VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

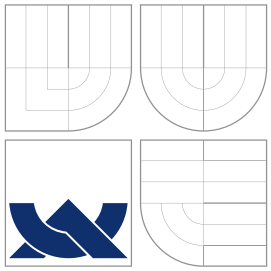
VISIPEDIA: MULTI-DIMENSIONAL OBJECT EMBEDDING BASED ON PERCEPTUAL SIMILARITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

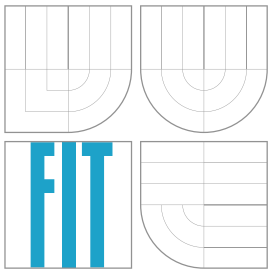
AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ MATERA

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VISIPEDIA: MULTI-DIMENSIONAL OBJECT EMBEDDING BASED ON PERCEPTUAL SIMILARITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ MATERA

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Dr. Ing. PAVEL ZEMČÍK

BRNO 2014

Abstrakt

Problémy jako je jemnozrnná kategorizace či výpočty s využitím lidských zdrojů se v posledních letech v komunitě stávají stále populárnějšími, což dosvědčuje i značné množství publikací na tato témata. Zatímco většina těchto prací využívá "klasických" obrazových příznaků extrahovaných počítačem, tato se zaměřuje především na percepční vlastnosti, které nemohou být snadno zachyceny počítači a vyžadují zapojení lidí do procesu sběru dat. Práce zkoumá možnosti levného a efektivního získávání percepčních podobností od uživatelů rovněž ve vztahu ke škálovatelnosti. Dále vyhodnocuje několik relevantních experimentů a představuje metody zlepšující efektivitu sběru dat. Jsou zde také shrnuty a porovnány metody učení multidimenzionálního indexování a prohledávání tohoto prostoru. Získané výsledky jsou následně užity v komplexním experimentu vyhodnoceném na datasetu obrázků jídel. Procedura začíná získáváním podobností od uživatelů, pokračuje vytvořením multidimenzionálního prostoru jídel a končí prohledáváním tohoto prostoru.

Abstract

Some problems like fine-grained categorization or human-based computation has become popular in recent years in the community, which has been proven by a large number of published works concerning these topics. Whereas most of these works uses a "classical" visual features extracted by machine, this one in partricular focuses on perceptual properties which cannot be easily sampled by machine and which involves humans into this data retrieval process. There are examined ways, how to obtain perceptual similarities from humans cheaply and effectively also in terms of scalability. There are performed various experiments and purposed several methods to improve this efficiency. The work also reviews and compares existing methods of embedding learning and navigating through its space. The acquired observations are subsequently used in a complex experiment evaluated with a food image dataset, covering the whole procedure from similarity retrieval from humans, over data embedding learning up to searching in such multi-dimensional space.

Klíčová slova

Visipedia, jemnozrnná kategorizace, percepční podobnost, získávání podobností, crowdsourcing, MTurk, multidimenzionální indexování dat, mental matching, výpočty s využitím lidských zdrojů

Keywords

Visipedia, fine-grained categorization, perceptual similarity, similarity retrieval, crowdsourcing, MTurk, multi-dimensional data embedding, mental matching, human-based computation

Citace

Tomáš Matera: Visipedia: Multi-dimensional Object Embedding Based on Perceptual Similarity, diplomová práce, Brno, FIT VUT v Brně, 2014

Visipedia: Multi-dimensional Object Embedding Based on Perceptual Similarity

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana profesora Pavla Zemčíka. Další informace mi poskytl můj konzultant, profesor Serge Belongie. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Matera
28. května 2014

Poděkování

Na tomto místě bych chtěl poděkoval lidem, kteří mi radili, pomáhali a motivovali mě při psaní diplomové práce. V první řadě děkuji svému konzultantovi, profesoru Sergeovi Belongiemu, za vřelé přijetí na University of California v San Diegu, uvedení do problematiky práce, o které jsem měl předtím jen okrajové znalosti, cenné rady a trpělivost během mé celé desetiměsíční stáže. Dále bych chěl poděkovat svému vedoucímu práce, profesoru Pavlu Zemčíkovi, za organizaci zahraničního výjezdu a věcné připomínky k práci samotné. Poděkování patří také všem členům Computer Vision Labu na UC San Diego a Caltechu za nápady a připomínky k mé práci.

Acknowledgments

At this place I would like to thank to people who advised, helped, and motivated me during the work on my Master's thesis. Firstly, I thank to my adviser, professor Serge Belongie, for a warm welcome at University of California in San Diego, introduction to the field of the thesis, in which I had before just peripheral knowledge, valuable advices and his patience during my whole ten-month internship. I also thank to my supervisor, professor Pavel Zemčík, for the arrangement of the internship and for substantive comments to the work itself. My thanks also go to all members of Computer Vision Lab at UC San Diego and Caltech for their ideas and comments to my work.

Contents

Chapter 1

Introduction

Human beings have five traditional senses giving them an opportunity to distinguish things, which the machines cannot. Although the machines are nowadays able to process audiovisual information at a decent level, they do not cope with properties as a taste or a smell. But if they did, it would open the door to a number of new approaches how the machines could be helpful and useful for humans.

As an example we may consider food. People are able to distinguish between different types of food nearly perfectly on the basis of their taste, and they group meals which taste similar together somewhere inside the brain. If such human is familiar with tastes of multiple types of food, this grouping process creates a virtual distribution of different types of food in the human's brain.

On the opposite side, there are machines, which are nowadays able to process the visual information of the meal, but they do not obtain any information about the taste without human assistance. However, if they are somehow given this information, they could model the distribution of different types of food in a similar way as the human brain does. This step of the information exchange is not trivial and it requires some investigation in methods, how to do this exchange effectively, which is the subject of this work.

In this thesis I examine ways how to compare objects on the basis of their perceptual similarity in order to obtain the structural information among them. This structural information is subsequently transformed to a generally multi-dimensional space, in which more similar objects should be placed close to each other whereas the less similar ones far apart. Such space then serves as a guideline for the searching algorithm, which also involves humans to the searching process. Since the cooperation with humans is in this system widely used, the particular focus is placed to efficiency and low-cost solution of the interaction with them.

The aim of the research, developed software, and conclusions described in this thesis was to extend the Visipedia project, which is introduced in Chapter 2. In Chapter 3 there is introduced crowdsourcing, benefits of its usage, its types, and on-line services, which were also used for some experiments. Chapter 4 is dedicated to algorithms for embedding learning and methods of navigating through the space of embedding. In Chapter 5 there are defined some fundamental terms used in this work and there are presented approaches of getting objects similarities from users. Experiments on a toy dataset of US and Canadian cities, dataset of country flags, and a food image dataset respectively are described in the chapters 6, 7, and 8. The summary of achieved results and possible directions of a future work are presented in Chapter 9.

Chapter 2

About the Visipedia Project

The idea of the project Visipedia dates back to the year 2009, when Pietro Perona presented his thoughts in the paper *Vision of a Visipedia* [16]. As Wikipedia is based on text articles with connections to another related articles, the aim of Visipedia is to be an analogy focused towards the images. Although the word Visipedia stands for "Visual Encyclopedia", it is not literally an encyclopedia but rather a layer on the top of Wikipedia (or generally any other knowledge database).

This chapter presents reasons which led to this project. It briefly introduces the Visipedia project reviews it's architecture and points out some remarkable and unique features of this system.

