CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Optimization of Recommender Systems
Student:	Radek Bartyzal
Supervisor:	Ing. Pavel Kordík, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2016/17

Instructions

Review algorithms behind recommender systems and approaches how to evaluate their quality. Focus on online evaluation metrics and AB testing methodologies. Implement visualization methods allowing to evaluate quality of different parameter settings of recommender systems. Design and implement an online optimization of these systems based on AB testing results. Employ surrogate modeling to reduce the number of necessary evaluations. Demonstrate the functionality of the optimization method on several databases and discuss the performance in specific use cases (e.g., small number of user interactions).

References

Will be provided by the supervisor.

L.S.

doc. Ing. Jan Janoušek, Ph.D.	prof. Ing. Pavel Tvrdík, CSc.
Head of Department	Dean

Prague November 20, 2015

Czech Technical University in Prague

Faculty of Information Technology

Department of Theoretical Computer Science

Bachelor's thesis

Optimization of Recommender Systems

Radek Bartyzal

Supervisor: Ing. Pavel Kordík, Ph.D.

10th May 2016

Acknowledgements

I would like to thank my supervisor Ing. Pavel Kordík, Ph.D. for his professional guidance, and Ing. Tomáš Řehořek for his various insights into the recommender systems. I would also like to thank my family and friends for their support throughout my whole studies.

Declaration

Citation of this thesis

Bartyzal, Radek. *Optimization of Recommender Systems.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Tato práce popíše typy doporučovacích systémů, algoritmy stojící za nimi a způsoby hodnocení těchto systémů se zaměřením na online metodiky. Dále představí nový SAOOA algoritmus inspirovaný evolučními strategiemi, který s využitím modelu z gaussovké směsi optimalizuje nasazené doporučovací systémy za běhu. Fungování algoritmu bude objasněno na simulovaných problémech a nakonec otestováno na skutečných doporučovacích systémech. Spolu s algoritmem budou také představeny metody zobrazení jeho vnitřního stavu nebo kvalit jednotlivých konfigurací optimalizovaného systému.

Klíčová slova Doporučovací systémy, online optimalizace, A/B testování, gaussovská směs

Abstract

This thesis will review classification of recommender systems, the most frequently used algorithms behind them and approaches how to evaluate them with a special attention paid to the online methodologies. It will continue with description of a new SAOOA algorithm inspired by evolutionary strategies that is capable of optimizing the recommender systems online using a Gaussian mixture model. The exact functionality of the algorithm will be explained using simulated experiments and tested on real world recommender systems. Methods to visualize the quality of different parameter settings of a recommender system or the state of the algorithm will be presented as well.

Keywords Recommender systems, online optimization, A/B testing, Gaussian mixture

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With more and more information being easily accessible on the Internet, users are forced to spend equally more time looking for the most relevant ones, whether it is the best mobile phone to suit their needs or the perfect movie for their evening. That is the reason why personal recommender systems were invented. To help users with the selection by recommending them what they might be looking for. Thanks to the growing amount of information the recommender systems are on a large number of sites today, such as: Youtube [2], Netflix [3], Steam [4] and others. Which makes this topic even more interesting.

During the evolution of recommender systems many different algorithms have been invented. Because these algorithms perform differently in various domains and tasks, it is important to know which one is going to perform best under specific conditions. Therefore several methods and metrics have been proposed to compare them. We are going to review the most frequently used algorithms and some of the comparison methods.

After selecting the best algorithm to suite our domain, another problem arises. Due to the fact that the domain constantly changes, it is necessary to optimize settings of the system online to ensure a lasting performance. We will present an optimization method designed with that purpose in mind and demonstrate its functionality on both simulations and real world databases.

1.2 Goals

Main goals of this thesis are:

- Design and implement an online optimization method that will search for the best configuration of a recommender system while overcoming noise and changes in the environment.
- Review algorithms behind recommendation systems and approaches how to evaluate their quality.
- Implement visualization methods allowing to evaluate quality of different parameter settings of recommendation systems.
- Demonstrate the functionality of the optimization method on several databases and discuss the performance in specific use cases.

1.3 Outline

The thesis starts by describing and categorizing the recommender systems and algorithms they use in Chapter 2. It continues by presenting the proposed optimization algorithm (Chapter 3) and follows by testing it and demonstrating its ability to work in real environments (Chapter 4). The work is concluded by a summary of achieved results and suggestions for future work.

CHAPTER 2

Related work

This chapter describes recommender systems, methods to evaluate and optimize them and the most popular algorithms. It also explains other terms and methods used during the design of the proposed optimization method described in the next chapter.

2.1 Recommender systems

Recommender (or recommendation) systems were originally defined as a system that takes user's recommendations as input and forwards them to users that may find them interesting [5]. Since then, many other ways of producing recommendations have been introduced and as a result the definition has been broadened to any system that provides recommendations of items interesting to the user [6].

The recommended *items* mentioned previously represent what the system works with, which can be almost anything nowadays, news articles, music, movies, electronics, books or jobs, basically anything that is offered online.

With an ever increasing amount of content on the Internet the popularity of recommender systems flourishes due to a phenomenom called *information overload* meaning that users are no longer capable of browsing through all the offered items in a reasonable time span. Recommender systems fight with this modern problem by tailoring the recommendations to each user, effectively personalizing their experience by showing them relevant items [7]. Another thing that increased the interest in recommender systems was a *Netflix prize*, a competition to design a better recommendation system with a winning prize of one million dollars [8, 9].

Recommender systems are usually divided into four categories based on how they produce the recommendations [10]:

- *Collaborative filtering* generates recommendations based on what users with similar past behavior were interested in [11]. While it can recommend novel items that would be otherwise left undiscovered by the user it also suffers from a problem called *Cold start* meaning it needs a lot of information about a large number of users to be able to match them together and recommend relevant items [12].
- *Content based filtering* generates recommendations consisting of items similar to the ones that are interesting to the user. This eliminates the need of large number of users with known histories. On the other hand, completely novel items, not similar to any of the items the user interacted with in the past, are not recommended at all [13].
- *Knowledge based* uses a knowledge of user's preferences to generate recommendations of items with corresponding attributes. In comparison with collaborative and content filtering it does not need to gather any data about users because every recommendation is unique and based solely on the knowledge that the user provided [10].
- *Hybrid approaches* combine two or more types together. Thanks to its ability to benefit from strengths of multiple previously described methods while at the same time reducing their weaknesses, the hybrid approach is used most often [6]. A simple example may be using a Content based filtering till enough information is accumulated about the user and then switching to Collaborative filtering.

2.2 Evaluation of Recommender systems

With an increasing popularity of recommender systems in past years, a large number of different algorithms was invented which prompted a need to evaluate these algorithms to be able to recognize the best ones for a specific task and domain.

Unfortunately, evaluating recommender system and their algorithms has proven to be very difficult for multiple reasons [14].

First, recommender systems can be required to fulfill different tasks, for example trying to predict user's ratings is not similar to increasing revenues or suggesting novel items.

Secondly, different algorithm were designed with different goals in mind, some are trying to recommend as many items as possible and some are focusing on the highest accuracy of predictions. A lot of different goals exist and it is up to discussion which ones or what combinations of them best represent the domain's needs.

A brief introduction into both of these problems combined with an overview of different experimental settings is provided in the following paragraphs.

2.2.1 Experimental settings

In order to decide which one of the recommender algorithms should be deployed or whether is a new algorithm better than the current one, it is necessary to compare several candidate solutions.

Three different types of experiments are being widely used, since each one of them has its strengths and weaknesses they are usually combined to achieve a more complex evaluation. [15]

2.2.1.1 Offline evaluation

Offline evaluation uses previously collected data to compare the algorithms, therefore it assumes that the user behavior captured in these historic data will be similar to the behavior at the time of deployment.

Main advantage of offline evaluation is that it does not require any user interaction making it very fast and cheap. On the other hand, it allows to evaluate only the quantitative properties of the algorithm such as prediction accuracy.

As a result, the offline evaluation is mostly used to rule out the substantially unsuitable algorithms whose evaluation would be a waste of an expensive user's time or it could it even drive them away if tested during an online experiment.

2.2.1.2 User study

User studies consist of gathering a group of users and asking them to do certain tasks that interact with the recommender system while their behavior is observed and recorded. The biggest advantage of this experimental setting is that not only the quantitative evaluation of the algorithm are received such as, how many times the user clicked on the recommendation but assessment of the qualitative properties of the whole recommender system is also possible by asking the participants questions before, during and after the experiment. An example of otherwise unreachable feedback is whether the user is satisfied with the recommendations or if he trusts the recommender system not to violate his privacy.

On the other hand, user studies are very expensive and the more accurate representation of the user base is required the more participants are needed. There is also a possibility that the behavior of the monitored users will be different because they are aware of being observed or that their answers will not be truthful due to various reasons.

2.2.1.3 Online evaluation

During online evaluation the system is being used by real users without knowing that they are a part of an experiment. It is usually realized by directing a random samples of users towards specific variants of recommender system that are being compared.

Due to the fact that this type of evaluation is basically a *controlled experiment*, similar terminology is being used, it unfortunately varies throughout the literature and that is why are the most important terms explained below:

- *Overall Evaluation Criterion* (OEC) or simply an evaluation metric. Choosing a relevant metric is a very important and difficult task that is more thoroughly discussed in Section 2.2.2. If multiple different metrics are chosen, it is recommended to combine them to a single global metric that represents the quality of the whole system [16].

- *Variable* or *factor* is one specific part of the system that can be controlled and the influence of its change is needed to be tested. For example if the recommender system uses k-nearest neighbors algorithm, the k can serve as a variable.

- *Version* or *level* is one setting of a *variable.* If we use the example from above, each specific k is one of its *levels.*

- *Variant* of the recommender system is a state of the system that is presented to one group of users, therefore it is one assignment of *levels* to *variables.*

Multiple versions of controlled experiments exist, which one to use depends on how much user traffic is available and whether are all the *variants* implemented and ready to be compared to the default *control variant.* Brief overview of the possible experiments follows:

- *A/B Testing* is the simplest representative of controlled experiments. One *variable* with two *levels*, A and B, is tested. The original *variant* of the system (A) is called *Control* and the one that is being tested (B) is *Treatment.*

- *A/B/n Testing* is a modification of *A/B Testing* where is still tested only one *variable* but with multiple *levels.* All the *variants* that are being compared to the original *Control variant* are called *Treatments.*

- *MultiVariable Testing* (MVT) is a modification of *A/B/n Testing* where multiple *variables* are tested. The advantage over testing a single *variable* at a time is that it is much faster and we can find out the possible interactions between the *variables.* On the other hand, the interpretation of the measured data is much more difficult.

- *Null test* or *A/A test* is when both groups are exposed to two exactly the same *variants* of the system. It is recommended to run this kind of test prior to any other online evaluation to make sure that the experiment has been set up correctly. For example whether is the user splitting unbiased or whether has statistically enough data been accumulated before making any assumptions. [17]

While we compare the different *variants*, the global metric of the system can be optimized by redirecting more users towards the promising *variants* according to, for example, methods similar to *Multi-armed bandit* algorithms [18].

Online evaluation is the most relevant comparison of different candidate solutions allowing us to asses the system in real world environment and gives us unique access to overall evaluations such as total revenues.

While undoubtedly very useful, it has to be approached with caution because testing a system recommending wrong items can deter users from using it ever again, negatively impacting the whole system as a result. It is therefore recommended to make offline experiments first, followed by a user study and then polish the final recommender system during an online evaluation.

2.2.2 Tasks and corresponding metrics

Recommender systems were initially evaluated according to their accuracy of prediction of user's preferences however a lot of studies recently showed that this kind of metric does not represent the real qualities of the system

[19, 20, 21]. One of the examples pointing out this problem is that recommending most popular items tends to have a high accuracy but users are usually already familiar with them, therefore these suggestions are not at all useful.

