transform the Q matrix, obtained by the previous operation, into a data structure, called Binary Quadratic Model (BQM), necessary for the solvers.

- **Waiting time**: this phase is present only on the QPU experiments where the QPU solver is used. It consists of two components: the network time and the queue waiting time. The former is about the network time required to send the problem to the QPU and to receive the sample solutions from the QPU, while the latter is the waiting elapsed time on the queue of the QPU since it is shared among all users.

- **Sampling time**: in both SA and QPU experiments, this is the time required to obtain all the solution samples requested. However, the elapsed time for these two cases scales very differently. In SA, the bigger is the QUBO, the higher is the time required to obtain a sample solution. Meanwhile, in QPU, the time required to obtain a sample is always the same, regardless of the size of the QUBO. More specifically, for QPU, the sampling time is actually the access time explained in Section 2.6.3. Therefore, for whatever QUBO problem embeddable in the QPU, the access time is constant. However, in general, with newer quantum machines, the programming time also increases. In fact, the QPU solver has at least 2× higher programming time w.r.t. D-Wave2000Q solver.

- **Post-processing time**: as stated previously in Section 3.1.3, the model requires a post-processing phase where samples obtained by the solver are aggregated or selected to obtain a unique solution for the optimization problem. This is the elapsed time required to find all unique solution of all the optimization problems, which are the columns of the similarity matrix.

In the end, the goal of these experiments consists of analyzing how the model fit time scales as some size features of the rating matrix changes. In fact, the experiments in the following section are based on the variation in the number of items, number of users, density, and number of samples.

4.3.1 Variation in the number of items

One of the most important feature of a RS is the number of items in its catalog. This is the same as the number of columns in a rating matrix. As explained in Section 2.2.1, in item neighborhood methods, the technique is based on building the similarity matrix, which is a matrix $|I| \times |I|$. In QSLIM the similarity matrix is built by solving $|I|$ optimization problems. Therefore, the number of items $|I|$ affects both the size of the optimization problem and the number of optimization problems to be solved, since the number of variables in the QUBO problem is the number of items. Without considering the size of the optimization problems, the time complexity is already $O(|I|)$. But, by considering the type of solver we are using, the time complexity might increase even more. In theory, with the QPU solver, the size of the optimization problems does not change the time required to solve it, but there is a limit in the size. Meanwhile, with SA, it depends greatly on the number of variables when the algorithm tries to find the neighbor.

First of all, we focused on rating matrices embeddable in the QPU solver. Since the largest clique size graph, that can be fitted in the Pegasus P_{16} architecture, is one with 120 variables due to the actual embedding function; thus, we opted to choose a set of number of items between 25 and 100. The first experiment, shown in Figure 4.12, compares the two solvers SA and QPU. In the first two plots, we suppose that the QPU machine is local which means that there is no waiting time. In this case, we can clearly see that SA takes much more time w.r.t. the QPU. Moreover, as stated before, SA time complexity w.r.t. number of items is much more than linear. In fact, doubling the number of items, e.g. from 50 to 100, has an increase in the sampling time of about 600%. However, if we consider the waiting time, the QPU solver requires much more time to solve an optimization problem. Indeed, we can see in the second two plots that the waiting time increases as the number of items increases since the number of optimization problems, that needs to be solved, also increases. Meanwhile, focusing on pre-processing and post-processing time, they do not have any substantial increment.

Then, for dataset with a bigger number of items, we did another experiment by using the SA solver and the variant of the QSLIM model that uses item selection method. More specifically, we used the absolute entropy as the item selection method and we reduced the size of items to 100 for each QUBO problem. In this way, the problem can be embedded into the QPU solver. However, due to limited resources, we did not really execute the experiment on QPU solver, but we used the same elapsed time computed in