2.1 Motivation

Imagine this example: You see a mushroom during a stroll and the questions like "Can I eat it?", "Should I pick it up?", come into your mind. If you are not familiar with mushrooms, you will not recognize the species of the mushroom you see and therefore you do not know if you can eat it. If you started to browse the web pages about mushrooms on your cell phone in order to find the particular species, it could take a long time. On the other hand, if there was such a system as Visipedia, you could grab a picture of the mushroom, upload it to the Visipedia system as a query, and hopefully you would be redirected to the Wikipedia page of the mushroom species you are looking at.

This is just a simple example, but using just image information is not always easy or possible for several reasons: The state of the art of computer vision and machine learning does not allow to reliably classify objects on a picture and also there may be another features that cannot be extracted directly from an image. Either can be hidden or not capable to be captured by camera (e.g. volume information, smell, hardness, etc.). Although there are some methods how to improve the amount of captured information (supplying video sequence instead of a single picture, using stereo cameras, structured light, or using other detectors), for some tasks the system still requires cooperation with humans.

2.2 Visipedia Concept and Design

Such a project as Visipedia that would simplify searching, indexing and linking the parts of images cannot be done by individual or a small team. It rather involves a cooperation of

[1] Image taken from [16].

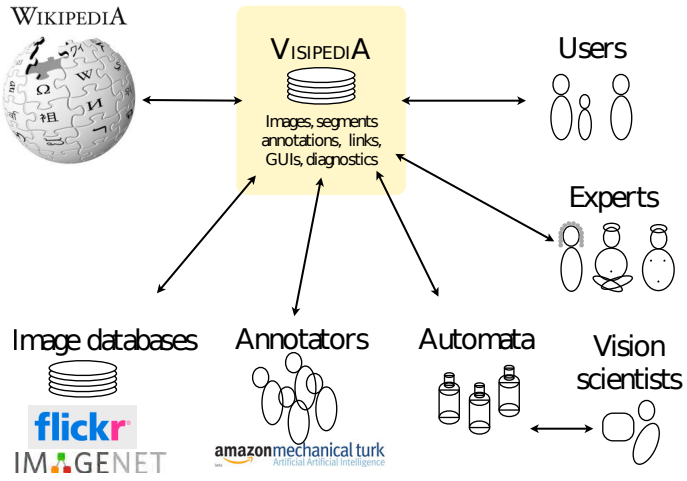


Figure 2.1: The Visipedia system connects together users sending queries to the system, pages like Wikipedia as a source of information, available image databases for training data, human domain experts providing knowledge, crowd workers for annotation tasks and computer vision and machine learning systems. [1]

many experts from different fields. That is the reason why Visipedia has been designed to be open to public. Anyone will be able to participate in this project by creating software augmenting the functionality, uploading and annotating images, etc. in order to improve the level of automation as much as possible and hence save a human labor.

As previously mentioned, it is not yet possible to build automata that would carry out all this job, therefore the concept of the system proposes interaction with 5 groups of people: The *users* benefiting it while looking for a useful information using queries, *domain experts* willing to share results of their research and providing basic knowledge, non-expert *editors* helping with data cleaning, *crowd workers* used for annotation and other mass work and *automation experts* providing computer vision and machine learning support. The concept of Visipedia system is summarized in Figure 2.1. The approach that combines automation and human labor is called "humans-in-the-loop".

To make such idea working, it is necessary to process each input image and gather as much information as possible from it. Since the system works also with humans providing needful information which can be again used by automata, the image processing has to be performed iteratively. Then the default Visipedia pipeline consists of the following steps:

1. Image upload,
2. automatic image processing – saliency detection, meaningful feature measurement, and decomposition to sub-tasks,
3. distribution of sub-tasks to appropriate system parts (automatas or human resources),
4. collection of the results from sub-tasks and its processing
5. go to #2.

Besides human resources and automata parts the system is designed to be connected with Wikipedia and other similar knowledge databases, that can be used as a source of

information and also for example Wikipedia pages can be associated with corresponding images or its parts. This would allow Visipedia users to access the information they are looking for quickly and directly from their supplied image.

2.3 Relation to the Thesis

As I pointed out, Visipedia is the image oriented project. It performs a wide range of different tasks with images like segmentation, classification, annotation, etc.

Taking into account classification part, Visipedia does not aim only to classification or clustering into groups, but there is also an effort to support fine-grained categorization, where the objects are not just a part of a particular group, but rather they are placed into a multi-dimensional space according to their similarity.

"Classification" of non-taxonomically related objects is a continuous function, that assigns coordinates of multidimensional space to the input objects, is called *embedding*. Position of objects in embedding is determined by their similarity: more similar objects are located closer together in the embedding. In the case of this thesis a special focus is placed on perceptual similarities.

Chapter 3

Crowdsourcing

Crowdsourcing is a technique how to distribute some work to a large amount of generally anonymous workers in order to obtain and merge their contributions. This practice has been used especially in on-line community. The word "crowdsourcing" arose in 2005 as a combination of words "crowd" and "outsourcing". The main idea is to divide a complex task into several subtasks that often repetitive and time-consuming. The individuals working on such tasks as volunteers or part-time workers are usually rewarded for their work and hence this cooperation is mutually beneficial.

Definition. An integrating definition of crowdsourcing was developed by Enrique Estellés-Arolas and Fernando González Ladrón-de-Guevara [7]:

> "Crowdsourcing is a type of participative on-line activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken".

This chapter summarizes benefits and various use cases of crowdsourcing. There are also presented existing applications in this field, which were used for some experiments in this work. This chapter also summarizes ways and benefits of crowdsourcing usage in machine vision.

3.1 Crowdsourcing Types

Crowdsourcing has already been developing for several years and there has emerged various direction of its exploitation. Some of the most important and expanded are presented in the following list:

- *Crowdfunding* is the collection of finance to support projects or realize ideas. Contributors are people who usually provides a small amount of money voluntarily or in

exchange for some benefits from such project The goal for requesters is to reach a target amount [3].

- *Crowdvoting* is a type, where the "crowd" is asked about a judgment or an opinion on some topic. This method is usually used to organize or rank some content as photos, articles, etc.
- *Language-related data collection* has been used for collecting translations for dictionaries or to refine translations in services as Google Translator.
- *"Wisdom of the crowd"* is a process where multiple individuals are asked for an opinion rather then a single expert. The collection of answers is very often followed by their aggregation and processing to a final result. It is advantageous in many cases, because the answers of the crowd are usually as good as (and often even better than) the answer of the best individual from the group [26].
- *Makrowork* is a type of crowdsourcing where the workers are called to do some more complex work, which can require special skills. It can be for example some independent part of a large project or some specialized task.
- *Microwork* is a type where a requester divide the complex problem into a large number of simple, repetitive tasks, that also unskilled workers are able to work on them. It usually takes a couple of seconds or minutes to solve the task and hence these tasks are low payed.
- *Implicit* crowdsourcing can be represented by a software that serves primarily to a different purpose (at least from the crowd's point of view), but on the background it collects information about users' actions and profits from it. Some computer games or ReCAPTCHA [19, 20, 21] are examples of such software.

3.2 Reward

The important part of the crowdsourcing is to motivate workers to work on the crowdsourced tasks. Crowdsourcing services can be divided on the basis of the type how the workers are rewarded. There are some approaches which have been put into operation and which are interesting for a certain group of people.

Entertainment. The typical example of this category are computer games, which collects some useful data depending on users' actions while enjoying the game. The first example of this design was ESP game [19] originally aimed to image labeling. Games of this type are generally called *games with a purpose* (GWAP) [20].

Altruism and citizen science. Both of those cases are very similar, especially from crowdsourcing point of view. People participating in these tasks are self-motivated for whatever reason to work on (sometimes challenging) tasks in order to "help a good thing". In particular, citizen science is a scientific research conducted by amateur, enthusiast scientists. The interesting fields for amateurs are for example ornithology, astronomy or modern technology [22].