In order to relevantly compare recommender systems, a proper evaluation metric or their combination needs to be chosen. This choice is heavily influenced by the recommender's task because selecting a wrong metric can lead to choosing an improper algorithm for the task at hand [22].

The majority of recommender systems can be divided into three distinct categories according to their task with their evaluation in mind [14].

2.2.2.1 Recommending good items

The most typical recommendation task is to recommend items that the user finds interesting [23]. Finding such relevant items can be achieved by multiple ways, one of the examples is an internet-based retailer Amazon which is providing a *"Customers Who Bought This Item Also Bought"* list to users browsing their store. The most frequently used algorithms are described in Section Algorithms of Recommender systems.

If are all items considered as relevant or irrelevant according to user's preferences, the resulting recommender system is very similar to information retrieval systems such as web or library search engines, therefore similar metrics are being used to evaluate both of these systems [24].

The most popular metrics in information retrieval are *Precision* and *Recall* [25], which are computed from a *confusion matrix* constructed much alike Table 2.1.

items that were:	actually relevant	actually not relevant
predicted relevant	#TP	#FP
predicted not relevant	#FN	#TN

Table 2.1: Categorization of recommended items according to their relevance

.

#TP (true positives) represents the number of recommended items that were actually relevant to the user, #FP stands for false positives, #FN for false negatives and #TN for true negatives, their meaning follows the example of #TP and can be easily derived from the aforementioned table. [15]

$$Precision\ (P) = \frac{\#TP}{\#TP + \#FP} \tag{2.1}$$

$$Recall\ (R) = \frac{\#TP}{\#TP + \#FN} \tag{2.2}$$

Precision (2.1) is the fraction of relevant recommended items or the probability that a recommended item is relevant. Whereas *recall* (2.2) also called *True positive rate* or *Sensitivity* is the fraction of all relevant items which are recommended or in other words the probability that a relevant item is recommended. [20, 15]

Consider the following example:

If only one item is recommended and it is relevant then *precision* is 100% and *recall* is very low, on the other hand if all the possible items are recommended then the result is exactly the opposite.

It can be clearly seen that both of these measures are heavily influenced by the number of recommended items, as it gets larger the *recall* increases and the *precision* decreases. Setting the size of the recommended list to a fixed number leads to a task called *Top-N Recommendations.* If the size varies, evaluation can be done for example by calculating *precision-recall* curves [15].

Another thing that can be clearly seen, especially in the second scenario, is that the *recall* alone does not represent the quality of the recommender system very accurately, therefore both of these metrics need to be considered together. Many possibilities exist but the most popular one is F_1 score (2.3), combining the measures by calculating their harmonic mean.

$$F_1\ score = \frac{2 * P * R}{P + R} \tag{2.3}$$

The alternative to the *precision* and *recall* is the *Area under ROC curve* (AUC) where ROC stands for *receiver operating characteristic* [26]. ROC curve represents *recall* against *false positive rate*, its objective is to maximize *recall* while not recommending irrelevant items (minimizing *fall-out* (2.4)) [24].

$$False\ positive\ rate\ (fallout) = \frac{\#FP}{\#FP + \#TN} \tag{2.4}$$

It has been suggested that many of the traditional metrics used in information retrieval are not suitable to properly evaluate the recommender systems due to the different context. For example returning a known item is not at all wrong in information retrieval but it is usually not considered as a good recommendation [20].

Currently many researchers agree with that view and as a result a large number of new metrics have been proposed based on what exactly is the recommender system used for [19, 21, 23, 20].

One of the proposed popular metrics is the *coverage* or *catalog coverage* [22]. It represents the portion of all the items that is the system capable of recommending to users. For example a system returning only the best selling products may have a high precision and accuracy but it will surely have a low coverage. A similar concept is applied in *user coverage* which represents a portion of users that can the system generate recommendations for.

Among other metrics are, for example, *diversity* representing how diverse are the recommendations which is not as straightforward to calculate as the previous metrics. And even some that are nearly impossible to measure, such as user's satisfaction with the recommender system, his trust in it or *serendipity*, a feeling of receiving a novel, unexpected and interesting recommendation [21].

2.2.2.2 Optimizing utility

Another important task is the optimization of some utility function which in most cases translates into maximizing profit. Increasing revenues is logically the main aim of e-commerce websites and recommender systems can help with that goal by multiple ways [27]:

- Turning browsers into buyers by helping them finding what they want to buy.
- Improving Cross-sell by introducing users to new items that they would otherwise overlook.
- Increasing loyalty of users by making more relevant recommendations.

Recommending relevant items is still important, other aspects just have to be taken into consideration. If a site profits from the time users spend on it, for example a free news site with adverts, it will try to recommend items prolonging their session. On the other hand if users pay for the ability to find quickly what they need, the goal of the recommender system is the exact opposite.

Quality of such recommender system can be evaluated either through global metrics, or by calculating the utility of recommended items.

The global metrics directly measure the success rate of increasing the utility function and are therefore suitable to use for online evaluation of the recommender systems in which case they are called *online metrics*. Such metrics are usually quite simple, typically measuring a total increase in profit or how many recommendations led to a purchase. They are in most cases defined by the owner of the domain that is deploying the recommender system.

Due to the fact that multiple items are recommended at once, the utility of the whole recommended lists needs to be evaluated. This process is also called *ranked scoring* or *accuracy of ranks* because the recommender system assigns a higher rank to items that are more likely to be preferred by the user (are ranked high by the user).

One of the most popular ways to calculate the expected utility of a list is to multiply the probability that the user looked at the recommended item by the item's utility [28]. Estimation of that probability can be divided into two scenarios:

- A small number of clearly visible recommendations is shown on the website, for example four article headlines at the bottom of the page. In this case it can be assumed that the user skims through all of them noticing anything interesting. Meaning that the probability of him looking at a recommended item is 100% and therefore, the combined utility of shown items is a good estimate of the list's expected utility. [14]

- A list consisting of large number of recommendations or a short one offering a possibility to show more of them. Due to the large number of recommendations, the user cannot be expected to pay the same attention to all of them. Breese et al. presented a popular *half-life* utility score which establishes that each next item in the list has exponentially lower probability to be noticed by the user [28].

Other ways to calculate the expected utility of a list exist, such as *Pearson's product-moment correlation* or *NDPM (Normalized Distance-based Performance Measure)* measuring the correlation of the predicted rank to the rank of real preferences of the user [24].

2.2.2.3 Predicting ratings

Sometimes a knowledge of how would a user rate a specific set of items is needed. The most popular example is the movie provider Netflix which organized the previously mentioned *Netflix prize* consisting of predicting the ratings of a large set of items.

The predicted ratings can be later used to directly generate recommendations of items with highest ratings or as a part of a more complex recommender system.

From evaluation point of view this is a well understood case similar to classical evaluation of regression and classification algorithms, which has been studied for a long time [29, 30, 31].

Undoubtedly the two most popular metrics to measure the prediction accuracy are *Root Mean Square Error* (RMSE) and *Mean Absolute Error* (MAE). Both of them compute the size of error while predicting user ratings, the difference is that the RMSE penalizes large errors more whereas MAE considers all errors equal [32].

Accuracy of rating predictions can be conveniently evaluated offline. The measured data is split into two parts: *training set* which is used to train the recommender system which in turn creates a model supplying the recommendations and *testing set* that is used to evaluate the model.

Important part of the evaluation process is how exactly are the two sets created and subsequently evaluated. Multiple methods exist, each one with their strengths and weaknesses, among the most frequently used are:

- *k-fold cross-validation*: Data is split into k folds of approximately same size that do not overlap. Validation is then run k times. Each of the folds is used in turn for testing while the others are used for training. This process can be repeated with different splits which would improve the validation but at the increased computational cost. [29]

- *Leave-one-out cross-validation*: A special case of *k-fold cross-validation* where k equals the number of available data instances. Therefore only one instance is being used as a testing set during a validation run which translates into low bias but high variance. [29]

- *Bootstrap*: A bootstrap sample is created by uniformly sampling n instances from dataset of size n with replacement. It is then used for training while the rest is used as a testing set. This process is usually repeated for multiple bootstrap samples. Due to the sampling with replacement resulting validation has a large bias but low variance. [29]

In recommendation world the data is represented as users rating items, these interactions are registered at a certain time. Two approaches are possible, either is the time completely ignored and all the gathered data are split into the two sets or a certain time is chosen and all the interactions registered before that time are used as a training set and the model is tested against the data collected after the selected time [15].

2.3 Algorithms of Recommender systems

Recommender systems are in absolute majority used to generate a list of *Top-N Recommendations*, therefore all the algorithms described in this section share this particular goal.

The simplest algorithms are *non-personalized* which means that they recommend the same items to everyone. They are easy to implement and therefore usually serve as a benchmark for the more sophisticated ones. One of the representatives is for example an algorithm called *Top Popular* recommending items with the highest amount of ratings [33].

Many *personalized* algorithms exist but most of them are based either on traditional data mining such as *Association rules* or on collaborative filtering methods recommending items according to ratings from a certain neighborhood [1].

2.3.1 k-Nearest Neighbors algorithms

k-Nearest Neighbors algorithms(k-NN) use *neighbors* meaning similar items or users to predict the rank of items not yet rated by the user. The highest ranking items are then recommended, the general outline of this group of algorithms can be seen in Algorithm 1.

Rating values are stored in a $n \times m$ user-item matrix called R which is depicted in Figure 2.1. Rows represent users and columns represent items, therefore r_{23} is a rating that has user 2 given to item 3.

The calculation of similarity is usually either *Cosine* based or *Correlation* based comparing either row or column vectors [34].

Cosine based similarity equals to the cosine of the angle between two vectors, it ranges from -1 meaning that the vectors are completely opposite to 1 which means identity. The exact calculation is specified in Figure 2.2. Whole vectors can be compared with missing values replaced by zeroes [33].

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1m} \\ r_{21} & r_{22} & r_{23} & \dots & r_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & r_{n3} & \dots & r_{nm} \end{pmatrix}$$

Figure 2.1: User-item matrix of rating values.

Similarity based on correlation is calculated as a *Pearson's correlation coefficient* also called *Pearson's r* shown in Figure 2.3. In comparison with *cosine* similarity it requires selection of ratings that exist in both vectors, for example when comparing two items, only the users that rated both of them are taken into consideration whereas during the *cosine* approach, all information is used.

$$\text{cosine similarity} = \cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum\limits_{i=1}^{n} a_i b_i}{\sqrt{\sum\limits_{i=1}^{n} a_i^2} \sqrt{\sum\limits_{i=1}^{n} b_i^2}}$$

Figure 2.2: Cosine similarity of two vectors $\vec{a}$ and $\vec{b}$ derived from Euclidean dot product $(\cdot)$ [1].

$$\textit{Pearson's } r = r_{ab} = \frac{\sum_{i=1}^{n} (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{n} (a_i - \bar{a})^2} \sqrt{\sum_{i=1}^{n} (b_i - \bar{b})^2}}$$

$$\bar{a} = \frac{1}{n} \sum_{i=1}^{n} a_i \qquad \bar{b} = \frac{1}{n} \sum_{i=1}^{n} b_i$$

Figure 2.3: Correlation similarity of a and b, two samples of ratings of size n calculated as a *Pearson's r* [1].

Algorithm 1: General outline of neighborhood algorithms

```
input  : Number of items to be recommended $N \in \mathbb{N}$,
         Number of neighbors used for ranking $k \in \mathbb{N}$,
         User to recommend items to $u$,
         List of all items $Items$,
         User-Item matrix of ratings $R$
output: $N$ items to be recommended
foreach $item \in Items$ do
    if $item \notin u.rated\_items$ then
        $item.rank \leftarrow$ rank_according_to_nearest_neighbors($k, u, item$)
descending_rank_sort($Items$)
return top($N$, $Items$)
```

Both user and item based variants of the k-NN algorithms are very similar. Which one to use usually depends on the dimensions of the R matrix with a goal of minimizing the computational cost.