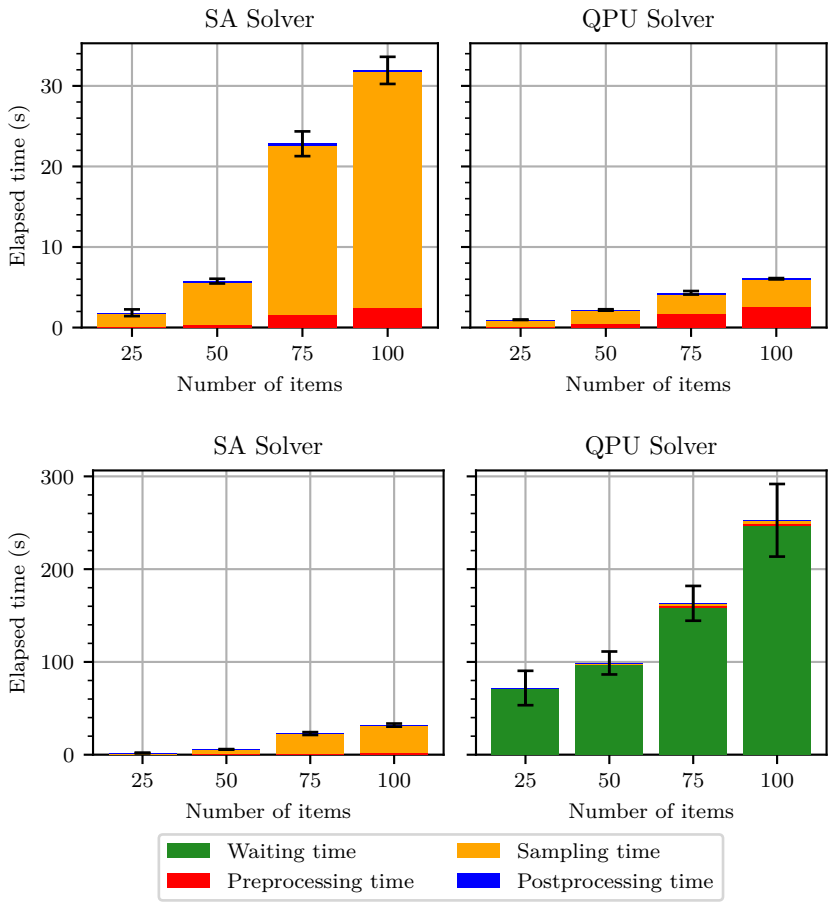

Figure 4.12: Comparison of real fit time about SA and QPU solvers while changing the number of items. The two above are plots without showing the waiting time for clarity, while the two below are plots with waiting time. Note that the error label in the graph indicates the standard deviation of the fit time.

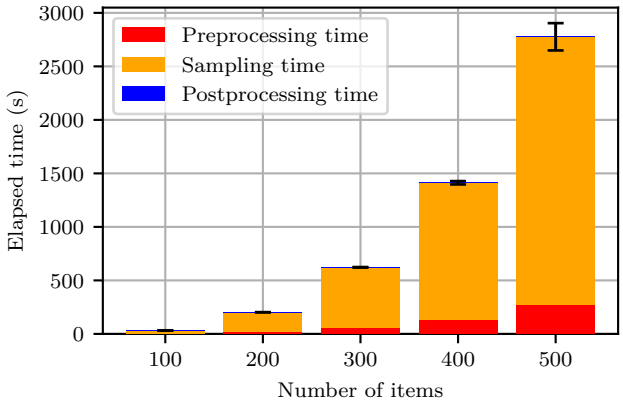

Figure 4.13: Bar plot of fit time about SA solver while changing the number of items from 100 to 500. The error label in the graph indicates the standard deviation of the fit time.

the previous experiment because it is known to be constant for each QUBO problem. Moreover, this reuse of data for QPU solver is done also in the following scalability experiments.

In Figure 4.13, it shows the elapsed fit time for the SA solver when the number of items is between 100 and 500. As we can see, the time required for the sampling time seems to increase exponentially just as we expected. Meanwhile, in Figure 4.14, there is a comparison on the variant of the model. As stated previously, each QUBO problem is reduced to 100 number of variables, but the number of QUBO problem to be solved is still the original number of items in the dataset. Thus, the SA solver with item selection takes much less time than SA solver without item selection. Instead, if we compare SA solver with item selection and QPU solver with item selection, then the QPU solver uses much less time because of the constant low sampling time. In addition, in this case, we can see that the pre-processing times increases as the number of items increases; because with bigger rating matrices, the matrix multiplication and conversion to the specific data structure requires much more time.

Additionally, as a further comparison, we also did a timing experiment on SLIM Elastic Net implemented using the Elastic Model of scikit-learn library. In particular, with all the experiment previously explained, we compared the fit time of all the variations of the QSLIM model with SLIM Elastic Net. As shown in Figure 4.15, the QSLIM model with item selection (QSLIM QPU w/IS) is slower than SLIM Elastic Net in the case where the quantum machine is local, i.e. no waiting time; this is expected since the

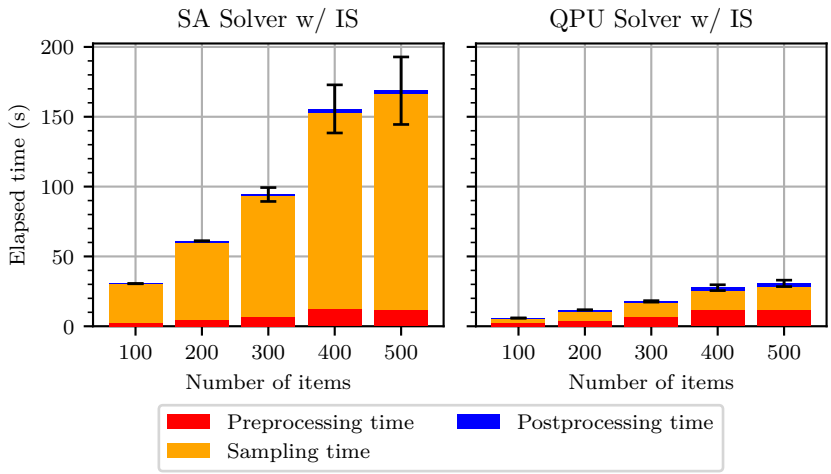

Figure 4.14: Bar plot comparison of fit time between SA solver with item selection and QPU solver with item selection, while changing the number of items from 100 to 500. The error label in the graph indicates the standard deviation of the fit time.

QUBO problem is still small. Moreover, here, we can see that SA solver without item selection has clearly an exponential behavior w.r.t. others.

In conclusion, regarding the variation in the number of items, the results are inline with what we expected. As the number of items increases, the number of optimization problems to be solved also increases; thus, the sampling time increases at least linearly. In particular, only for SA, the increment is even higher due to the increase in size of the QUBO problem. Meanwhile, for the pre-processing time, since the rating matrix is bigger, the matrix multiplication and the conversion to the data structure require more time. Instead, for the post-processing phase, there is not much variation in the time, but there is still some increment.