Financial reward. A complementary approach to the voluntary ones is a financial motivation of workers. Since workers are in this case paid for their contributions, it is necessary to watch over the quality of their work, otherwise the answer gathering process becomes expensive and ineffective. However, several ways how to assure quality are discussed in Section 3.5.

3.3 On-line Tools and Services

Since the main domain of crowdsourcing is on-line community, there are several web-based services with different level of generality and different specialization offering a crowdsourcing solutions. Furthermore, this work is focused on perceptual objects similarities, so some experiments described in this work are based on humans' responses. They fit to category *microwork* and therefore they take advantage of crowdsourcing.

Amazon Mechanical Turk (MTurk) is a well-known platform for task crowdsourcing with hundreds of thousands workers on demand, which also provides an API for faster deployment and management. Rights therefore all crowdsourced tasks in this works were deployed on MTurk. Since the financial reward is only way how to reward workers on MTurk, a part of the thesis aims to explore effective ways of answer retrieval from workers. Another software used in this work is a SaaS (software as a service) application called *Visipedia: Crowdwork*, which cooperates with MTurk, and which provides an interface for simple tasks creation and management. Both of these applications are described more in detail in the following sections.

3.3.1 Amazon Mechanical Turk

Amazon Mechanical Turk (MTurk)[1] is a marketplace for crowdsourcing tasks, that allows individuals or businesses to outsource tasks, that computers are unable to do, to human workers. Although MTurk falls according to its main specialization to category *mikrowork*, its API makes it a very universal platform, that allows developers to use it flexibly according to their requirements.

MTurk is also one of the biggest on-line marketplace. Indeed, hundreds of thousands HITs are available on MTurk at any time and there were registered more than 500 000 workers from over 190 countries in January 2011. Using some monitoring and quality assurance techniques is can be also consider as a source of inexpensive and fair-quality data from the crowd [15, 5]. MTurk recognizes two groups of participants: *Requesters* and *workers*. A default unit of work (a task deployed to MTurk) is called *Human Intelligence Task* (HIT).

Requesters are individuals or businesses who create and deploy HITs to the marketplace in order to let workers the solve them. Requesters also specify the wage, which a worker receive when she complete the HIT, and Amazon collects 10% commission on top of the specified reward. Requesters can also specify the number of assignments for each HIT (number of redundant HITs displayed to different workers), which allows synthesis of opinions and which is also useful for quality assurance. Another possibility how to assure high quality of responses is to apply qualification criteria. Only workers who are qualified for certain HITs, may start to work on them. MTurk offers a couple of pre-prepared qualification criteria like country of residence, minimal number of completed HITs, percentage

[1] https://www.mturk.com/

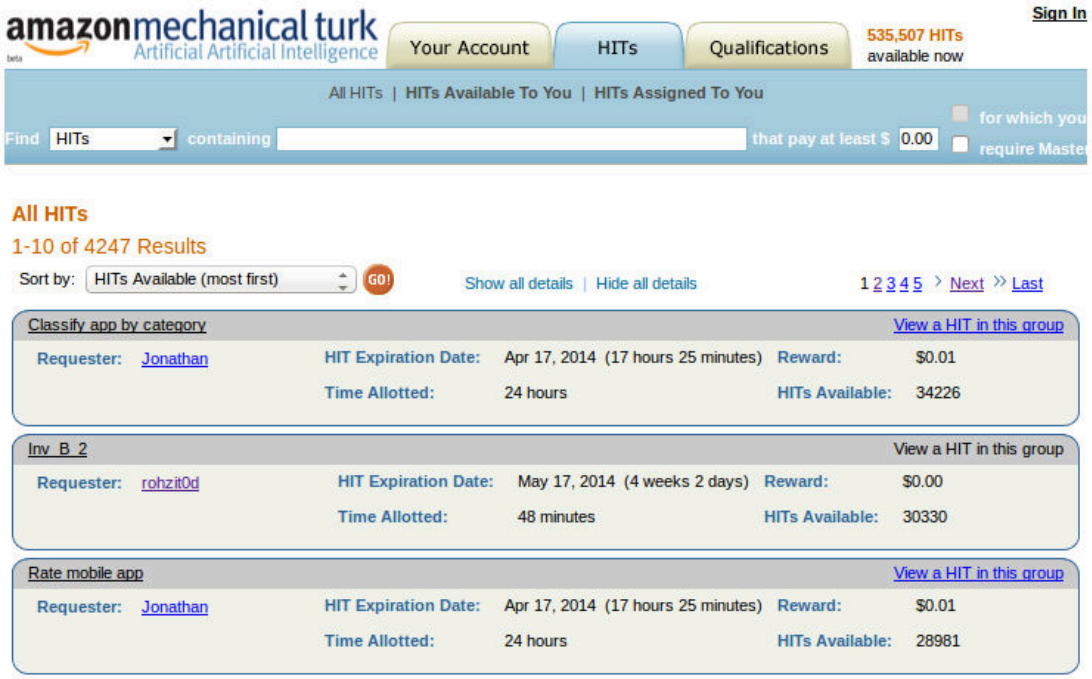


Figure 3.1: Screenshot of MTurk workers interface, where they can browse, preview, and accept HITs.

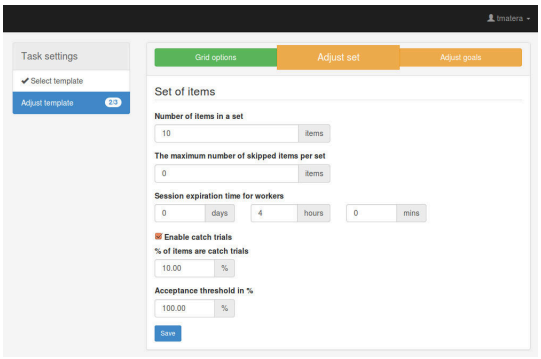
of accepted HITs etc., but it is also possible to create a custom qualification criteria. There are two possibilities for requesters how to deploy their tasks on MTurk: Using MTurk wizard or using MTurk API. In the first case a requester can simply create a HIT using MTurk website and supplying HTML code of the interface presented to workers. This way is probably in most cases easier then using API, but it is not so flexible, because all HITs have to be created manually, whereas using API allows system developers to deploy tasks to MTurk automatically. There are several, either official or unofficial SDKs for different programming languages as PHP, Python, Ruby, Perl etc. [2]

Workers are able to browse the HITs and work on them in case they meet the qualification requirements. When they decide to accept a HIT, a HIT assignment is allocated for them for a specific time period, within which they have to submit the HIT. If they decide not to complete the HIT they can return it and it is offered to other workers. For any HIT which they complete and which is accepted by the requester, they obtain the specified reward. A screenshot of MTurk workers interface is in Figure 3.1.

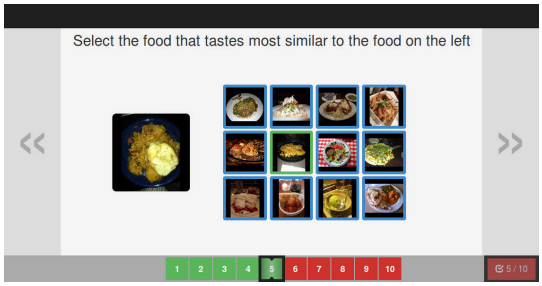
MTurk offers two modes of their system: Production and Sandbox mode. Production mode is the one where workers complete tasks and get paid for their work, whereas Sandbox mode is a testing environment – a copy of the production system, where no charges are applied. It serves developers to test there their software using MTurk API, and requesters and workers to familiarize with the interface.