2.3.1.1 User-based k-NN

User-based k-NN algorithms calculate the rank of item i for a user u by firstly finding the k most similar users by comparing their row vectors of matrix R (see Figure 2.1) and then aggregating the neighbor ratings of item i in a way described in a Figure 2.4.

$$rank(k, u, i) = \sum_{\hat{u} \in N^k(u)} similarity(u, \hat{u}) \cdot (r_{\hat{u},i} - \overline{r_{\hat{u}}})$$

Figure 2.4: Predicted rank of the item i for the user u calculated from k most similar users to u.

The $\overline{r_{\hat{u}}}$ denoting an average rating of user $\hat{u}$ is subtracted from his ratings to reduce a bias coming from the fact that users approach rating scales differently. For example, some consider 3 stars out of five as an above average rating while others may have the same feeling about 4 stars. The similarity can be calculated using these adjusted ratings as well.

The calculated rank could be divided by $\sum_{\hat{u} \in N^k(u)} similarity(u, \hat{u})$ to normalize it into the original rating scale but because the point of the al-

$$rank(k,u,i) = \sum_{\hat{i} \in N^k(i)} similarity(i,\hat{i}) \cdot r_{u,\hat{i}}$$

Figure 2.5: Predicted rank of the item i for the user u calculated from ratings of u.

gorithm is to find N most relevant items, predicting the exact ratings is not necessary. The advantage of not normalizing the resulting rank is not losing the information about the confidence of the predicted rank which means that items relevant to more neighbors have a higher rank. This variant of rank calculation has been found to be working best with *cosine* similarity and together with it is called *Non-Normalized Cosine Neighborhood* (NNCosNgbr) [33].

2.3.1.2 Item-based k-NN

The item-based k-NN calculates the rank a little bit differently from the user based variant. First it finds the k most similar items to i, from the ones that the user u rated, by comparing the corresponding columns of the matrix R. Then these ratings of neighbor items are combined according to Figure 2.5.

The reduction of bias of the user ratings explained in User-based k-NN can be applied to calculation of similarity but is not used on the rating multiplying the similarity because all the used rating are from the user u.

Non-Normalized version of the rank calculation offers the same advantages that has been mentioned in User-based k-NN.

2.3.2 Association rules

Mining of association rules is used to find which products are likely to be bought together or in case of recommender systems which items are relevant to the user together [35].

To formally define the rules let $\mathcal{I}$ be a set of items and $\mathcal{T}$ a set of transactions with a single transaction T_u representing all the relevant items for a user u. The association rule between two non-empty subsets of $\mathcal{I}$, X and Y such as $X \cap Y = \emptyset$ represents that if X is relevant to a particular user then Y is probably relevant to him as well. It is usually denoted as $X \Rightarrow Y$.

To recommend N items to a user u, the algorithm firstly mines the association rules that are *supported* by u, which means that the user finds all the items in X from a rule $X \Rightarrow Y$ relevant. Then the algorithm selects top N items according to the highest ranking rules [1].

One of the most important metric to evaluate the quality of the association rules is *support*, calculated as a fraction of users that find $X \cup Y$ relevant [35]. It can be used to limit the number of acquired rules by removing those that do not meet some predefined *minimal support*.

A metric that is usually used to rank the rules is called *confidence* and calculated as:

$$confidence\ (X \Rightarrow Y) = \frac{\text{number of users that find } X \cup Y \text{ relevant}}{\text{number of users that find } X \text{ relevant}} \tag{2.5}$$

It can be also interpreted as a conditional probability that $X \cup Y$ is relevant to a particular user given that X is relevant to him [1].

Many other evaluation metrics exist, for example *lift* defined as a *confidence* of the rule $X \Rightarrow Y$ divided by a fraction of users that find Y relevant [36].

Using association rules can result in new and unexpected recommendations which makes them a perfect candidate for environments where novelty is desired.

2.4 Optimization of Recommender systems

Although optimization of recommender systems can mean anything with a goal of improving the quality recommendations, this thesis focuses on optimization through parameters of the algorithms used to generate the recommendations.

This kind of optimization is basically an optimization of a multidimensional function representing the quality of the recommender system where each dimension represents a specific parameter of an algorithm. This problem has been studied extensively in the literature and is still studied today [37, 38].

The gradient or hessian of the function usually cannot be computed and therefore methods such as simulated annealing [39], genetic algorithms [40] or particle swarm optimization [41] are used.

Optimization can be done either offline or online. Offline means that we optimize according to the interactions collected in the past while online refers to changing the deployed system and evaluating the results in real time.

2.4.1 Offline optimization

Because of the fast evaluation time made possible by the offline nature of the optimization we can spend a large number of evaluations in search for the global optimum, and it does not matter which points are tested.

On the other hand, due to the fact that the recommender system is being optimized on historic data, it may not perform well at the time of deployment. Another disadvantage is that once is the system optimized and running, it cannot react to any changes in the environment and therefore its performance deteriorates over time.

2.4.2 Online optimization

In comparison with offline optimization each evaluation takes much more time because we have to wait for the interactions to happen, instead of already having them collected. The evaluation time is further increased in order to reduce the noise that comes with every real world testing. And what is more, testing wrong system configurations can lead to disappointed customers.

The online optimization methods therefore have to cope with these problems, mainly the small number of evaluations and the noise. Among the most prominent methods used under these circumstances are *evolutionary strategies* and *surrogate modeling*. Both are briefly described in the following sections 2.5 and 2.6.

The crucial advantage of online optimization is that it is based on real time data which allows to adapt the recommender system according to the latest changes in the environment.

2.5 Evolutionary strategy

In order to optimize recommender systems we need to find the best settings of their parameters, and doing it online means that we have to spend as less evaluations as possible. Similar problems are being solved by evolutionary strategies.

Evolutionary strategies (ES) are evolutionary algorithms with individuals represented by vectors of real values. Their usual goal is to optimize an objective or a quality function by finding the best values of a given set of parameters [44].

The ES algorithm can be simply described in 2 steps:

1. Create new individuals (offspring) by changing or combining the old ones (parents). The changes of the individuals, called *mutations* usually follow a Gaussian distribution.

2. Evaluate the individuals and reduce the population to the original size by removing the worst ones.

A special variant of ES is a *self adapting evolutionary strategy* which keeps a global parameter affecting the creation of the new individuals and changes it from generation to generation. That makes the evolutionary strategy able to react to changes in either the composition of the population or the environment in which is the ES running. An example of such parameter can be a *mutation rate* specifying the possible size of mutation [44].

2.6 Surrogate model

The online quality of different recommender system configurations is an example of a highly accurate model that is too expensive to run. It is impossible to measure all possible configurations in a reasonable time. Therefore, the only way to get the desired results is to use a simpler, less accurate approximation of the model. This kind of an approximation is called a *surrogate model* or a *metamodel* because it is basically a model of a model [42].

The whole model is usually fitted to a number of points evaluated by an experiment or a high fidelity simulation to ensure that it represents the complex reality to at least some degree [43].

Due to the fact that the *surrogate model* is much simpler, it also reduces the amount of real world noise present in experimentally gathered data and is therefore much more suitable for various optimization tasks.

Chapter 3

Design and implementation

This chapter describes the final version of the designed algorithm. It starts by presenting the environment in which is the proposed algorithm going to run and then follows with its detailed description. A special attention is paid to important or innovative parts. The chapter is concluded with a brief section about the implementation.

3.1 Environment

Recombee [45] is providing its recommender system as a service to many e-commerce companies wanting to maximize their profits by having as many users as possible consuming the items they offer. The whole system is therefore already working, with all the algorithms already parallely implemented and optimized both for speed and memory consumption. Our goal is to extend the recommender engine by an automated optimization algorithm which will continuously improve its performance by finding better parameter settings.

All the collected data is being stored in a central database with a separate schema for each of the deployed recommender systems. The whole A/B testing methodology, described in Section 3.2, is already prepared and has an interface in form of a database table called **ab_testing** present in each schema. Resulting program wrapping the proposed algorithm takes the generated set of new configuration parameters and inserts them into this table. The recommender system will then start the A/B testing of those configurations. After a sufficient amount of information is collected, the algorithm evaluates the configurations by querying tables containing the collected recommendations, user actions and conversions.

Therefore, all the interactions with the rest of the recommender system are only through the database. As a result, the proposed algorithm is capable of optimizing any recommendation algorithm that allows to change its parameters through the A/B testing interface table.

3.2 Evaluation method

A variant of *A/B testing* called *Multivariable testing* is employed to evaluate multiple configurations of the recommender system at the same time. Each of the configurations consists of multiple parameters allowing us to find the optimal combination of their values, which would be otherwise complicated because some of the parameters influence each other.

All users coming to the site are randomly distributed into groups of approximately even size and each of that groups is assigned to one of the tested recommender system configurations. For a relevant evaluation, a sample of certain minimal size needs to be gathered to reduce the noise. In our case, that means generating at least a couple thousands of recommendations by each of the tested variants before evaluating them by a chosen metric.

In most cases a *conversion rate* is the selected evaluation metric. It represents a portion of recommendations that the user not only interacted with, but also consumed the recommended item, meaning that he purchased, watched or read it, with respect to what the current domain offers. This metric is a popular way of evaluating recommender systems because it is directly tied to increasing the number of satisfied users.

In case of web sites with a number of purchases too low to make reliable evaluations possible in a reasonable time, mostly due to a small amount of visitors, an *action rate* is used as an evaluation metric instead. It captures the portion of recommendations that the user clicked on. Although interacting with the recommendation undoubtedly means that it caught the user's attention, it does not have to mean that he found what he was looking for and it is therefore not clear whether will such recommendation lead to an increase of revenues or not. On the other hand, we observed some correlation between *actions* and *conversions* during our experiments, meaning that while not perfect, the *action rate* appears to be a usable substitute for the *conversion rate*.

3.3 Optimization algorithm description

In order to adaptively optimize the recommender systems online through the use of A/B testing, we propose a Surrogate Assisted Online Optimization Algorithm (SAOOA) that has been inspired by Evolutionary Strategies.

It is designed to explore the complex multidimensional space of system configurations while overcoming the real world noise and frequent changes of the environment. To make that possible, a Gaussian mixture model, representing the probability that a certain configuration will be selected for evaluation, is being adapted each generation according to the current population of individuals. Individual, a real valued vector of parameter values, represents a single configuration of the recommender system.

An outline can be seen in Algorithm 2, where N is the number of individuals added to the population in one generation. Because we are evaluating every individual through A/B testing, a larger N requires more users for a proper fitness measurement, therefore it is usually small (< 10).

M is the number of last generations kept in population, more generations means more data, which translates into a more detailed mixture model, but it also means slower reaction to changes. Therefore, we set the M according to the speed of change in the environment. We have experimentally found that $M = 3$ works well under all sorts of different conditions.

A *fitness* of a configuration refers to the value received by evaluating it, and a *gaussian* refers to a single Gaussian function of the Gaussian mixture model.

The algorithm starts by sampling an initial population from a Gauss distribution with a mean placed where we estimate the optimal configuration and a standard deviation chosen according to how sure are we about that estimate. If we cannot estimate the optimal configuration in any way, we simply place the function at the center of the configuration space and set the deviation large enough to enable generation of any configuration.

After the creation of the initial population, the actual optimization starts. Firstly, new individuals are evaluated according to a method described in Section 3.2, then the standard deviation of the Gaussian functions, making up the mixture model, is updated. Importance of this step and how exactly is it done is described in Section 3.6. After that, the whole mixture model is created as a sum of gaussians representing each of the individuals. Some of the gaussians are penalized, which is a process examined in Section 3.5, and all of them are multiplied by a weight exponentially correlating with the number of generations they are in the population. Other details of the model are explained in Section 3.4.