4.3.2 Variation in the number of users

Similarly to what we did with the number of items, we analyzed what happens when the size of the other dimension of the rating matrix, i.e. the number of users, changes. Differently from the number of items, the algorithm does not depend highly on this dimension. In theory, the number of users should affect the algorithm only in the pre-processing phase; more particularly, it only affects the matrix multiplication. Therefore, we do not expect much variation in the fit time while varying the number of users.

For this case, we did only SA experiments with the following discrete values of number of items: 1000, 5000, 10000 and 50000. As shown in

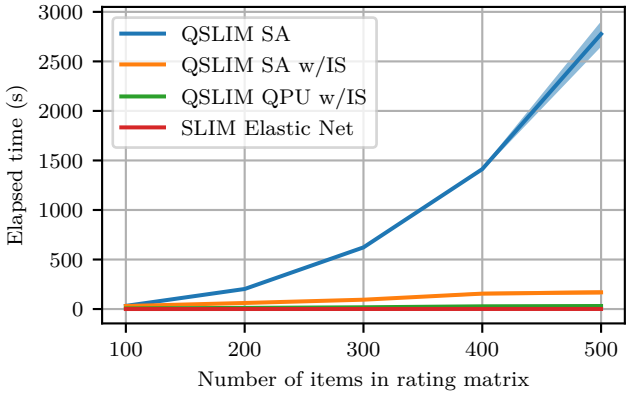

Figure 4.15: Bar plot comparison of fit time between SA solver, SA solver with item selection (QSLIM SA w/IS), QPU solver with item selection (QSLIM QPU w/IS), and SLIM Elastic Net, while changing the number of items. The error fill lines in the graph indicates the standard deviation of the fit time.

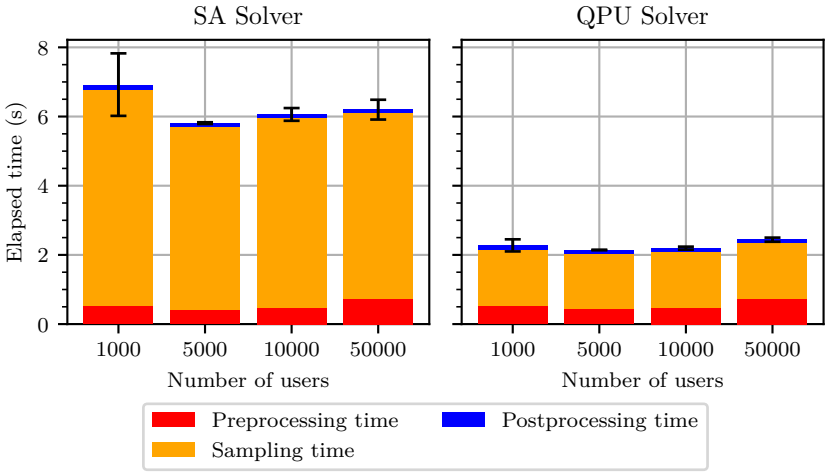

Figure 4.16: Bar plot comparison of fit time between SA solver and QPU solver while changing the number of users. The error label in the graph indicates the standard deviation of the fit time.

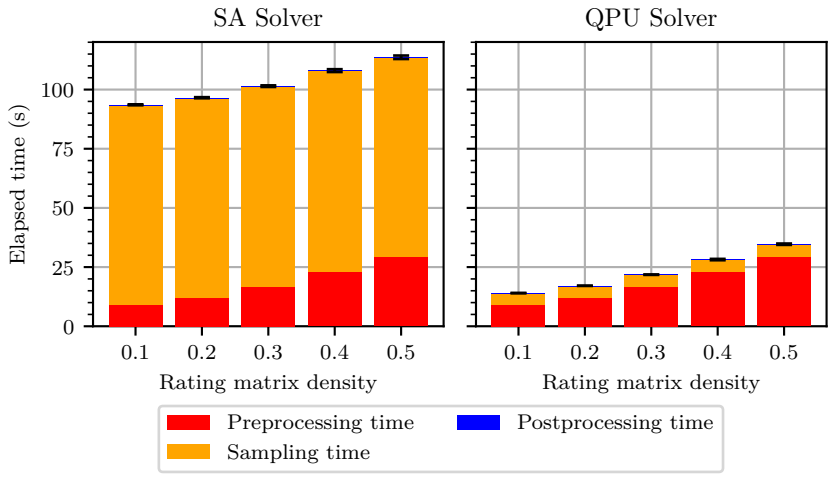

Figure 4.17: Bar plot comparison of fit time between SA solver and QPU solver while changing the density of the rating matrix. The error label in the graph indicates the standard deviation of the fit time.

Figure 4.16, changing the number of users do not affect much the fit time. However, in the case of 50000 number of items, we can see a small increase in the pre-processing time.

In conclusion, the number of users, as theorized, affects the fit time only in the pre-processing phase. However, the variation in the pre-processing time is very small in this case. Probably, with higher number of items, the increase in the pre-processing time might be more relevant.

4.3.3 Variation in the density

Another rating matrix's parameter is the density of the matrix. In general, common RS datasets have very sparse matrices, e.g. 0-5% density. In fact, the sparsity is a common problem of the RS research field. However, in this scalability analysis, what we want to see is the behavior of the time function w.r.t. the dataset. For the density, in theory, it should only change the complexity time of the matrix multiplication.

In this experiment, differently from the previous ones, we also changed the number of items to obtain longer and more accurate elapsed times. In particular, we changed it to 150 number of items, while the density changes linearly from 10% to 50% with a step of 10%. As shown in Figure 4.17, the pre-processing time increases as the density increases, just as we expected. In Figure 4.18, we can see more clearly that, by doubling the density from 0.2 to 0.4, the pre-processing time increases by almost 100%.