3.3.2 Visipedia: Crowdwork

Collecting contributions from crowd workers is not as fast as it could could be. It often entails setting up server-side database and software, creating user interface, assuring workers competency, etc. All these time-consuming subtasks are usually carried out over and over



(a) Requester environment



(b) Worker environment

Figure 3.2: Screenshots from *Visipedia: Crowdwork* application showing environments for requesters and for workers.

again when it is necessary to crowdsource any new type of task, although the majority of such subtasks is each time almost identical. This was an inspiration to create application *Visipedia: Crowdwork* [14], that makes all these steps easier. The advantage of being incorporated into Visipedia system is that it uses unified Visipedia account management and the image resource service.

This crowdsourcing task manager is designed as Software as a Service (SaaS) running on top of Amazon Web Services and it allows requesters to set up their tasks and deploy them to crowdsourcing services as MTurk in "a few clicks". The idea of this manager was to make as universal and reusable as possible the whole process of task setup and to allow users to extend the functionality by implementing their interface.

The manager is divided into two separate parts: *Requester environment* and *Worker environment*. The requester environment consists of setting form where she can select and adjust a template (a layout which will be displayed to workers) specify task goals as number of answers or a time period, create catch trials and deploy the tasks to MTurk. The requester environment also shows statistics about workers activity and supports browsing and exporting the contributions. The worker environment displays the selected template to workers and allows them to solve the task. It also handles navigation between screens and implements extra functions such timers or onboarding tutorials. Screenshots from these two environments are displayed in Figure 3.2.

3.4 Cost Estimation

When deploying some task to a crowdsourcing service with financial reward, it is always necessary to determine the reward for a work unit (HIT) and estimate total costs. According to several forums[2] [3] [4], the optimal reward, when workers consider HITs as "worth turking for", is between $4-6 per hour.

Another interesting fact discovered authors of [13] when they figured out then higher reward increases quantity but has minimal impact to quality of work. In other words the

[2] http://www.reddit.com/r/HITsWorthTurkingFor/

[3] http://turkernation.com/showthread.php?8027-Must-read-for-turkers!-Guideline-for-requester-pay

[4] http://www.mturkgrind.com/forums/8-Hits-Worth-Turking-For

time necessary to solve certain number of HITs decreases with higher reward but the quality of work remains nearly constant.

3.5 Quality Assurance

Since the workers working on crowdsourced tasks are generally anonymous and such task is not assigned to some particular worker, where the requester would know her capabilities, and as well there are also potentially bad workers still around trying just to earn some money regardless of the quality of their work, it is necessary to assure, or at least evaluate the relevance of workers' contributions. Although this section is mainly focused on several approaches how the quality of answers can be evaluated or measured, there are also mentioned ways of preliminary workers selection.

Qualifications. Some crowdsourcing services (e.g. MTurk) offer system of qualifications to filter out ineligible workers before they start to work on tasks. Workers can be classified on the basis of country residence, number of submitted tasks, its approval rate, etc. Such qualification systems are often specific for particular crowdsourcing marketplaces, however, it is very advantageous to use them.

Redundant answers. Instead of assigning a particular task to just one workers, it is assigned to multiple workers and then their answers are somehow synthesized. Depending on the particular use case it can be taken the average from the answers, the highest-voted answer or the answer selected by a reviewer. The drawback of this method is the costs growth in direct proportion to the number of redundant tasks.

Repetitions. Repetitions are appropriate especially in tasks composed of a larger number of small subtasks of the same type. The principle of repetitions is to present to a worker multiple times some of these subtask and subsequently compare her answers. Although such subtasks do not have to look exactly the same, it is essential to ask the worker for the same thing multiple times. Basically, it is a measurement of intra-class variation of redundant answers.

Catch trials. Catch trials, also known as *Gold standard*, are intentionally created subtasks, where the answer is indisputable and defined a priori. Once a worker submits her answers for a catch trial, the quality of her answer can be determined by comparison with the predefined one. As well as repetitions, catch trials are also beneficial primarily in tasks consisting of a several subtasks.

3.6 Crowdsourcing in Machine Vision

Many tasks and solutions in machine learning and computer vision involve training and testing on large annotated datasets of various type. Basically, these tasks fall into two groups according to the employment of workers. The tasks from the first group are completed by workers "off-line", whereas in the case of the second one ("on-line"), the human workers are involved directly in the system.

Annotation tasks. In many cases the annotation tasks of datasets does not require domain experts but rather a large number of workers who might be also unskilled in the field. As examples of such tasks can be mentioned image segmentation, written text transcription, labeling, etc. The usage of crowdsourcing is in these cases very convenient, advantageous, and sometimes almost only way how to gather sufficient amount of annotations for a reasonable price. ImageNet [6], which is a database of more than 11 million images hierarchically organized, and CUB-200 [24] of 6000 birds of 200 species are examples of datasets, where the crowdworkers were used to annotate images by segmentation, bounding boxes and binary attribute annotation.

Systems with humans-in-the-loop. A different approach, how the crowdworkers can be useful, is to involve them directly in the pipeline of a computer vision system [22]. Workers can be employed for different stages of the pipeline, for example during model learning, classification, detection, feature extraction etc. Then the system is continuously processing their contributions on a basis of which it adapts its future behavior. This type of system architecture is a fundamental pillar of whole Visipedia project.

Chapter 4

Data Embedding

Data embedding is a traditional problem in many fields including mathematics, machine learning, data mining, etc. In the field of machine learning, it falls into the *Manifold learning* category, which represents a group of unsupervised or semi-supervised methods aiming to reduce the dimensionality of the data preserving the important features. In particular, it is an approach to non-linear dimensionality reduction. Dimensionality reduction is often required in terms of machine learning, either for intuitive data visualization or for reduction of memory and computational requirements [11].

In the first part of this chapter there is defined the term *embedding* and there are reviewed existing algorithms for data embedding construction. These methods are compared among themselves as well as in the terms of perceptually similar data modeling. In the second part of the chapter there is reviewed an existing method, which allows navigation and searching in the space of embedding.

Although data embedding comes from the effort of dimensionality reduction with minimization of error, the dimensionality reduction is not always necessary. In other words, data embedding is generally a projection of input objects into d-dimensional space. Traditional approaches used in machine learning such as linear discriminant analysis (LDA) or principal component analysis (PCA) are special types of data embedding.

Definition. Given a set of inputs $Z = \{z_1, \ldots, z_n\}$ and the number of dimensions d, embedding is the map $Z \to \mathbb{R}^d$. Specifically, we define embedding as a matrix

$$\mathbf{E} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{pmatrix}, \tag{4.1}$$

such that i^{th} row in the matrix $\mathbf{E}$ corresponds to the vector of coordinates of input z_i in the embedding.

4.1 Techniques to Learn Data Embedding

Lot of research has been done and a several algorithms has been proposed in the field of data embedding. There are two main approaches which these algorithms follow:

- Top-down approach is based on a distance (dissimilarity) matrix of input objects regardless of their dimensionality. Methods based on this approach try to place objects into multidimensional Euclidean space with respect to the constraints arising from the distance matrix. It does mean that those methods try to (relatively) preserve distances by minimizing the total error.

- Bottom-up approach uses information of local neighborhood of input objects and consolidates it in order to derive the global structure. There are often used local coordinates, distances, or weighted linear combination of surrounding points as a source of local information.

There are presented algorithms using either of these approaches.

4.1.1 Multi Dimensional Scaling (MDS)

Also known as Euclidean embedding, is a classical metric embedding method which has been used as a technique for analysis of data similarity or dissimilarity on a set of objects. It is a process of visualization of the given distance matrix. MDS algorithm places each object from the set into d-dimensional space, where the number of dimensions d is specified a priori [4].