Finally, new individuals are sampled from the mixture model and the whole cycle starts again. And because of the ever changing environment, it never stops.

3.4 Gaussian mixture model

The GMM, representing the probability of new individuals being generated at certain coordinates, is not technically a *surrogate model* as described in Section 2.6, because we evaluate the selected individuals online each generation. On the other hand, its aim is to simplify the noisy real world environment and to provide a way to generate new individuals easily.

Each parameter is being optimized inside a defined interval of values. All used gaussians have diagonal covariance matrices with standard deviation in each dimension proportionate to the size of interval corresponding to the parameter representing that dimension. The exact way of calculating the standard deviation is described in Section 3.6.

When sampling new individuals, they cannot be generated closer than $\sigma/2$ from each other (in every dimension) because that would have two negative consequences. Firstly, it would reduce the exploration of the available space which is undesirable due to the expensive evaluation of each individual and secondly, the gaussians representing individuals very close to each other, or even on top of each other, would merge and create an impression of an individual with much higher fitness than it actually has. This is different from a scenario when gaussians with centers $\sigma/2$, or more, away from each other merge and create one very high "hill". That just means that we are very confident about this area having above average fitness, and therefore generate individuals in it with a high probability.

3.5 Penalization

One of the most important aspects of building the mixture model is penalizing gaussians representing sub-average individuals, which amplifies the differences between the individuals that would otherwise be undistinguishable, it also prevents a scenario where are multiple sub-average individuals generated close to each other, their gaussians merge and create an impression of an above-average area. This can be observed happening in the center area of Fig. 3.1, while Fig. 3.2 shows how penalization solves the problem

Algorithm 2: SAOOA

```
input  : Number of individuals in one generation $N \in \mathbb{N}$,
         Number of generations kept in population $M \in \mathbb{N}$,
         Generation number $K \in \mathbb{N}$
output: $N$ new individuals

/* Initialization                                                    */
$new\_individuals \leftarrow$ sample N individuals from an initial gaussian
$K \leftarrow 1$

/* Endless loop of optimization                                      */
while $true$ do
    $population \leftarrow population \cup new\_individuals$
    $old\_individuals \leftarrow$ individuals older than $M$ generations
    $population \leftarrow population \setminus old\_individuals$

    /* Evaluate the individuals                                      */
    foreach $individual \in new\_individuals$ do
        $individual.fitness \leftarrow$ evaluate($individual$)
        $individual.K \leftarrow K$
    $average\_fitness[K] \leftarrow$ average_fitness($new\_individuals$)

    /* Update the standard deviation                                 */
    $\gamma \leftarrow$ recalculate_gamma($\gamma$)
    $\sigma \leftarrow \sigma_d * \gamma$

    /* Create the Gaussian mixture model                             */
    for $i \leftarrow 1$ to length($population$) do
        $amplitude \leftarrow population[i].fitness$
        $mean \leftarrow population[i].genome$
        $gaussian \leftarrow$ Gauss($amplitude$,$mean$,$\sigma$)

        /* Penalize the sub-average individuals                      */
        if $individual[i].fitness < average\_fitness[population[i].K]$ then
            $gaussian \leftarrow gaussian + c * (population[i].fitness -$
              $average\_fitness[population[i].K])$

        /* Weight the gaussians                                      */
        $weight \leftarrow \left(\frac{1}{2}\right)^{K - population[i].K}$
        $gaussians[i] \leftarrow gaussian * weight$
    $GMM \leftarrow \sum_{i=1}^{length(population)} gaussians[i]$

    /* Sample new individuals from the GMM                           */
    $new\_individuals \leftarrow$ sample($GMM$)

    $K \leftarrow K + 1$
```

and transforms the model into a much more relevant estimate of promising regions.

Penalization is done by adding $c*(f_{Ki}-f_K)$ to aforementioned gaussians. c is an experimentally found constant equal to 13, if it was smaller, all the gaussians would merge into a one large "hill" because the sub-average individuals would not be penalized enough. On the other hand, if it was bigger, the penalization would be so severe that it would turn the model into a one large "valley". Therefore the value 13 is a sought for balance between these two extremes.

f_{Ki} is a fitness of an i-th individual from generation K, and f_K is an average fitness of individuals from generation K.

Individuals are always compared only within one generation because each generation can be evaluated for a quite long period of time which can result in different fitness values than in previous generations thanks to the constant changes in the environment.

3.6 Updating standard deviation

Another important feature of the proposed algorithm is updating the standard deviation before creating the mixture model.

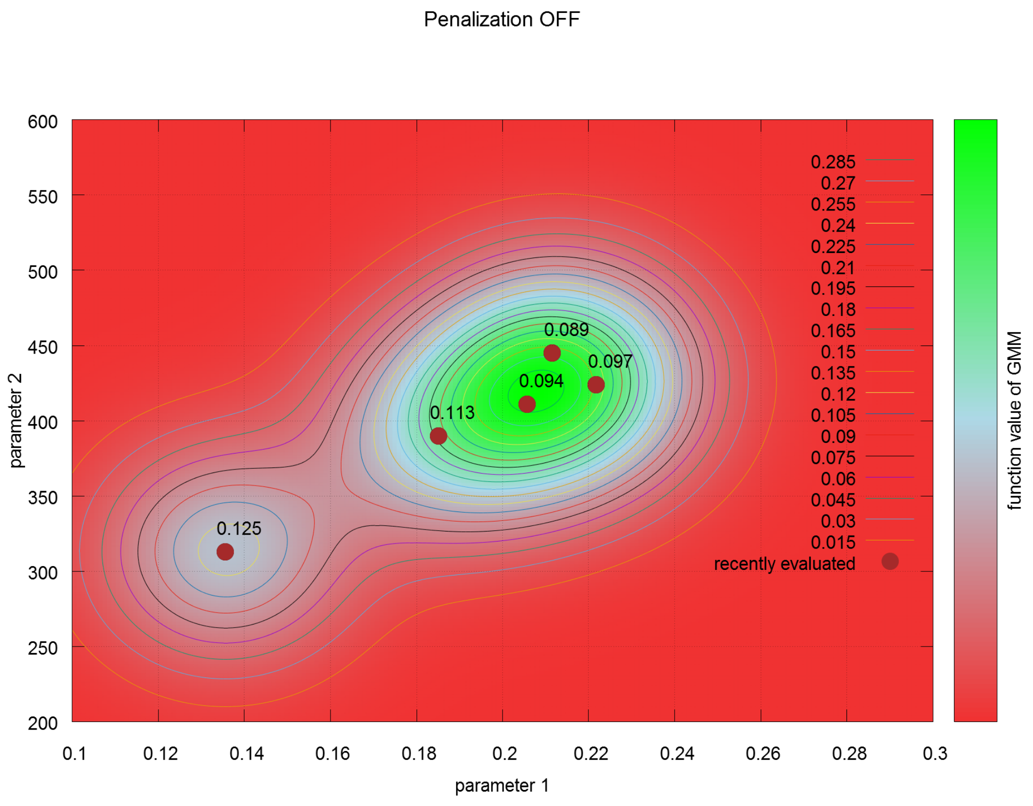
We generally do not want large changes of the individuals, because then we risk generating some horribly underperforming configurations, which is undesirable due to the nature of online A/B Testing. Therefore a default standard deviation is quite small, specifically a 1/100 of parameter's interval size.

In case we are not sure about the location of an initial area used to generate the first individuals, we are forced to make it significantly larger and then risk that the first population will be far away from optimum, making it very hard to reach with only small changes made possible by the default standard deviation.

Solution to this problem is a standard deviation multiplier γ allowing us to recalculate the standard deviation of all gaussians each generation as:

$$\sigma = \sigma_d * \gamma \tag{3.1}$$

σ_d is a default standard deviation set at the start of the algorithm. γ is initially set to a number large enough to allow the exploration of the whole space during the subsequent process of slowly reducing the multiplier to its base level.



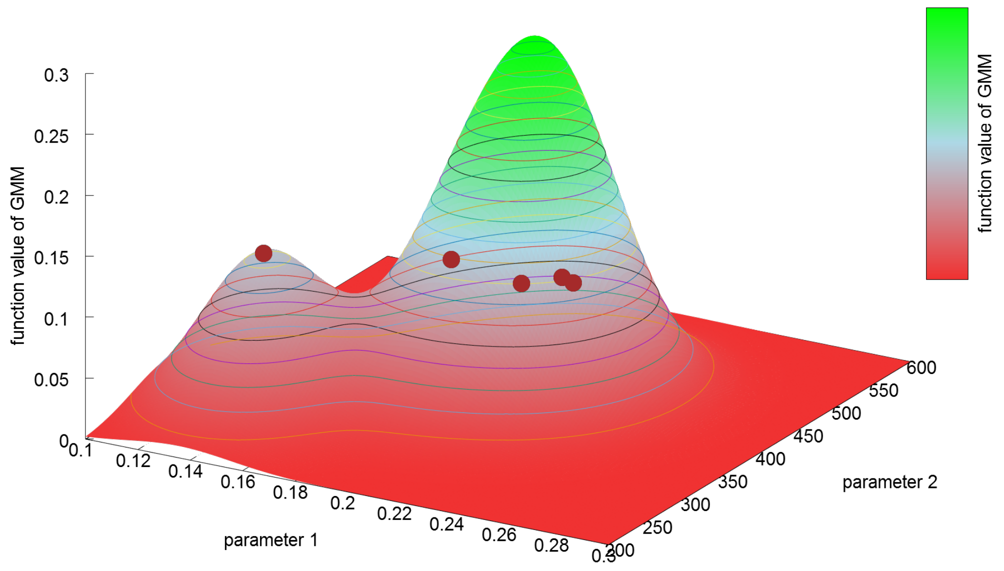
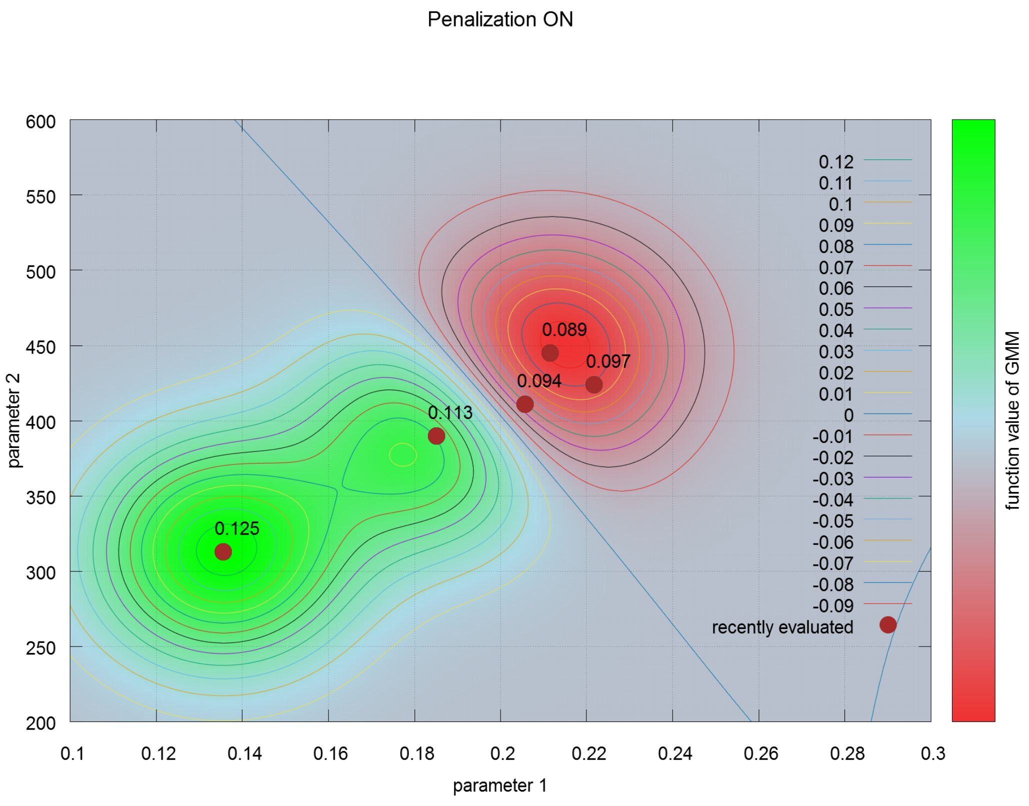


Figure 3.1: Multiple sub-par individuals creating an above-average region.