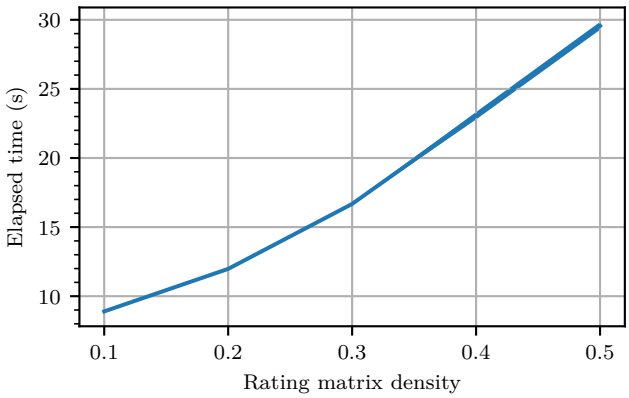

Figure 4.18: Pre-processing time of the QSLIM model while changing the density of the rating matrix. The error fill bars in the graph indicates the standard deviation of the pre-processing time.

In the end, just like we expected, the density affects only the pre-processing phase.

4.3.4 Variation in the number of samples

Finally, the last parameter, we analyzed, is the number of samples requested to the solver for each QUBO problem. Differently from the other analysis, this one focus on a parameter independent from the dataset. However, it is still a parameter that should affects the time complexity. In particular, its variation surely affects the sampling time. Moreover, it should also affect the post-processing phase due to the increase in the dimension of the number of samples.

Similarly from the previous variations, in this case, we run experiments using only SA solver with an exponential variation in the number of samples, from 10^2 to 10^6. However, we also changed some standard parameter due to timing problems. First, the number of items in these experiments is 5; otherwise the time required to compute all those samples was too much. Second, the aggregation strategy used in post-processing is changed to the mean of the samples but the only non-zero values are the one related to the minimum energy sample. The comparison results between SA and QPU solver from these experiments, as shown in Figure 4.19, are inline with what we expected. To summarize, in the figure, we can see that the sampling time increases as the number of samples increases. Notice that in this case, the QPU solver takes much more time than SA solver because the number of items is so small. Regarding the post-processing, in Figure 4.20, we can

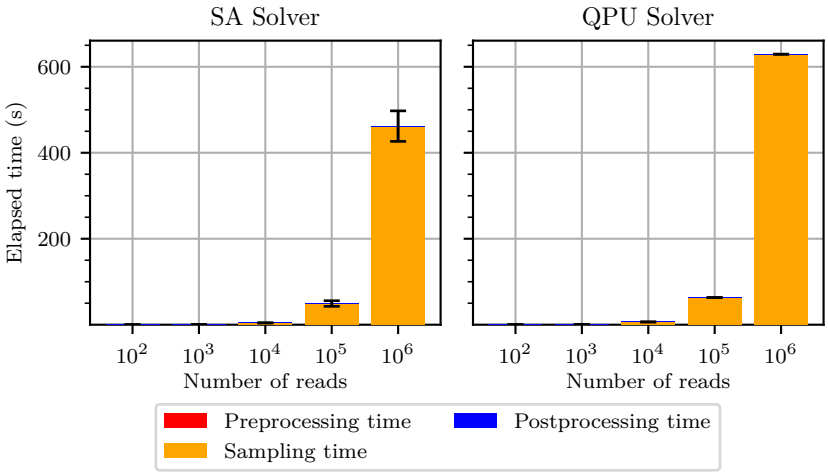

Figure 4.19: Bar plot comparison of fit time between SA solver and QPU solver while changing the number of samples. The error label in the graph indicates the standard deviation of the fit time.

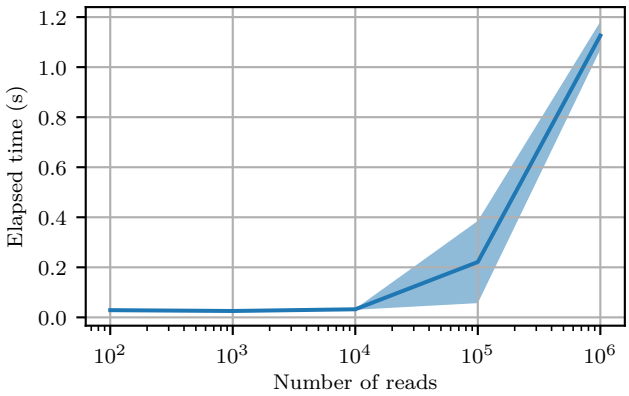

Figure 4.20: Post-processing time of SA solver experiments. The error fill bars in the graph indicates the standard deviation of the post-processing time.

clearly see that the post-processing time increases as the number of samples increases. Notice that the scale of the x-axis is logarithm; therefore, the time does not have an exponential behavior, but it is more inline with a linear behavior.

In conclusion, as we expected, the sampling time and the post-processing time depends on the number of samples. In particular, both of them seems to have a linear dependency with it.

4.4 Quality evaluations

In this section, our goal is to find the best set of hyper-parameters of our model in both SA and QPU solver that maximize a specific metric. As explained in Section 2.4, there are many metrics that measures the quality of a RS. Since our task problem is the top-k recommendation problem, a natural choice is a ranking accuracy metric; and, generally, a common choice is the Mean Average Precision (**MAP**). To sum it up, the goal is to maximize the MAP by finding the best set of hyper-parameters of the QSLIM model. However, other metrics are, also, shown in the following results. More particularly, among the accuracy metrics, there is the precision, recall, MAP and NDCG; all of these are better if the value is higher. Meanwhile, for the beyond-accuracy metrics, we reported the catalog coverage correct (CC correct), the mean-inter list diversity (MIL), and the average item popularity (AIP). In general, since they are not accuracy metrics, there is no best value; however, the goal in a RS is, generally, to have a higher diversity in the items recommendation, a higher coverage in the items recommended, and to be less popular-driven. But, this is not always true.