Input of this method is a distance matrix $\mathbf{\Delta}$ of set of objects $Z = \{z_1, \ldots, z_n\}$, on which a distance function is defined as $\delta_{i,j} :=$ distance between objects z_i and z_j.

$$\mathbf{\Delta} = \begin{pmatrix} \delta_{1,1} & \delta_{1,2} & \cdots & \delta_{1,n} \\ \delta_{2,1} & \delta_{2,2} & \cdots & \delta_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n,1} & \delta_{n,2} & \cdots & \delta_{n,n} \end{pmatrix} \tag{4.2}$$

is subject to $\delta_{i,i} = 0$, $\delta_{i,j} = \delta_{j,i}$.

The goal of the algorithm, given a matrix $\mathbf{\Delta}$ and a number of dimension d, is to find n vectors

$$\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d, \text{ such that } \forall i, j \in 1, \ldots, n : \|\mathbf{x}_i - \mathbf{x}_j\|_2 \approx \delta_{i,j}. \tag{4.3}$$

One possible and also quite common way how to determine these vectors, is to formulate it as an optimization problem. For example

$$\min_{\mathbf{x}_1,\ldots,\mathbf{x}_n} \sum_{i<j} (\|\mathbf{x}_i - \mathbf{x}_j\|_2 - \delta_{i,j})^2. \tag{4.4}$$

This algorithm does not allow neither infinite nor missing distances $\delta_{i,j}$, which makes this method not appropriate in applications, where the distance magnitudes are not available, unreliable or too difficult to measure.

4.1.2 Non-Metric MDS (NMDS)

This non-metric modification of MDS tries to break away from distance magnitudes and it uses only a provided set of order relations [1]. Such formulation leads to the problem also called *Shepard-Kruskal Scaling*. Given a distance matrix $\mathbf{\Delta}$ and a number of dimension d, find vectors

$$\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d, \text{ such that } \forall i, j, k, l : \|\mathbf{x}_i - \mathbf{x}_j\|_2 < \|\mathbf{x}_k - \mathbf{x}_l\|_2 \iff \delta_{i,j} < \delta_{k,l}. \tag{4.5}$$

The algorithm that solves *Shepard-Kruskal Scaling* problem is based on minimization of the stress-1 functional

$$\sigma_1(\mathbf{E}) = \min_{\theta} \frac{\sum_{i,j}(\|\mathbf{x}_i - \mathbf{x}_j\|_2 - \theta(\delta_{i,j}))^2}{\sum_{i,j}\|\mathbf{x}_i - \mathbf{x}_j\|_2}, \tag{4.6}$$

where $\theta(\cdot)$ is an arbitrary monotonic function. The minimization is performed with respect to the embedding $\mathbf{E}$.

Although this method might seem to be more useful for a data with unknown distance magnitude, there are some issues which make it barely usable: The method requires all order comparisons, which can be sometimes really difficult if not impossible to provide. Although the NMDS concerns just about ordinal information, it still needs a distance matrix $\mathbf{\Delta}$ on its input. The process of perceptual properties retrieval cannot meet these requirements easily, therefore also this method is not very suitable.

4.1.3 Generalized Non-Metric MDS (GNMDS)

This algorithm was developed to get rid off completely the dependency on the distance matrix and it uses just the paired comparisons instead [1]. That was formulated to the problem called *Paired Comparisons*, which is derived from the *Shepard-Kruskal Scaling* problem.

Given a set of quadruples S, find $\mathbf{E} = \mathbf{x}_1, \ldots, \mathbf{x}_n | \, \mathbf{x}_i \in \mathbb{R}^d$ such that

$$(i,j,k,l) \in S \iff \|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq \|\mathbf{x}_k - \mathbf{x}_l\|_2. \tag{4.7}$$

Algorithm review. The algorithm solving the previously mentioned problem has been proposed and described in [1]. This is just its brief review. Let S to be a set of quadruples (i,j,k,l). The algorithm aims to find an embedding $\mathbf{E} = \mathbf{x}_1, \ldots, \mathbf{x}_n$ such that

$$\|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq \|\mathbf{x}_k - \mathbf{x}_l\|_2, \ \forall (i,j,k,l) \in S \tag{4.8}$$

The algorithm finds a Gram matrix $\mathbf{K} = \mathbf{E}^{\mathrm{T}}\mathbf{E}$ and tries to minimize its rank by trace-norm minimizing. The advantage of this method is that it accepts inconsistencies in the set of input paired comparisons. In order to allow inequality violations, it introduces a slack variable $\xi_{i,j,k,l}$ for each inequality constrain and the objective of algorithms solving this problem is to minimize the total amount of slack. In order to accommodate the algorithm to users demanding low-dimensional embedding, there has been added the regularizer λ has which trades-off the embedding complexity with the total slack. The above results to the program

$$\min_{\mathbf{K}, \xi_{i,j,k,l}} \sum_{(i,j,k,l) \in S} \xi_{i,j,k,l} + \lambda \mathrm{Trace}(\mathbf{K}) \tag{4.9}$$

subject to $k_{k,k} - 2k_{k,l} + k_{l,l} - k_{i,i} + 2k_{i,j} - k_{j,j} \geq 1 - \xi_{i,j,k,l}, \sum_{ab} k_{a,b} = 0, \mathbf{K} \succeq 0$.

4.1.4 Crowd Kernel Learning (CKL)

The motivation to introduce CKL system was an effort to make available data embedding and learning algorithm deployment on a specific domain without assistance of machine learning researcher. Given a set of triplets, the CKL algorithm learns a similarity matrix

over all n^2 pairs [17]. It introduces probabilities that are inversely proportional to the quality of triplet modeling [18]:

$$p_{i,j,l} = \frac{k_{i,i} + k_{j,j} - 2k_{i,j} + \mu}{(k_{i,i} + k_{j,j} - 2k_{i,j}) + (k_{i,i} + k_{l,l} - 2k_{i,l}) + 2\mu}, \tag{4.10}$$

where μ servers as a regularizer preventing numerical problems. The kernel is learned by empirical log-loss minimization:

$$\min_{\mathbf{K}} \sum_{(i,j,l)\in\mathcal{T}} \log(p_{i,j,l}) \text{ subject to } \forall i : k_{i,i} = 1, \mathbf{K} \succeq 0. \tag{4.11}$$

The gradient descent method is used for CKL learning and the resulting embedding is obtain by singular value decomposition of kernel $\mathbf{K}$.

Besides the algorithm for embedding construction, this system also introduces a method of adaptive triplet selection, which uses a history of user's answers on presented triplets and selects the most informative triplet, that maximizes information gain to be presented to the user. Since this process become computationally expensive for a larger datasets, the approximation consists of the selection of the best candidate from a randomly sampled subset.

4.1.5 Stochastic Triplet Embedding (STE)

Stochastic Triplet Embedding is another method for embedding construction, which is more local than previous ones. More specifically, it gives nearly constant rewards to triplets that are satisfied with a large margin and nearly constant penalties to large triplet violations [18].