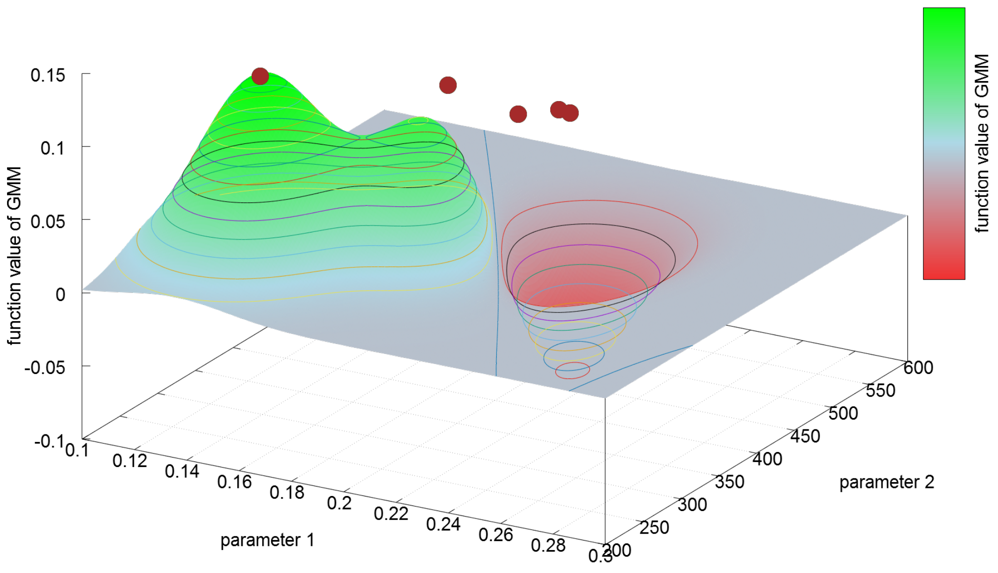


Figure 3.2: Solution to the problem in Fig. 3.1 by penalization.

Therefore a method inspired by simulated annealing is employed, reducing the exploration rate of the algorithm with an increasing number of passed generations.

Because of the dynamic environment, we cannot converge to a specific point and stay there, that is why we never decrease the multiplier past a certain base threshold, meaning that exploration always continues.

If we detect a significant change in the environment exceeding typical noise, we increase the multiplier to ensure that we do not get stuck in a suddenly sub-average area. In our case, we consider a change significant if the difference is more than 25%. Changes are detected by keeping the best individual from the previous generation and then comparing the two measurements of its fitness value. This approach is called *elitist selection* and we will call this particular individual an *elitist.*

Unfortunately, due to the large noise and complex environment, recognizing a true global change has proven to be difficult.

3.7 Implementation

The whole optimization algorithm has been implemented from scratch in Java 7 [46] as a multi-threaded application capable of optimizing several recommender systems at once. It is also capable of looking for the optimal settings of unlimited number of parameters at the same time, unfortunately the amount of user interactions is usually too small to optimize more than two parameters simultaneously.

As has been mentioned earlier in Section 3.1, the program interacts with the rest of the recommender system deployed for each of the customers only through the tables in their respective schema. Java Database Connectivity (JDBC) [47] is used to connect it with the central database and access the following tables:

- **ab_testing** Provides interface for evaluation of the selected configurations in form of two columns: A/B testing group and parameter settings to be evaluated.

- **recomms** Contains all information about recommendations including time, the recommended item, which configuration of the system generated it and other details.

- **conversions** or **actions** These tables store information about user purchases or other actions based on recommendations. Only one of those tables is used for evaluation of the configurations, which one depends on the chosen metric.

- **optim_data** The optimization program creates this table as soon as it is started and uses it to store information about the evaluated configurations.

Because the newly generated parameter settings are inserted into the **ab_testing** table in a JSON format [48], a library called Jackson [49] is used for manipulation with that format.

3.7.1 Visualization

The final program having the proposed algorithm in its core is capable of automatically generating different kinds of graphs depending on the amount of optimized parameters or whether should the mixture model be depicted. They are all being created in Gnuplot 5.0 [50].

After each evaluation of multiple configurations, some way of a graphical representation was needed to clearly show the tested parameter settings and how well they performed. A simple graph consisting of colored points proved to be the best variant. An example can be seen in Figure 3.3, color of a point represents fitness of the individual.

In order to transparently depict the selection of new configurations, a graph of the algorithm's mixture model was created. Due to the fact that optimization of two parameters is the most common, all graphs outside of this section depict 3 dimensional models. But the program is able to generate graphs for any number of parameters. From one, depicted in Figure 3.4, to many. The problem of displaying an $N > 3$ dimensional model is solved by creating $N - 1$ projections of two parameters similar to Figure 3.2 while the other parameters are set either automatically according to the best individual or to a manually specified values.

The X and Y axis stand for the parameters that are being optimized. Color represents the function value of the GMM which has a direct impact on the selection of new individuals. Because we generate a new graph after each evaluation we are most interested in the freshly evaluated individuals, that is why they are represented by contrasting brown points with their fitness written above them. The older individuals are depicted as black circles colored according to their fitness. The pictured GMM is always based on the visible points with the brown ones being most influential due to the weighting of older individuals. New individuals are generated according to the portrayed GMM and added to the graph as yellow squares. At that

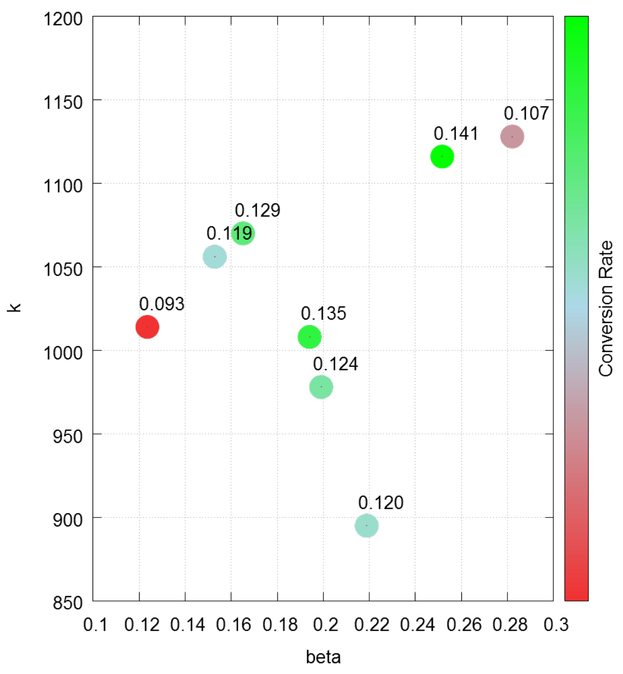


Figure 3.3: A simple graph showing all the evaluated configurations.

moment we do not know their fitness so we assign them one according to the GMM, this assigned fitness is only for the purpose of putting the squares somewhere in the graph, it bears no other significance what so ever.

Brown points therefore represent the same individuals as the yellow squares in the graph of a previous evaluation. And because of the elitist approach, the best individual from the current evaluation is also selected for the next evaluation and is therefore depicted as a brown point surrounded by a yellow square in a top down view of a three dimensional model.

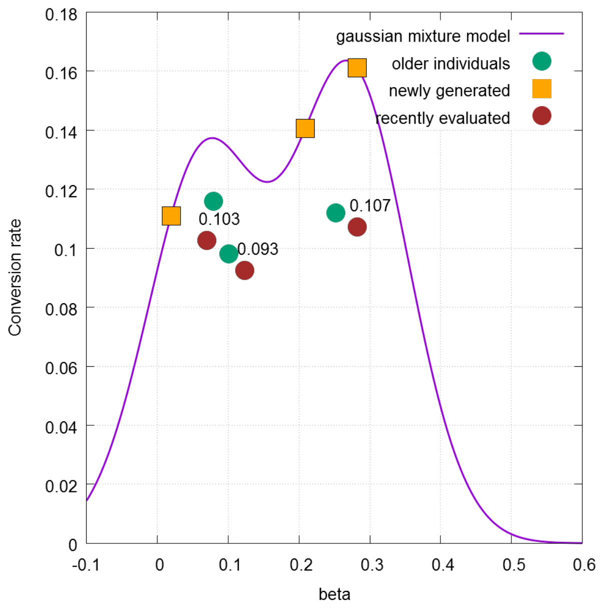


Figure 3.4: Graph of a mixture model while optimizing only one parameter.

Chapter 4

Experiments

This chapter contains both simulated and online experiments. The simulations are used to demonstrate the functionality of the algorithm and explain reasons behind some of the design decisions and their benefits. The online experiments prove that the algorithm is capable of optimizing real world recommender systems.

4.1 Simulated experiments

All simulations are inspired by situations encountered during online tests that were conducted while developing the algorithm. The three presented situations along with their combinations cover every possible problem that the optimization algorithm has to deal with.

The experiments are simulated because all the simulated environmental changes are happening at once in the real world and it would be complicated to show specific features of the algorithm. Therefore the actual data from online measurements are not used in order to provide a more transparent demonstration of how the algorithms works.

The first evaluation of a simulation is always based on data collected online. The algorithm then generated new individuals based on its mixture model and we manually assigned fitness values to those individuals instead of evaluating them online. This is the only place where we intervened with the optimization process.

Data presented in the simulations are based on an experiment in which two parameters: *beta* and k were optimized. The included graphs therefore show a top down view of the three dimensional mixture model.

4.1.1 Movement of the above-average area

This experiment simulates a slow movement of the above-average area that we have converged to. It is one of the most usual changes in the environment and the proposed algorithm is therefore designed to cope with it. It can be caused by the composition of users becoming slightly different over time and demanding a new configuration of the recommender system as a result.

The movement of this area can also happen very rapidly which is typically caused by a global change in the environment. This scenario is more thoroughly described in Section 4.1.2.

Each evaluation and its impact on the mixture model can be seen in Figure 4.1. The simulation starts with the algorithm already converged to an area positioned in a bottom left corner of the first image. This area is slowly getting worse while the above-average one is moving towards larger *beta* and *k*. We can see the adaptation of the SAOOA algorithm during the next evaluations which would continue till it reached an area surrounded by configurations with lower fitness.

This experiment demonstrates the basic ability of the algorithm to continuously search for the best configurations even after it converged to some specific area. It does so by never decreasing the standard deviation multiplier past a certain threshold and therefore never ending the optimization process.

4.1.2 Global fitness change

The environment in which is the optimization running undergoes a radical change from time to time, for example, web page displaying the recommended items is redesigned. These changes can be divided into two groups depending on how they impact the user perception of recommendations.

- The relative quality of all configurations stays the same and the only thing that changed is that their fitness was increased or decreased by a same amount. For example, the fitness can be globally decreased because recommendations are newly displayed at the bottom of the page instead of the top, making them harder to notice.
- Users now prefer different recommendations and the large changes in fitness of evaluated individuals are accompanied by the creation of new above-average areas. This scenario is actually very similar to the first one in a way how the algorithm reacts. Instead of returning to the previous location it simply converges towards a new region instead.