As explained in Section 4.1, we chose two datasets for our experiments. The first one is the Jester Jokes dataset which is easily embeddable into the QPU solver. While, the second one is the MovieLens100k dataset which is not embeddable directly into the QPU solver. Therefore, for the second dataset, we used a variant of the QSLIM model, explained in Section 3.2, that uses item selection technique to reduce the number of items to be used in the QUBO problem. Moreover, both the datasets are divided in 3 parts (training, validation, and test set) using the hold-out method as explained in Section 4.1.3. Notice that the validation set is the one used to validate the models during the hyper-parameter tuning process to find the best set of hyper-parameters.

Regarding the hyper-parameter tuning method, we used a state-of-the-art optimization search method, called **Bayesian Optimization** [24], for the global optimization of blackbox functions.

4.4.1 Baseline models

To have a better comprehension on the quality of the QSLIM models, we selected a few state-of-the-art baseline models for comparison:

- **Top Popularity**: for a minimal baseline, we chose this non-personalized model, that simply recommends unseen top popular items to the each user.
- **Item KNN CF**: an item-based KNN CF model that computes the similarity matrix with a heuristic similar measure. Since, in the variant of QSLIM, we used the cosine similarity, then we also decided to use it in this model for better comparison (see Section 2.2.1 for details)
- $\mathbf{RP}^3_\beta$: a graph-based method that follows the random walk idea used in P^3_α (see Section 2.3.1 for details)
- **MF BPR**: a matrix factorization method that learns through stochastic gradient-descent with the BPR loss (see Section 2.3.2 for details)
- **SLIM Elastic Net**: since SLIM Elastic Net, abbreviated in the following results with SLIM EN, is the model from which QSLIM is inspired upon, then we decided to use it as comparison (see Section 2.3.3 for details)

4.4.2 Jester Jokes

In this section, we present the quality evaluations of the QSLIM model compared with other previously developed models, as stated before. Since our problem is a top-k recommendation problem, we need to choose k, i.e. the number of items in the list of recommendations. Due to the small number of items in the Jester Jokes dataset, we decided to keep only 5 items in the list. Therefore, the accuracy and beyond-accuracy metrics are listed by considering a list of 5 items.

As stated previously, here, we consider only the original QSLIM model without any item selection method. Moreover, both validation and test set results are shown.

Validation set performance

First, we report the performance@5 of different best models that we found on the validation set. As shown in Table 4.1, the SLIM EN has the highest values among all the models reported in all accuracy metrics.

Model	Accuracy metrics ↑				Beyond-accuracy metrics		
	Precision	Recall	MAP	NDCG	CC correct	MIL	AIP
Top Popularity	0.2833	0.2447	0.2166	0.2594	0.2600	0.5514	0.8730
Item KNN CF	0.2884	0.2503	0.2206	0.2640	0.6100	0.6035	0.8599
$\mathbf{RP}^3_\beta$	0.2914	0.2574	0.2231	0.2681	0.5200	0.6807	0.8366
MF BPR	0.2777	0.2499	0.2100	0.2572	1.0000	0.8106	0.7393
SLIM EN	**0.3027**	**0.2707**	**0.2345**	**0.2810**	0.9700	0.7678	0.7751
QSLIM SA	0.2950	0.2630	0.2261	0.2725	1.0000	0.7540	0.7672
QSLIM QPU	0.2698	0.2224	0.1957	0.2357	0.5400	0.6704	0.8199

Table 4.1: Jester jokes' performance@5 of different tuned models (except QSLIM QPU due to limited computational resource) on the validation set (all accuracy metrics are better if the value is higher and the range of all metrics goes from 0 to 1).

Best hyper-parameters	**Value**
Selection constraint - k variables selected	22
Selection constraint - multiplier	4.4655
Number of samples (fixed during tuning)	100
Post-processing	Average of samples

Table 4.2: Best hyper-parameters of QSLIM SA model found

Best hyper-parameters	**Value**
Selection constraint - k variables selected	5
Selection constraint - multiplier	0.5
Chain constraint - multiplier	2
Number of samples	100
Post-processing	Average of samples

Table 4.3: Best hyper-parameters of QSLIM QPU model found

Model	Accuracy metrics ↑				Beyond-accuracy metrics		
	Precision	Recall	MAP	NDCG	CC correct	MIL	AIP
Top Popularity	0.3709	0.3161	0.3345	0.3460	0.4500	0.6929	0.8300
Item KNN CF	0.3886	0.3299	0.3528	0.3587	0.6600	0.7138	0.8211
RP$^3_\beta$	0.3938	0.3438	0.3591	0.3681	0.6100	0.7456	0.8075
MF BPR	0.3842	0.3394	0.3468	0.3591	1.000	0.8283	0.7302
SLIM EN	**0.4234**	**0.3727**	**0.3934**	**0.3975**	0.9800	0.8321	0.7323
QSLIM SA	0.4134	0.3627	0.3820	0.3874	1.0000	0.8112	0.7487
QSLIM QPU	0.2939	0.2359	0.2141	0.2398	0.6800	0.6936	0.7267

Table 4.4: Jester Jokes' performance@5 of different tuned models (except QSLIM QPU due to limited computational resource) on the test set using both training and validation set during the fit process (all accuracy metrics are better if the value is higher and the range of all metrics goes from 0 to 1).