The methods defines probabilities

$$p_{i,j,l} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2)}{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2) + \exp(-\|\mathbf{x}_i - \mathbf{x}_l\|_2^2)} \tag{4.12}$$

which measure the probability that the triplet (i, j, l) is satisfied. Given a set of training triplets T, the program aims to maximize log-probabilities over all supplied triplets:

$$\max_{\mathbf{E}} \sum_{\forall(i,j,l)\in\mathsf{T}} \log p_{i,j,l}. \tag{4.13}$$

Such program is a convex optimization problem and can be solved by gradient descent or by singular value decomposition using trace-norm regularizer to minimize the rank of the kernel matrix.

t-Distributed STE (t-STE). This modification of classical STE has been proposed due to a too rapid decline of gradient, that makes hard to fix errors made in the beginning of the optimization process. This led authors of [18] to propose another, heavy-tailed Student-t kernel with α degrees of freedom. Then the triplet probability is defined as

$$p_{i,j,l} = \frac{\left(1 + \frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\alpha}\right)^{-\frac{\alpha+1}{2}}}{\left(1 + \frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\alpha}\right)^{-\frac{\alpha+1}{2}} + \left(1 + \frac{\|\mathbf{x}_i - \mathbf{x}_l\|_2^2}{\alpha}\right)^{-\frac{\alpha+1}{2}}}. \tag{4.14}$$

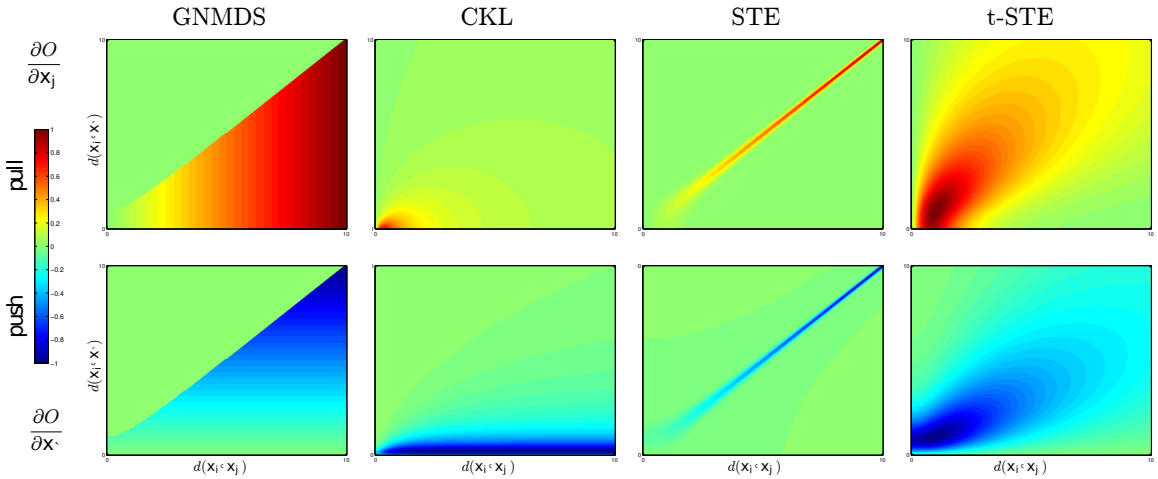


Figure 4.1: Partial gradients induced by a triplet constraint for methods GNMDS, CKL, STE a t-STE. Given the triplet (i, j, l) of input points z_i, z_j, z_l, on every image the *short edge* distance $\|z_i - z_j\|$ is on axis x and the *long edge* distance $\|z_i - z_l\|$ on axis y. The top-left region indicates a constraint satisfaction $\|z_i - z_j\| \ll \|z_i - z_l\|$, the bottom-right region indicates a strong constraint violation $\|z_i - z_j\| \gg \|z_i - z_l\|$. The bottom-left to top-right diagonal indicates the equality between *short edge* and *long edge* $\|z_i - z_j\| \approx \|z_i - z_l\|$. The top row shows the rate (the redder the higher) of pulling z_l apart from z_i and the bottom row shows the rate (the bluer the higher) of pushing z_j towards z_i. [1]

Using such heavy-tailed function as Student-t kernel is more advantageous than "standard" kernels. Given a triplet (i, j, l), t-STE decreases distances between $\mathbf{x}_i$ and $\mathbf{x}_j$ and analogically increases distances between $\mathbf{x}_i$ and $\mathbf{x}_l$ even if the constraint is already satisfied. The result of such a behavior is that it collapses points unless there is a triplet keeping them apart. Similarly it separates points unless there is a triplet keeping this points together.

4.1.6 Comparison of Previously Reviewed Methods

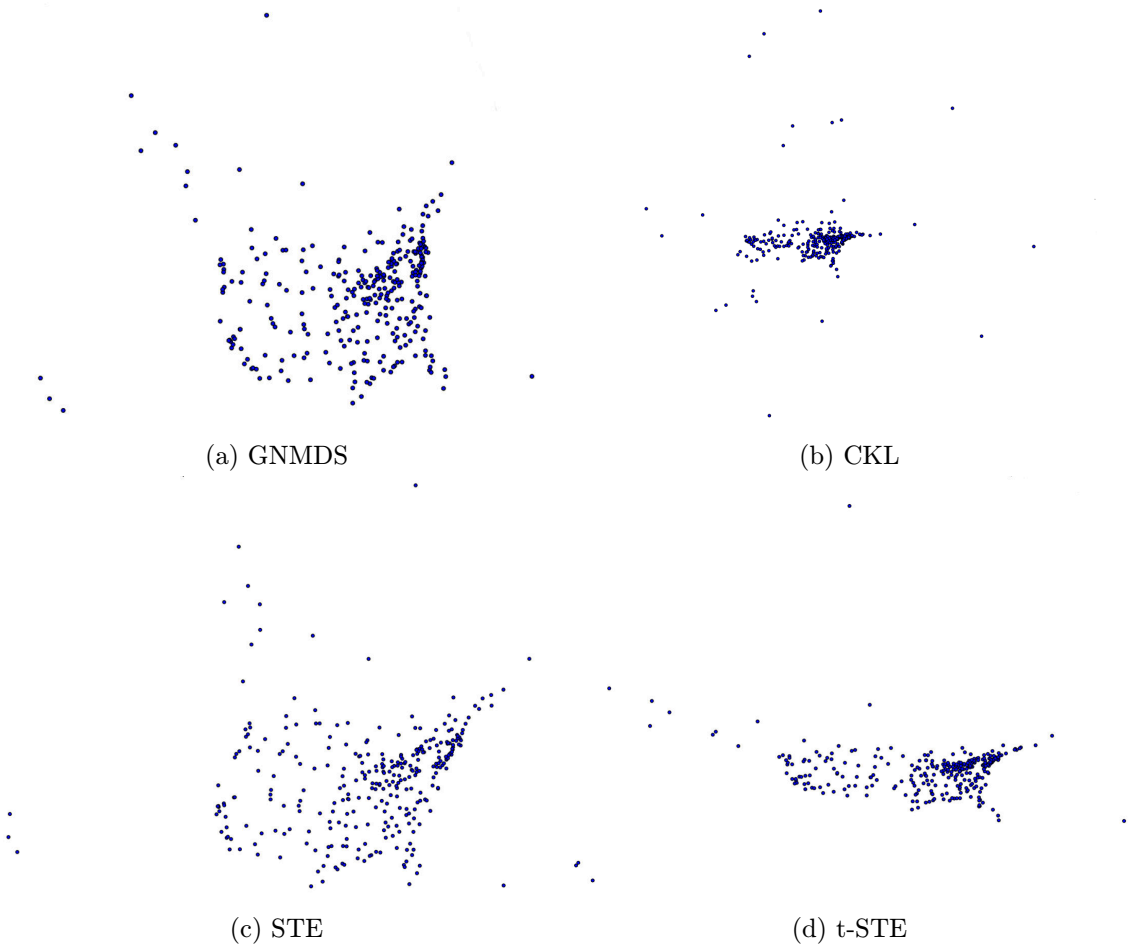
In previous section there were reviewed common algorithms used to build data embeddings. Each method has different properties, advantages, and disadvantages, and is suitable for different input data. This section is focused on comparison of those methods according to different criteria. At the end of this section, there are selected methods, which are potentially useful to work with perceptually similar data.