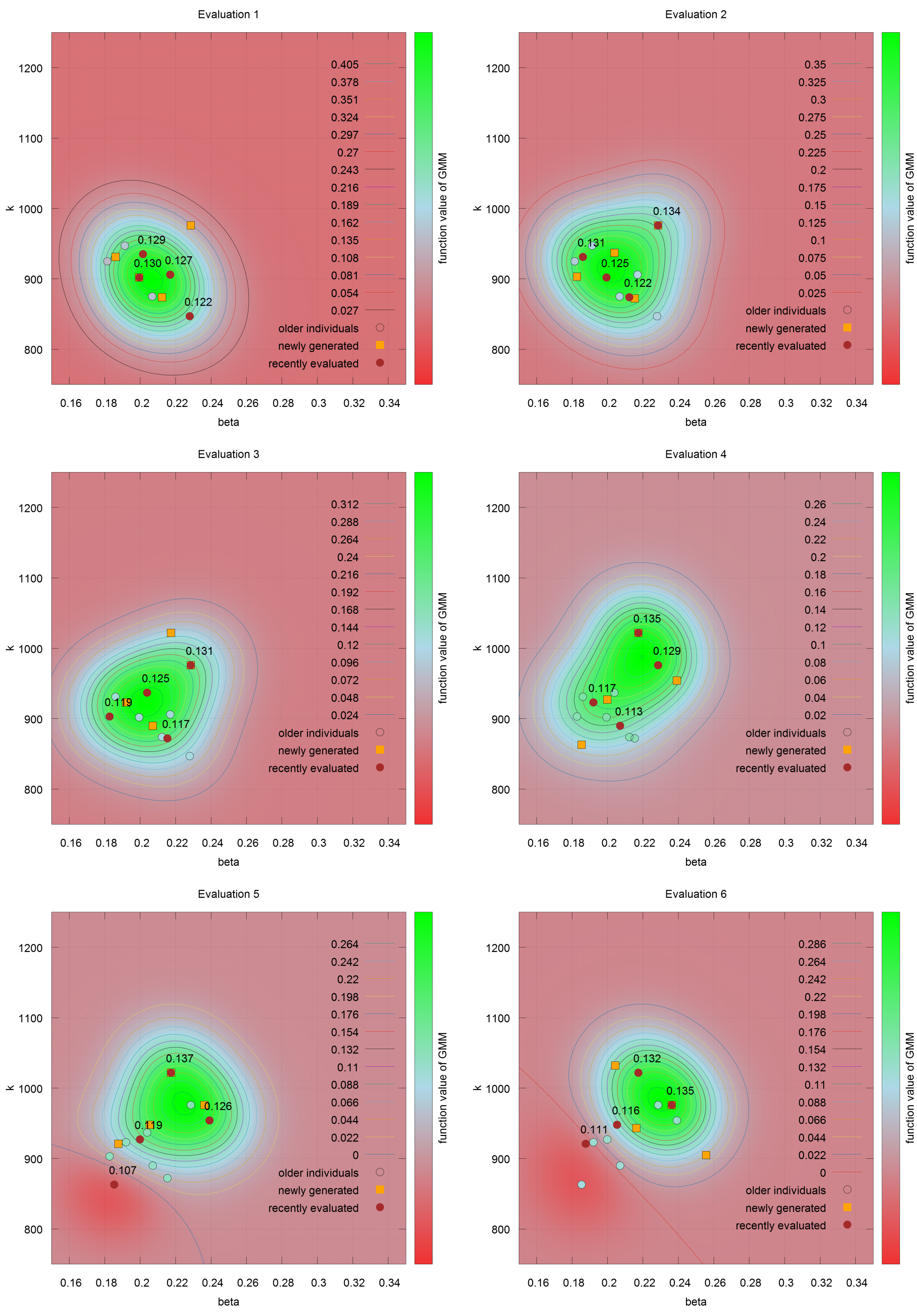


Figure 4.1: Simulation of a moving above-average area.

The first scenario is simulated in this experiment and its progress is depicted in Figure 4.2. The first image shows all the individuals concentrated in a small area. That means that the algorithm has already converged to this specific area and therefore has the lowest possible γ. Then, the second evaluation detects a large change in fitness of the elitist as a result of the global environment change. Multiplier is increased and newly generated individuals are scattered in a suddenly much larger area. During the next evaluations the γ is being linearly decreased and due to the fact that the relative fitness of different individuals has not changed, the algorithm converges to an area approximately same as the one at the beginning of the simulation. This can be clearly seen by comparing the location of individuals in the sixth evaluation with individuals from the first one.

This experiment provides an insight into how exactly does the SAOOA algorithm reacts to global changes in the environment and shows that it does not have a problem with returning to the previously found above-average area after such changes.

It also demonstrates why is the multiplier updated after detection of such large changes. Imagine an algorithm without this dynamically increasing exploration feature. If a new above-average region is formed far away from the current one as a consequence of the global changes, it may be impossible to find out in which direction it is due to the very low exploration rate which is necessary to keep all generated individuals in the already found area. Therefore, even though such an algorithm would be capable of converging to an above-average area, it would be unable to react to any substantial changes afterwards.

4.1.3 Unusually large noise

If the fitness of some individual i changes significantly due to an unusually large noise, two scenarios must be considered.

If the individual i is the current elitist, meaning that it is kept from the previous generation, then the algorithm detects the change and reacts by increasing the multiplier. This increases the exploration rate of the algorithm and allows it to reach the previously found above-average area again, unless disrupted by another very noisy measurements. Such a return to the previous location can be observed in Figure 4.2 which is explained in Section 4.1.2.

On the other hand, if the individual i is not the current elitist, which means that it has probably not been evaluated before, then the algorithm does not react in any special way, because it cannot compare the current

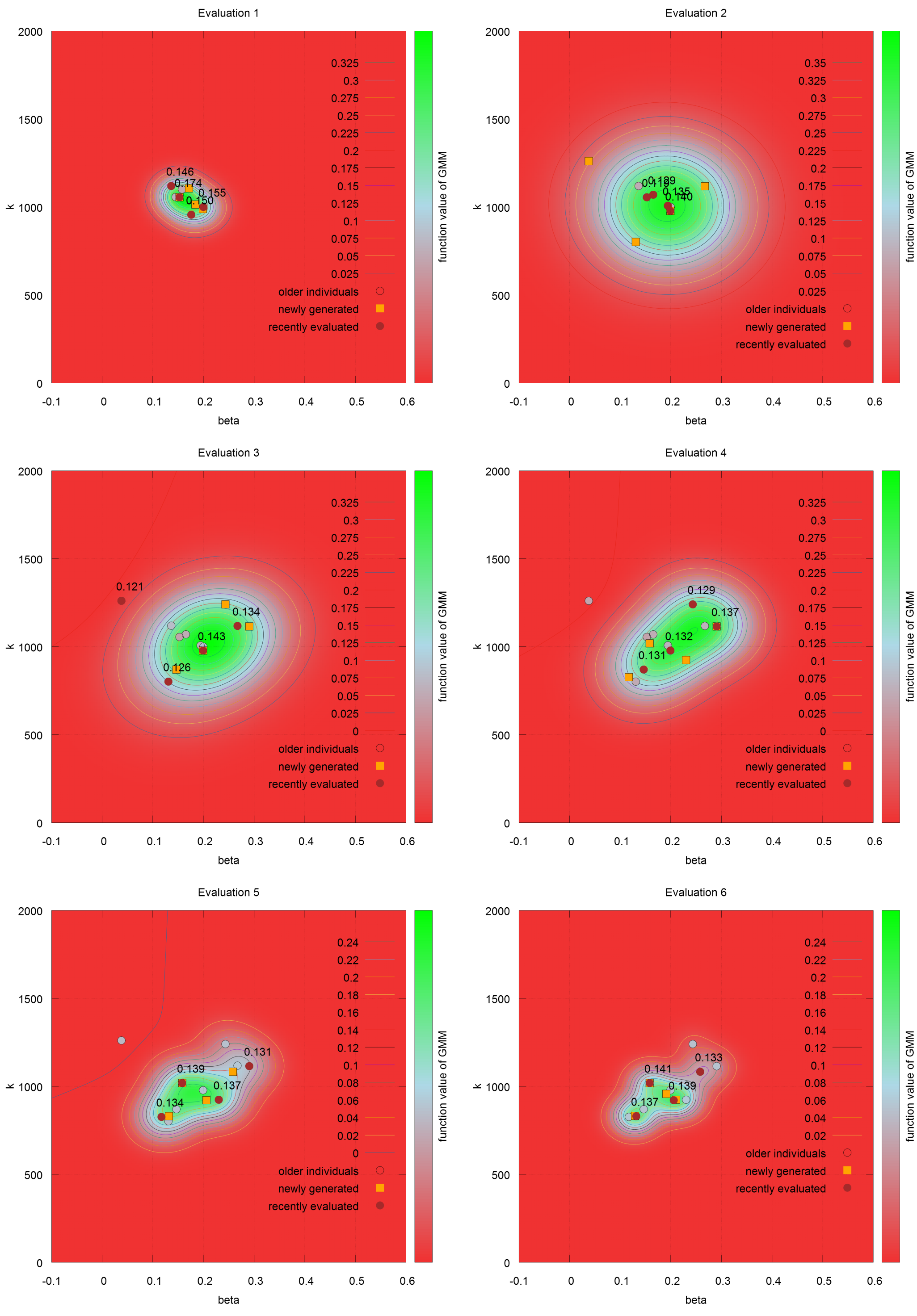


Figure 4.2: Simulation of global decrease of fitness affecting the evaluation of all individuals.

fitness value to any previous one. As a result, the noise affects only the formation of the mixture model.

Two cases are possible depending on how exactly the noise influenced the measured fitness.

- If it increased the fitness of the individual, it would very likely become the new elitist, which would mean that it would be evaluated again next generation. That evaluation would reveal that the previous fitness value was significantly different from the current one and therefore trigger an increase of the standard deviation multiplier. Unless the second measurement was also very noisy of course. After the increase of γ the algorithm would converge to an old above-average region or a newly formed one. How exactly does the convergence takes place is described in Section 4.1.2.
- Lower fitness of the individual would result in a shape of the mixture model causing new individuals to be generated away from the individual i. The fact that i is not the elitist means that there was an individual with higher fitness in the previous generation, which is now more likely to become the elitist again because one of the individuals that could have had the highest fitness out of this generation, has been negatively impacted by noise. As a result, even though the model was less accurate, new individuals are allowed to be generated from the so far above-average area because the knowledge about its location was carried from the previous generation through the elitist. A similar scenario is simulated in this experiment.

The progress of the simulation can be seen in Figure 4.3. The first evaluation of the simulation reveals unusually low fitness of the individual depicted in the top left corner. The algorithm reacts by generating new individuals away from the affected one. This trend continues through the second and third evaluation while the effect of the noisy measurement looses importance due to the weighting of gaussians representing old individuals. After three generations, the negatively affected individual is removed from the population and the algorithm starts exploring the top left corner again.

As a result, this experiment shows that the algorithm is capable of coping with extremely noisy measurements. Of course, if multiple following evaluations are heavily influenced by noise, it becomes impossible to effectively search for better configurations. Although we can reduce the noise by prolonging the evaluation time of recommendations, this approach have some drawbacks that are discussed more thoroughly in Section 4.2.