Regarding **QSLIM SA**, the tuning procedure found a set of hyper-parameter as shown in Table 4.2. Interestingly, the post-processing operation applied is just the average of all the samples obtained from the solver. Meanwhile, the accuracy metrics of QSLIM SA is the second-best among all models. Instead, for the beyond-accuracy metrics, the values are quite comparable to SLIM EN and MF BPR. In particular, we can see that its correct recommendations covers all items at least once, just like MF BPR. However, this item coverage is expected since the number of items in Jester Jokes is just 100. Furthermore, the diversity and average item popularity has good value w.r.t. others since, in general, the goal of a RS is to have more diversity in its recommendations and to be less driven by popularity.

About **QSLIM QPU** model, we did not do any tuning procedure due to limited resource. In the end, we selected the model with the hyper-parameters as shown in Table 4.3. About the results, the accuracy metrics are the lowest w.r.t. all models, even lower than Top Popularity model. Additionally, in CC correct, the coverage of the items is not so great; it is quite similar to RP$^3_\beta$. Moreover, also, the other two beyond-accuracy metrics are similar to RP$^3_\beta$.

Test set performance

Furthermore, we also reported the metrics@5 regarding the test set while fitting both training and validation set. In general, the test set is evaluated only once by a model; however, since we want to validate the quality of all models obtained in the tuning process, we also tested them with the test set for another comparison.

In Table 4.4, we reported the metrics values of all the models. As we can see, since, in the fit phase, there is much more data (i.e. 33% more), then the values are much higher w.r.t. the one about validation set. However, the relationship between the models remains the same: SLIM EN is still the best in terms of all accuracy metrics. Then, followed by QSLIM SA that still has a quite comparable MAP w.r.t. SLIM EN. In particular, the values are below just by about 0.01 in all accuracy metrics. As the validation set, the beyond-accuracy metrics of QSLIM SA remains similar to SLIM EN.

Regarding QSLIM QPU, the boost it received from the data is insignificant w.r.t. others. The MAP just increased from 0.19 to 0.21: while the others had a higher jump of at least 0.12 points. Instead, if we look at the beyond-accuracy metrics, the values are still very similar to RP^3_β. Overall, this low performance might be due to inefficient hyper-parameters or to the issues shown in the model analysis done in Section 4.2. In particular, the ICE issue might have caused to obtain very sub-optimal solutions to the QUBO problems. Therefore, the final similarity matrix obtained was much worse w.r.t. QSLIM SA.

4.4.3 MovieLens100k

For MovieLens100k, since the total number of items in the dataset is more than a thousand, we decided to report performance metrics with a 10 size recommendation list. Moreover, as stated before, due to the high number of items, we used the variant of the QSLIM model that uses the item selection method to reduce the number of items in the QUBO problems. Specifically, we chose to reduce the number of items to 100 since it is embeddable in the QPU solver. However, we did not use QPU solvers in this experiment due to the limited resource in computation time. Meanwhile, for SA solver, In the following results, we distinguish two version of the variants called: **QSLIM SA AIS** and **QSLIM SA RIS**.

- QSLIM SA AIS: it is the model with SA solver that uses an Absolute Item Selection (AIS) method to choose the items to keep. As explained in Section 3.2, the absolute methods implemented are based on either variance, absolute entropy, and item popularity.

- QSLIM SA RIS: differently from AIS version, it uses a Relative Item Selection (RIS) method. The one that we implemented is based on the cosine similarity.

Also, in this case, there are two evaluation results that are reported: one on validation set and another on test set.

Model	Accuracy metrics ↑				Beyond-accuracy metrics		
	Precision	Recall	MAP	NDCG	CC correct	MIL	AIP
Top Popularity	0.1552	0.1067	0.0837	0.1200	0.0178	0.3899	0.8125
Item KNN CF	0.2476	0.1802	0.1625	0.2065	0.1361	0.8966	0.5414
RP$^3_\beta$	0.2509	0.1826	0.1609	0.2065	0.1432	0.8903	0.5547
MF BPR	0.2344	0.1705	0.1474	0.1925	0.1682	0.9057	0.5400
SLIM EN	**0.2563**	**0.1852**	**0.1652**	**0.2095**	0.1165	0.8648	0.6074
QSLIM SA AIS	0.2040	0.1464	0.1268	0.1667	0.0713	0.7931	0.6498
QSLIM SA RIS	0.2442	0.1749	0.1501	0.1948	0.1920	0.9290	0.5013

Table 4.5: MovieLens100k's performance@10 of different tuned models on the validation set (all accuracy metrics are better if the value is higher and the range of all metrics goes from 0 to 1).