There can be observed two groups of methods according to data required on their inputs. First group is formed by methods MDS and NMDS that requires a complete matrix $\mathbf{\Delta}$ of pairwise distances between objects from the input set. (Although NMDS uses just the ordinal information.) The second group consists of GNMDS, CKL, (t-)STE requiring on the input a set of paired distance comparisons. In particular, all these methods accept on its input a set of triplets $\mathcal{T}$.

If we consider that the input data are perceptually similar objects, where the exact distances between object cannot be neither measured nor exactly determined, we can exclude the first group (MDS and NMDS) from future considerations.

The next comparison shows how the particular methods move the points inside embedding when a triplet is presented. If we look to Figure 4.1, we can see that different methods

[1] Image taken from [18].



(a) GNMDS (b) CKL

(c) STE (d) t-STE

Figure 4.2: Comparison of embeddings created by different methods.

put similar objects together or pull apart in different ways. If we look in detail to each method, we can observe this behavior:

GNMDS. This method moves the points just when the triplet constraint is violated and ignores cases when the triplet constraint is already satisfied. The gradient is linear along the axes in lower right triangle, which is why the method does not care much about the number of *short* or *long edges* between two particular points and it just tries to enforce triplet constraints. This leads to a quite uniform distribution of points in the embedding.

CKL. CKL method suffers a similar problem as GNMDS, however, the gradient decrease is not so rapid. Also the gradient is large only when a triplet is strongly violated. This means that CKL is concerned with correcting only strongly violated triplets.

STE. In contrast to CKL, STE gradient converges to zero for both strong constraint violations and strong constraint satisfaction. This implies that it does not tend to correct strong violations and thus it is resistant to triplets contradicting the consensus. On the other hand, the gradient decreases too rapidly, which makes hard for the method to correct errors made in the beginning of the optimization.

t-STE. Unlike the previous methods, t-STE gradient looks different and it has several good properties. The gradient is large even for already satisfied triplets, so given an already satisfied triplet, it tends to collapse points on the *short edge* and to separate points on the *long edge*. Another quality of t-STE is that the gradient is around zero in the region, where triplet constraint is strongly violated. That handles a noise in data because it does not try to satisfy such triplets that contradict consensus.

4.2 Navigating Through the Space of Embedding

Imagine a situation when a user is considering some particular category of objects and her task is to find an image in a dataset that matches this category. Such type of tasks falls into group *Query-by-Example*. Suppose that there is a sufficiently large set of images that contains, among others, a relatively small subset of images that match user's target category. The user is continuously presented a screen with few images and she is supposed to select one of those images until some of the presented images matches the target category.

The *Statistical Framework for Image Category Search from a Mental Picture* aims to optimize selection of the images presented to a user until she is given an image from her target category. This searching process should be, hopefully, done in a few rounds [9]. The core of the framework is a statistical model for relevance feedback. The session starts with a random screen of images and in each iteration the user is supposed to select the image from her target category when such image is displayed or the image that is closest to the category in case that no image from the category is displayed.

4.2.1 Statistical Framework

Formally let $Z = \{z_1, \ldots, z_n\}$ be a dataset of objects and $S \subset Z$ a target category. Let also D_t be a set of m images displayed in round t. It is supposed that if $D \cap S \neq \emptyset$ the user identifies $z_k \in S$ and the algorithm terminates. Otherwise it is supposed that the user selects image that is according to her metric "the closest" to S. There is a binary variable y_k associated with every image $z_k \in Z$, such that $y_k = 1$ when $z_k \in S$ and $y_k = 0$ when $z_k \notin S$. The framework maintains a response model for each i and updates posterior distribution on y_k after each feedback iteration. Let B_t denote the user responses for the first t rounds. Then $p_t(k) = P(y_k = 1|B_t)$ is a parameter that represents distribution of y_k. Since the images in the first round are taken randomly, then $p_0(i) = 0.5$. There are three principal components in the statistical model.

Update model. Let X_{D_t} be a user's response to a set of displayed images D_t. Update model computes $p_{t+1}(k)$ given $p_t(k)$ and X_{D_t}.

$$\begin{aligned} p_{t+1}(k) &= P(y_k = 1|B_{t+1}) \\ &= P(X_D = i|y_k = 1, D_{t+1} = D)p_t(k)/C_{t+1} \\ &= p_+(i|k, D)p_t(k)/C_{t+1}, \end{aligned} \tag{4.15}$$

where the normalizer $C_{t+1} = p_+(i|k, D)p_t(k) + p_-(i|k, D)(1 - p_t(k))$.

Answer model. Let $D = D_t$ be the displayed set of images at iteration t. If $D \cap S \neq \emptyset$, the algorithm terminates, otherwise suppose images $k \in S$ and $i \in D$ such that i is the

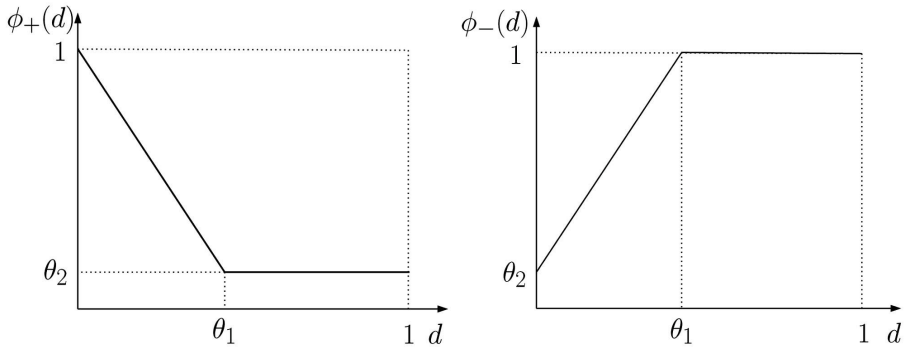


Figure 4.3: Shape of ϕ_+ and ϕ_+ functions userd in answer model. [2]

closest image to k according to user's point of view. For some metric δ there is defined a positive and a negative model:

$$p_+(i|k, D) = \frac{\phi_+(\delta(i,k))}{\sum_{j\in D} \phi_+(\delta(j,k))}, \tag{4.16}$$

$$p_-(i|k, D) = \frac{\phi_-(\delta(i,k))}{\sum_{j\in D} \phi_-(\delta(j,k))}. \tag{4.17}$$

The design of functions ϕ_+ and ϕ_- is based on the fact, that perceptual similarity of two objects is inversely proportional to their distance in metric δ and therefore ϕ_+ is monotonically decreasing and ϕ_- monotonically increasing function. Figure 4.3 shows the proposed functions as they are used in the framework.

The positive and negative functions introduce parameters θ_1, which serves as a threshold, from which the probability remains constant, and θ_2 which controls the coherence between normalized metric system δ and user's decisions.

Display model. Display model chooses which images to display for every round t. The algorithm computing distance model assumes, that the user selects randomly one image i from her tatget set S and uses this image i as a reference for all her responses. Since S is random subset of Z and i is randomly chosen from S, the reference image is a random variable R. Then, given a search history and a new answer $X_{D_{t+1}}$, the derived formula to compute next display set is

$$D_{t+1} = \arg\min_{D\subset Z} H(R|B_t, X_D), \tag{4.18}$$

where $H(\cdot)$ is the entropy.

This optimization problem requires looping over all $\binom{n}{m}$ combinations and can become intractable for larger sets of input objects and therefore not practically useful. In order to solve above equation there is formed a *Voronoi partition* based on D and the metric δ, which has cells of equal mass under the normalized $p_t(k)$ distribution over Ω. The algorithm uses sequential method to construct display set D that approximates the cell centers from Voronoi partition. The whole procedure is described in [8].