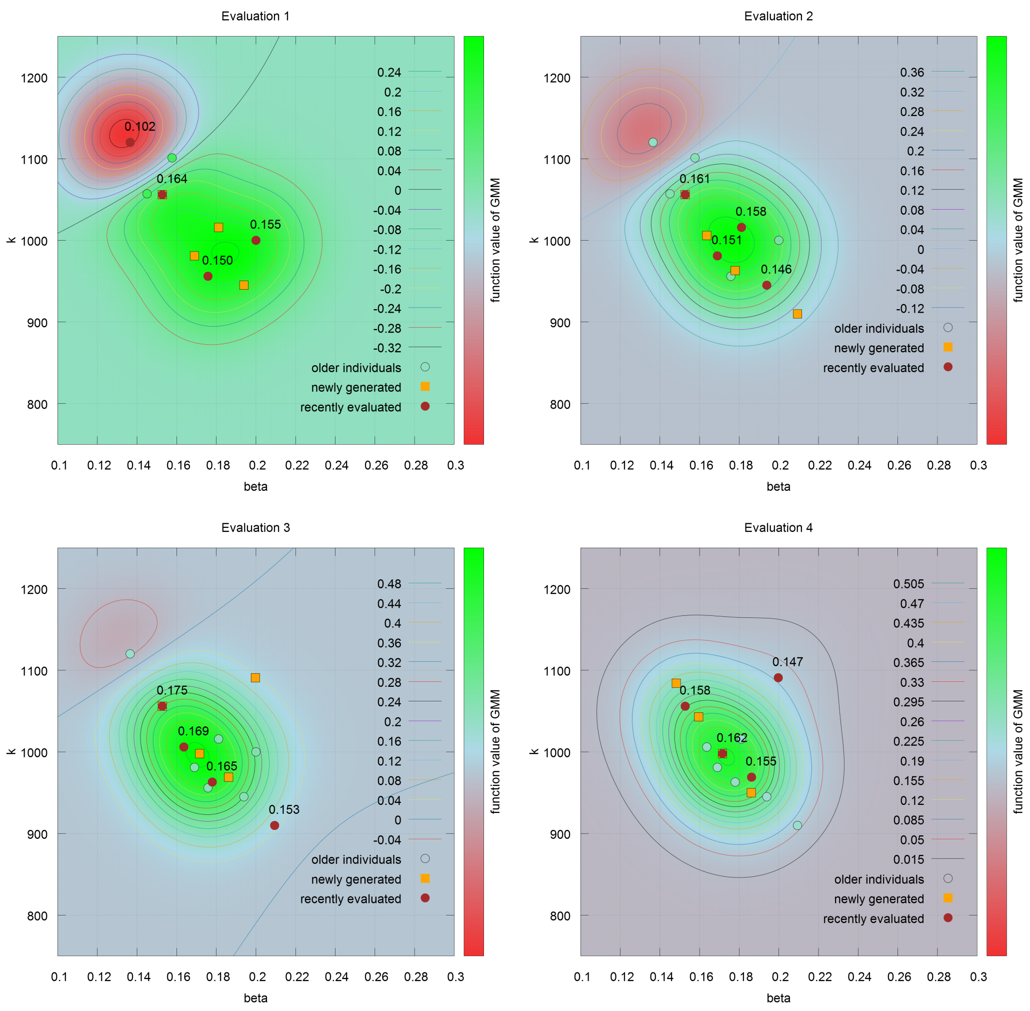


Figure 4.3: Simulation of unusually large noise affecting the evaluation of a single individual.

The experiment also shows positive effects of employing the elitist approach. It stabilizes the algorithm by making sure that all the new individuals are not going to be generated far away from the above-average area which would be possible due to the probabilistic nature of the process. It also makes it more conservative by evaluating a configuration that has already been tested online with good results. And lastly, it provides means to detect and quantify changes in the environment.

4.2 Online experiments

The algorithm has been tested on four different e-commerce websites during the development and has been improved to the current state in order to deal with the encountered problems. The final version of the algorithm as presented in Chapter 3 has been successfully tested on two websites. We will refer to them simply as *Website 1* and *Website 2*.

In case of the first one, we did not know the location of the optimal area and therefore started the algorithm with a very large standard deviation multiplier. After 15 generations, it has successfully converged to an area with a 20% higher conversion rate than the one it started in. In order to measure the difference, individuals in both areas were evaluated at the same time to rule out the effects of global changes happening between the beginning of the experiment and its end. To make sure that the measured increase in conversion rate was not caused by noise, we were evaluating the comparison generation for twice as long as the standard generations (96 hours).

We have already found out an area of very good parameter settings for the *Website 2* before deploying the optimization method, therefore we set the standard deviation multiplier to a level representing that the algorithm has already converged. It has successfully stayed in that area for over a month at the time of writing this, overcoming several global changes and noisy measurements during which it explored the surrounding areas thanks to the temporarily increased standard deviation multiplier.

The most typical problem that we had to deal with was a small number of user interactions usually resulting in a large noise. The only solution is to prolong the time of the evaluation of the tested configurations which allows to compensate for the lack of interactions. On the other hand, it significantly slows down the whole optimization process which may make it unable to locate the above-average areas or react to the global changes in a reasonable time.

Consider the following example: the successful experiment mentioned above, where the algorithm converged after 15 generations, had to evaluate each generation for 48 hours to obtain relevant measurements. It therefore took 30 days to find a sufficiently good area, which is still acceptable considering the low traffic and the fact that we could not narrow the search down at the beginning. If a single evaluation had to last a week, it would take months to converge, but due to global changes in the environment that usually happen each couple of months, it could take forever. As a result, the optimization algorithm does not work very well on websites with extremely low number of visitors. On the other hand, even only a thousand visitors a day create enough interactions for the algorithm to work without any problems.

It is important to note that due to the expert knowledge of the algorithms and provided information about the domain, we are in most cases able to pinpoint the initial location to an area significantly smaller than the whole configuration space. This in turn greatly reduces the time that the algorithm needs to converge. The *Website 1* has got approximately 10 000 interactions per day which represent the typical size of traffic that we had to deal with. For such websites, it usually takes around 5 evaluations to find an above average area starting from the initial one. That translates into 5 - 10 days of optimization depending on the amount of noise.

Longer evaluation time reduces the noise in all cases not just in the one with small number of interactions. This can be observed in Figures 4.4 and 4.5 that are showing results of measurements made online at *Website 1*.

The Figure 4.4 depicts fitness of five individuals evaluated for increasingly longer duration of time. It clearly shows that longer evaluation results in less noise and better reveals the underlying global changes. In this case, the fitness of all individuals is apparently globally decreasing.

The Figure 4.5 is based on the same data as the previous one. It shows for how many percent has the fitness changed from one evaluation to the next. We can see that the sudden changes can overcome 20% during the 24 hour long evaluations that are clearly very noisy. The measurements with increased duration minimizing the noise suggest, that the rate of global change is increasing. If it crosses a certain value (we have experimentally chosen 25%) the algorithm will react by increasing the standard multiplier.

Both figures show that the 24 hours long measurements are too noisy in case of the *Website 1*. On the other hand, increasing the time past 48 hours does not bring significant improvements in comparison with how much longer it then takes to converge. As a result the length of evaluation has been set at 48 hours in this specific case.

The online experiments have shown that the proposed algorithm is stable enough for a long term optimization and that it is capable of overcoming all sorts of problems encountered during online testing. The experiments have also revealed that the optimization of websites with approximately 10 000 interactions per day usually takes between five to ten days. And lastly, the reduction of noise through the length of evaluation has been discussed with the help of Figure 4.4 and 4.5.

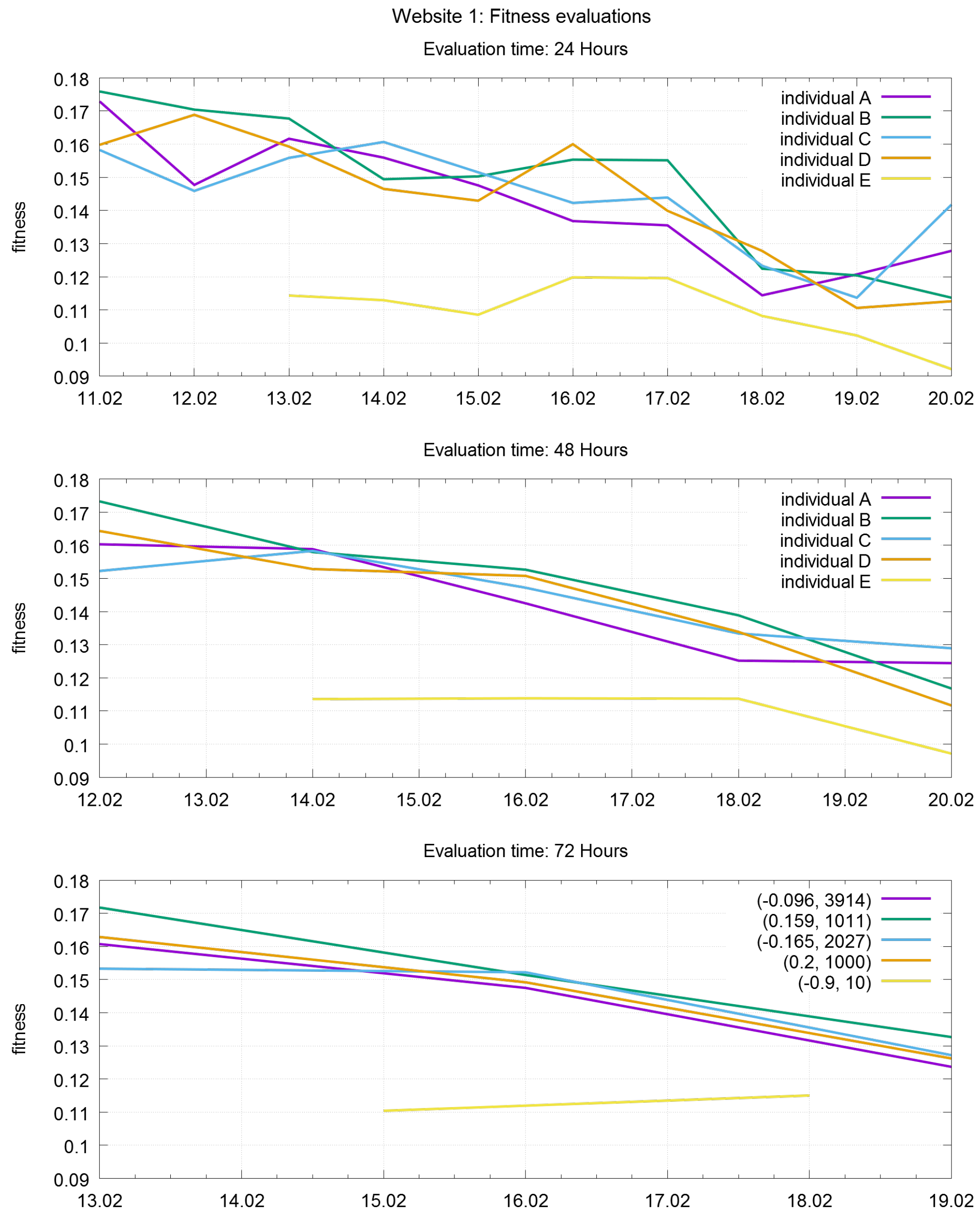


Figure 4.4: Fitness evaluations of five different individuals (recommender system configurations) at Website 1.