Best hyper-parameters	**Value**
Selection constraint - k variables selected	17
Selection constraint - multiplier	5
Number of samples (fixed)	100
Item selection method	Absolute entropy
Post-processing	Logarithm of samples

Table 4.6: Best hyper-parameters of QSLIM SA AIS model found

Validation set performance

Just like what we did with Jester Jokes dataset, first, we report the evaluation metrics related to the validation set. As shown in Table 4.5, we can see that SLIM EN is still the most accurate model. However, in terms of beyond-accuracy metrics, SLIM EN, just like the other baseline models, has lower coverage, lower diversity and more popularity-driven w.r.t. QSLIM SA RIS. This does not mean that QSLIM SA RIS is better in terms of beyond-accuracy metrics, but if we need a RS that has better coverage, higher diversity in the list, and less popularity-driven; then the QSLIM SA

Best hyper-parameters	**Value**
Selection constraint - k variables selected	10
Selection constraint - multiplier	3.5051
Number of samples (fixed)	100
Item selection method	Cosine similarity
Post-processing	Weighted average of samples

Table 4.7: Best hyper-parameters of QSLIM SA RIS model found

Model	Accuracy metrics ↑				Beyond-accuracy metrics		
	Precision	Recall	MAP	NDCG	CC correct	MIL	AIP
Top Popularity	0.1876	0.1146	0.1102	0.1328	0.0279	0.5044	0.7498
Item KNN CF	0.3291	0.2192	0.2603	0.2593	0.1658	0.9151	0.4910
RP$^3_\beta$	0.3495	0.2312	0.2843	0.2774	0.1747	0.9011	0.5101
MF BPR	0.3146	0.2064	0.2379	0.2425	0.1837	0.9129	0.5000
SLIM EN	**0.3663**	**0.2435**	**0.3042**	**0.2924**	0.1533	0.8961	0.5410
QSLIM SA AIS	0.2932	0.1961	0.2205	0.2319	0.0939	0.8767	0.5568
QSLIM SA RIS	0.3478	0.2278	0.2753	0.2721	0.2366	0.9399	0.4537

Table 4.8: MovieLens100k's performance@10 of different tuned models on the test set using both training and validation set during the fit process (all accuracy metrics are better if the value is higher and the range of all metrics goes from 0 to 1).

RIS has these characteristics.

Interestingly, as shown in Table 4.7, the post-processing of QSLIM SA RIS is a weighted average of samples. In particular, this heuristic method weights the samples based on their min-max normalized energy, i.e. the samples with the minimum energy has a weight of 0, while the samples with the maximum energy has a weight of 1. Moreover, as stated before, QSLIM SA RIS uses cosine similarity for the item selection process. Compared with the Item KNN CF with cosine similarity, QSLIM SA RIS can be seen as a hybrid method with Item KNN CF where the KNN algorithm is substituted by the QUBO problem. However, in Table 4.5, we can see that the advantage of this substitution is not good in terms of accuracy metrics.

Meanwhile, in Table 4.6, we reported the best hyper-parameters of QSLIM SA AIS that we found. Unfortunately, with absolute entropy, its performance is not very good. At least, it has higher accuracy metrics w.r.t. Top Popularity.

Test set performance

In the test set performance, as shown in Table 4.8, the results confirm what we found in the validation set performance. Just like in Jester Jokes, the metrics values have a substantial increase in the values due to the addition to the validation set during the fit process. Moreover, as in the validation set performance, SLIM EN remains the best in accuracy metrics.

Interestingly, QSLIM SA RIS model overcome the Item KNN CF model in accuracy metrics. Regarding the beyond-accuracy metrics, we can still see quite a difference in QSLIM SA RIS with its high coverage, high diversity and low average item popularity.

Overall, QSLIM SA RIS show very promising results. Its accuracy metrics compares well w.r.t. state-of-the-art models; moreover, it has interesting results in the beyond-accuracy metrics. This means that QSLIM QPU with item selection method with less noise might obtain similar results.

Chapter 5

Conclusion

In this thesis, we proposed a new learning-based collaborative filtering model based upon SLIM in Recommender System. Specifically, the main novelty of this approach is the exploitation of quantum annealing computers to solve the core problem of the algorithm, i.e. QUBO problems. Of course, these optimization problems can also be solved with classical heuristic algorithms, such as Simulated Annealing. However, due to various limitations, the new model is only capable of handling datasets of smaller item sizes. Therefore, we also proposed a variant of the model that can handle more datasets by reducing the QUBO problems to a specific size. In this way, it allows the QUBO problem to be embeddable in quantum computers.

Furthermore, we analyzed the model behavior by visualizing some features of the samples obtained from different QUBO solvers. As we expected, there are no issues in the model behavior. Especially in case, the solver is the classical heuristic algorithm. However, the quantum annealing solver presents two problems in the solution found. First, chains produced in the minor-embedding phase can break. Even if it is possible to manage the chains' strength, some broken chains remain when the strength is high enough. Second, the quality of the solutions is not as good as compared with the classical solver. These issues might originate from the noise introduced by the Integrated Control Errors (ICE) and the temperature stability of the quantum computers.

Additionally, we also analyzed the scalability in time of our model to evaluate how good is the quantum speedup. Overall, we have seen that, with various synthetic datasets of different dimensions, there is the quantum speedup w.r.t. the classical solver only if the quantum computer is executed locally. However, we need to consider that the classical solver was implemented to run on a single-core.

In the end, we compared our model with other state-of-the-art collaborative filtering approaches. Unfortunately, due to computational resource limitation, we could not find a good set of hyper-parameters of our model solved with quantum computers. Hence, its performance is nowhere near w.r.t. other models. Anyway, we expect our model to have a promising future since the performance of our model solved with the classical solver is quite good. Moreover, the beyond-accuracy performance shows intriguing results.

Regarding possible future works, when newer quantum computers will become available, it will be possible to carry out further experiments on larger datasets. Moreover, if the computation resource increases, then it might also be possible to use the hybrid solver that we could not use. Additionally, another future work might be to improve the model. For instance, in the sparsity regularization terms, we could represent them in a more general way. Due to limitations, we added a constraint to the objective function; however, it is possible to use the two sparsity terms.

Bibliography

[1] A survey on quantum computing technology. *Computer Science Review*, 31:51 – 71, 2019.

[2] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Managing popularity bias in recommender systems with personalized re-ranking, 2019.

[3] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[4] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1), Jan 2018.

[5] B. Apolloni, C. Carvalho, and D. de Falco. Quantum stochastic optimization. *Stochastic Processes and their Applications*, 33(2):233–244, 1989.

[6] Steven M. Beitzel, Eric C. Jensen, and Ophir Frieder. *MAP*, pages 1691–1692. Springer US, Boston, MA, 2009.

[7] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22(5):563–591, 1980.

[8] Zhengbing Bian, Fabian Chudak, William G Macready, and Geordie Rose. The ising model: teaching an old problem new tricks. *D-wave systems*, 2, 2010.

[9] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109 – 132, 2013.

[10] Tomas Boothby, Andrew D. King, and Aidan Roy. Fast clique minor generation in chimera qubit connectivity graphs. *Quantum Information Processing*, 15(1):495–508, 2016.

[11] Robin Burke. *Hybrid Web Recommender Systems*, pages 377–408. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[12] Jun Cai, William G. Macready, and Aidan Roy. A practical heuristic for finding graph minors, 2014.

[13] Pau Farré Catherine McGeoch and William Bernoudy. D-wave hybrid solver service + advantage: Technology update. Technical report, D-Wave Systems Inc., September 2020.

[14] R. Chen, Q. Hua, Y. Chang, B. Wang, L. Zhang, and X. Kong. A survey of collaborative filtering-based recommender systems: From traditional methods to hybrid methods based on social networks. *IEEE Access*, 6:64301–64320, 2018.

[15] Nick Craswell and Stephen Robertson. *Average Precision at n*, pages 193–194. Springer US, Boston, MA, 2009.

[16] D-Wave Systems Inc. *DWave Problem-Solving Handbook*, September 2020.

[17] D-Wave Systems Inc. *Getting Started with the D-Wave System*, September 2020.

[18] D-Wave Systems Inc. *Technical Description of the D-Wave Quantum Processing Unit*, March 2020.

[19] D-Wave Systems Inc. *Solver Computation Time*, January 2021.

[20] Prasanna Date and Thomas Potok. Adiabatic quantum linear regression, 2020.

[21] Yashar Deldjoo, Maurizio Ferrari Dacrema, Mihai Gabriel Constantin, Hamid Eghbal-zadeh, Stefano Cereda, Markus Schedl, Bogdan Ionescu, and Paolo Cremonesi. Movie genome: alleviating new item cold start in movie recommendation. *User Modeling and User-Adapted Interaction*, 29(2):291–343, 2019.

[22] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, January 2004.

[23] Maurizio Ferrari Dacrema. Demonstrating the equivalence of list based and aggregate metrics to measure the diversity of recommendations (student abstract). In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, 2021.

[24] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 9–13. Springer, Cham, 2019.

[25] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.

[26] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. Recsys 2010, pages 257–260, New York, NY, USA, 2010. Association for Computing Machinery.

[27] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models, 2018.

[28] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[29] Feng Hu, Ban-Nan Wang, Ning Wang, and Chao Wang. Quantum machine learning with d-wave quantum computer. *Quantum Engineering*, 1(2):e12, 2019.

[30] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008.

[31] Daisuke Inoue, Akihisa Okada, Tadayoshi Matsumori, Kazuyuki Aihara, and Hiroaki Yoshida. Traffic signal optimization on a square lattice using the d-wave quantum annealer, 2020.

[32] Y. Kanamori, S.-M. Yoo, W.D. Pan, and F.T. Sheldon. A short survey on quantum computers. *International Journal of Computers and Applications*, 28(3):227–233, 2006.

[33] Jack Raymond Kelly Boothby, Paul Bunyk and Aidan Roy. Next-generation topology of d-wave quantum processors. Technical report, D-Wave Systems Inc., February 2019.

[34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[35] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[36] Matevž Kunaver and Tomaž Požrl. Diversity in recommender systems –a survey. *Knowledge-Based Systems*, 123:154–162, 2017.

[37] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. Recommender system application developments: A survey. *Decision Support Systems*, 74:12 – 32, 2015.

[38] C. C. McGeoch. *Adiabatic Quantum Computation and Quantum Annealing: Theory and Practice*. Morgan & Claypool, 2014.

[39] Catherine McGeoch and Pau Farré. Solver computation time. Technical report, D-Wave Systems Inc., September 2020.

[40] I. C. Mogotsi. *Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: Introduction to information retrieval*. 2010.

[41] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. pages 497–506, 12 2011.

[42] Bibek Paudel, Fabian Christoffel, Chris Newell, and Abraham Bernstein. Updatable, accurate, diverse, and scalable recommendations for interactive applications. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 7(1):1–34, 2016.

[43] Pearl Pu, Li Chen, and Rong Hu. Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction*, 22(4):317–355, 2012.

[44] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[45] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, 2nd edition, 2015.

[46] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Nov 1994.

[47] Stefanie Zbinden, Andreas Bärtschi, Hristo Djidjev, and Stephan Eidenbenz. Embedding algorithms for quantum annealers with chimera and pegasus connection topologies. In Ponnuswamy Sadayappan, Bradford L. Chamberlain, Guido Juckeland, and Hatem Ltaief, editors, *High*

Performance Computing, pages 187–206, Cham, 2020. Springer International Publishing.

[48] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.