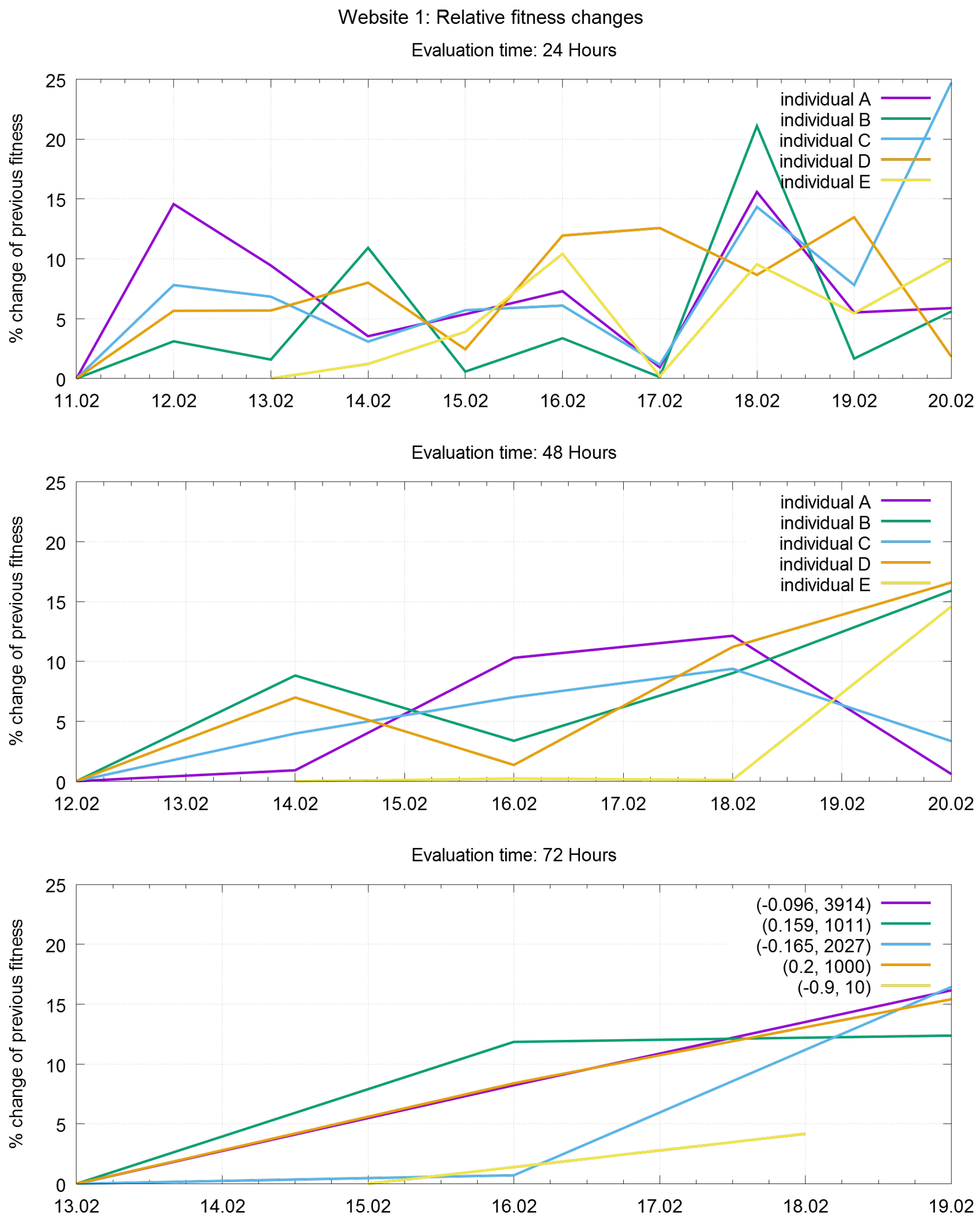


Figure 4.5: Fitness changes from one evaluation of a single individual to the next, expressed in percentage of the previous fitness. This figure is based on the same measurements as the Figure 4.4.

Conclusion

This thesis has described basic recommender system types and then proceeded to focus on their evaluation. It has been explained that to properly evaluate a recommender system two things have to be chosen. Firstly, an experimental settings, which can be selected from online, offline or a user study variants. And secondly, an optimization metric which is particularly hard to choose correctly because a large number of them exist and each one focuses on different aspects of the system. On the other hand, during online evaluations that we are most interested in, the chosen metric is typically simple and connected to the profit of the website.

Three most frequently used algorithms behind recommender systems and basic distinctions between online and offline optimization of these systems have been described in the rest of the Chapter 2.

The Chapter 3 has introduced the new SAOOA algorithm capable of online optimization of recommender systems while overcoming noisy measurements and changes in the environment. Along with how it works it has also described its implementation as well as implementation of the visualization methods used to depict the inner state of the algorithm or quality of different recommender configurations.

The capabilities of the proposed algorithm have been demonstrated on three simulations. First one has shown that it is able to follow small continuous changes in the environment. The second one has simulated a sudden change in quality of all configurations and the third one an unusually noisy evaluation of one individual. The last two simulations have demonstrated that the algorithm is capable of staying in the area it converged to in spite of global changes and noise.

The final version of the algorithm has also been successfully tested online on recommender systems deployed to two real world e-commerce websites, where it has proven its capabilities demonstrated during simulations.

In case of the first website, the algorithm has managed to converge to an area with a 20% higher conversion rate, where it has stayed making small adjustments. The second online experiment was different in that we have already known the location of the desired region of above-average configurations. The algorithm has successfully remained in that area for several weeks while occasionally testing the surroundings during adaptation to a number of changes in the environment.

At the time of writing this thesis, the algorithm was still running at several websites for over a month, proving that it is both stable and successful in online optimization of various recommendation algorithms and that it is therefore perfectly capable of improving the performance of recommender systems.

Future work

During our future work we plan to expand the optimization algorithm with ability to automatically determine the number of individuals evaluated each generation according to the size of user traffic. We would also like to add a possibility of comparing several recommendation algorithms online, automatically selecting the best one for the current domain and then further optimize it through its parameters.

Bibliography

[1] Sarwar, B.; Karypis, G.; Konstan, J.; et al. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM conference on Electronic commerce*, ACM, 2000, pp. 158–167.

[2] YouTube. `https://www.youtube.com/`, [Online; accessed 21-March-2016].

[3] Netflix. `https://www.netflix.com/cz/`, [Online; accessed 21-March-2016].

[4] Steam. `http://store.steampowered.com/`, [Online; accessed 21-March-2016].

[5] Resnick, P.; Varian, H. R. Recommender systems. *Communications of the ACM*, volume 40, no. 3, 1997: pp. 56–58.

[6] Burke, R. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, volume 12, no. 4, 2002: pp. 331–370.

[7] Ricci, F.; Rokach, L.; Shapira, B. *Introduction to recommender systems handbook*. Springer, 2011.

[8] Bennett, J.; Lanning, S. The Netflix Prize. In *KDD Cup and Workshop*, 2007.

[9] Melville, P.; Sindhwani, V. Recommender systems. In *Encyclopedia of machine learning*, Springer, 2011, pp. 829–838.

[10] Trewin, S. Knowledge-based recommender systems. *Encyclopedia of library and information science*, volume 69, no. Supplement 32, 2000: p. 180.

[11] Schafer, J. B.; Frankowski, D.; Herlocker, J.; et al. Collaborative filtering recommender systems. In *The adaptive web*, Springer, 2007, pp. 291–324.

[12] Schein, A. I.; Popescul, A.; Ungar, L. H.; et al. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2002, pp. 253–260.

[13] Lops, P.; De Gemmis, M.; Semeraro, G. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, Springer, 2011, pp. 73–105.

[14] Gunawardana, A.; Shani, G. A survey of accuracy evaluation metrics of recommendation tasks. *The Journal of Machine Learning Research*, volume 10, 2009: pp. 2935–2962.

[15] Gunawardana, A.; Shani, G. Evaluating Recommender Systems. In *Recommender Systems Handbook*, Springer, 2015, pp. 265–308.

[16] Kohavi, R.; Longbotham, R.; Sommerfield, D.; et al. Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, volume 18, no. 1, 2009: pp. 140–181.

[17] Peterson, E. T. *Web analytics demystified: a marketer's guide to understanding how your web site affects your business.* Celilo Group Media, 2004, ISBN 0974358428, 76–78 pp.

[18] Vermorel, J.; Mohri, M. Multi-armed bandit algorithms and empirical evaluation. In *Machine learning: ECML 2005*, Springer, 2005, pp. 437–448.

[19] Wu, W.; He, L.; Yang, J. Evaluating recommender systems. In *Digital Information Management (ICDIM), 2012 Seventh International Conference on Digital Information Management*, IEEE, 2012, pp. 56–61.

[20] Said, A. *Evaluating the accuracy and utility of recommender systems.* Dissertation thesis, Universitätsbibliothek der Technischen Universität Berlin, 2013.

[21] McNee, S. M.; Riedl, J.; Konstan, J. A. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, ACM, 2006, pp. 1097–1101.

[22] Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; et al. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, volume 22, no. 1, 2004: pp. 5–53.

[23] McNee, S. M.; Riedl, J.; Konstan, J. A. Making recommendations better: an analytic model for human-recommender interaction. In *CHI'06 extended abstracts on Human factors in computing systems*, ACM, 2006, pp. 1103–1108.

[24] del Olmo, F. H.; Gaudioso, E. Evaluation of recommender systems: A new approach. *Expert Systems with Applications*, volume 35, no. 3, 2008: pp. 790–804.

[25] Singhal, A. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, volume 24, no. 4, 2001: pp. 35–43.

[26] Hanley, J. A.; McNeil, B. J. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, volume 143, no. 1, 1982: pp. 29–36.

[27] Schafer, J. B.; Konstan, J.; Riedl, J. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, ACM, 1999, pp. 158–166.

[28] Breese, J. S.; Heckerman, D.; Kadie, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.

[29] Kohavi, R.; et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2, IJCAI, 1995, pp. 1137–1145.

[30] Duda, R. O.; Hart, P. E.; et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973, ISBN 0471223611.

[31] Stone, M. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, 1974: pp. 111–147.

[32] Chai, T.; Draxler, R. Root mean square error (RMSE) or mean absolute error (MAE). *Geoscientific Model Development Discussions*, volume 7, no. 1, 2014: pp. 1525–1534.

[33] Cremonesi, P.; Koren, Y.; Turrin, R. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, ACM, 2010, pp. 39–46.

[34] Sarwar, B.; Karypis, G.; Konstan, J.; et al. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, ACM, 2001, pp. 285–295.

[35] Agrawal, R.; Imieliński, T.; Swami, A. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, volume 22, no. 2, 1993: pp. 207–216.

[36] Zheng, Z.; Kohavi, R.; Mason, L. Real world performance of association rule algorithms. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2001, pp. 401–406.

[37] Byrd, R. H.; Hansen, S.; Nocedal, J.; et al. A stochastic quasi-Newton method for large-scale optimization. *arXiv preprint arXiv:1401.7020*, 2014.

[38] Bäck, T.; Schwefel, H.-P. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, volume 1, no. 1, 1993: pp. 1–23.

[39] Kirkpatrick, S.; Vecchi, M. P.; et al. Optimization by simmulated annealing. *science*, volume 220, no. 4598, 1983: pp. 671–680.

[40] Deb, K.; Pratap, A.; Agarwal, S.; et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions*, volume 6, no. 2, 2002: pp. 182–197.

[41] Kennedy, J. Particle swarm optimization. In *Encyclopedia of machine learning*, Springer, 2011, pp. 760–766.

[42] Razavi, S.; Tolson, B. A.; Burn, D. H. Review of surrogate modeling in water resources. *Water Resources Research*, volume 48, no. 7, 2012.

[43] Kleijnen, J. P. Kriging metamodeling in simulation: A review. *European Journal of Operational Research*, volume 192, no. 3, 2009: pp. 707–716.

[44] Beyer, H.-G.; Schwefel, H.-P. Evolution strategies–A comprehensive introduction. *Natural computing*, volume 1, no. 1, 2002: pp. 3–52.

[45] Recombee. `https://www.recombee.com`, [Online; accessed 21-March-2016].

[46] Java. `https://java.com/en/`, [Online; accessed 21-March-2016].

[47] JDBC. `http://www.oracle.com/technetwork/java/overview-141217.html`, [Online; accessed 21-March-2016].

[48] JSON. `http://www.json.org/`, [Online; accessed 21-March-2016].

[49] Jackson. `https://github.com/FasterXML/jackson`, [Online; accessed 21-March-2016].

[50] Gnuplot. `http://www.gnuplot.info/`, [Online; accessed 21-March-2016].

Appendix A

Acronyms

GMM	Gaussian Mixture Model
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
ES	Evolutionary Strategy

Appendix **B**

Contents of enclosed CD

readme.txt	the file with data disk contents description
src	the directory of source codes
└ implementation	implementation sources
└ thesis	the directory of LaTeX source codes of the thesis
text	the thesis text directory
└ BP_Bartyzal_Radek_2016.pdf	the thesis text in PDF format