[2] Image taken from [9].

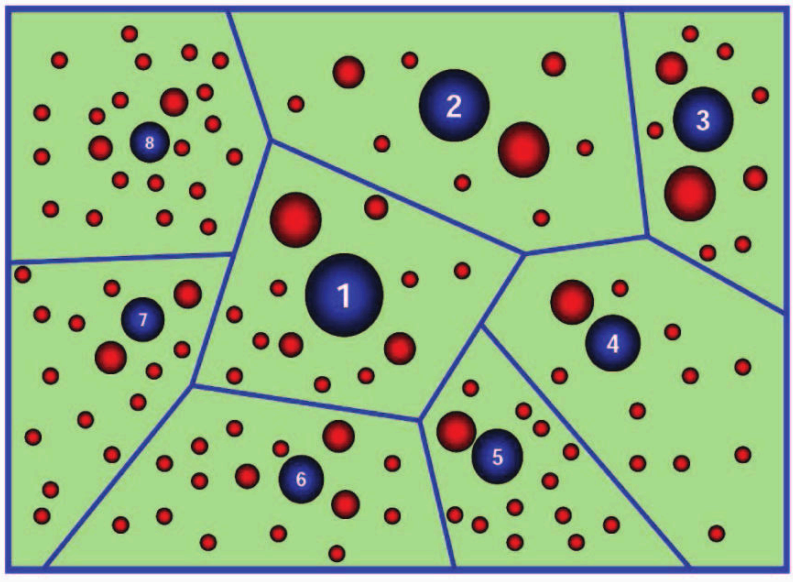


Figure 4.4: An example of Voronoi partition of a set of images with 8 cells. The sizes of disks representing images are proportional to their mass. Each image in some cell is closer to the center of the cell (blue disc) than to any other center. The centers are images in the optimal D. [2]

4.2.2 Parameter Determination

The positive and negative answer model depend on parametric functions ϕ_+ and ϕ_- and hence it is necessary to adjust parameters θ_1 and θ_2 for both models, which minimize the difference between metric system and how the similarity is perceived by humans. Especially θ_1 and θ_2 for the positive answer model have a strong impact on the performance of the method. The meaning of these parameters is clearly explained in Figure 4.3.

Determination of θ_1^+ (positive model) is based on statistical hypothesis test [9] in these steps:

1. Fix $\theta \in 0.05, 0.1, \ldots, 1$ to possible positive values of θ_1^+.
2. Choose randomly a target class S and its member $k \in S$.
3. Select two images $i, j \notin S$ such that $\delta(i,k) \approx \theta$ and $\delta(j,k) \in [\theta, 1]$ is chosen uniformly.
4. Present the summary of the target class S and the images i and j to the user asking her to select, which of the two images is closer to the target class S in her opinion.
5. Ask multiple users and repeat the previous steps for each user multiple times always with different S, k, i, j.

Considering two hypotheses

- $\mathbf{H_0}$: The i and j images are equally close to the target image in user's opinion,
- $\mathbf{H_1}$: The user has a preference for image i to be closer than j to the target image,

the aim of this θ_1 adjustment method is to select the highest value of θ, where the null hypothesis is rejected at the 0.05 significance level. Let n be the total number of users' choices for a θ value and $N(\theta)$ the number of times when the users selected i as the closer image, then the significance level p is approximated as

$$p(\theta) \approx 1 - \Phi\left(\frac{N(\theta) - \frac{n}{2}}{\frac{\sqrt{n}}{2}}\right), \tag{4.19}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function. Then the parameter θ_1 is chosen as the highest number θ, where the matching p is closest to the 0.05 significance level.

The estimation of θ_2 parameter assumes that given $k \in S$ and a display set D, a user chooses $i|\, \delta(i,k) \approx 0$ from D in case when all the other $m-1$ images $j \in D, j \neq i : \delta(j,k) \geq \theta_1^+$. Also $P(X_D \neq i | Y_k = 1) = 1 - p_+(i|k,D)$, then

$$\theta_2^+ \cong \frac{1}{m-1} \frac{1 - p_+(i|k,D)}{p_+(i|k,D)}. \tag{4.20}$$

The procedure to collect data from users in order to estimate θ_2^+ follows the algorithm is taken from [9]:

1. Randomly choose a target class S from the ground truth and an image $k \in S$.
2. Construct a display D for which there is an image $i \notin S$ with $\delta(i,k) \approx 0$ and the other $m-1$ images are at least θ_1^+ units away from k in the metric of the system.
3. Display D and a summary of S and ask the user to select the image that in his opinion is closest to S.
4. Record user's decision: $X_D = i$ or $X_D \neq i$.
5. Repeat these steps p times for each user.

The authors of [9] also tried to estimate parameters for the negative answer model in the same fashion as for the positive model, but the results were very similar when they used uniform negative model with parameters $\theta_1^- = 0, \theta_2^- = 1$. They also tried another extensions as "No preference" option or allowing users to view target class S at any time, but none of these attempts improved searching performance.

Chapter 5

Ways of Getting Similarities

The first part of this chapter serves as an overview of terms used in the thesis. In the second part there are proposed methods which improve efficiency of similarity retrieval process. Some of the proposed methods refers to experiments presented in later chapters.

5.1 Term Definition

To prevent misunderstanding, there is presented a brief definition of each term.

Taxonomically related objects. It is a group of objects where it is suitable to perform classification or fine-grained categorization into a fixed number of groups, because there have statically defined relations among subgroups. A particular example of such group is bird taxon. If we look into any ornithology book, we will very likely find there a taxonomy chart for birds. The hierarchy was specified by ornithology experts and it is static. Given a concrete bird, it is clear where to classify it. Such objects are called taxonomically related. The existence of taxonomic structure is suspended by a finite number of object classes.

Non-taxonomically related objects. Non-taxonomically related objects cannot be classified into a fixed groups, even though there might be defined some hierarchy or relations among them. As a representative of such group can be mentioned food.

Classification. Also known as categorization. It is the process of object understanding. The process itself refers to assignment of classes to input objects. Objects that fall to the same class are somehow similar. Objects, that can be classified have low inter-class distances and high intra-class distances. Animals can be consider as a representatives of such group, because there are methods in computer vision how to distinguish different species of animals.

Fine-grained categorization. It is similar to a classification, but the given objects are very similar among them. Inter-class and intra-class distances are in this case very similar, so it is often hard to determine the correct class for a presented object without some external assistance. Bird taxon can be mentioned as a representative which is subject to fine-grained categorization.

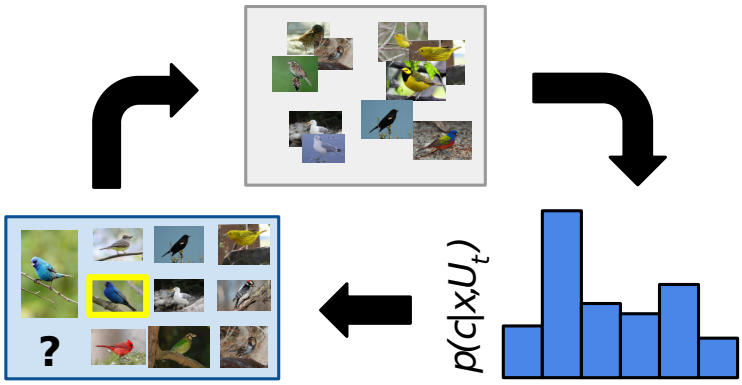


Figure 5.1: Scheme of a humans-in-the-loop system. [1]

Perceptual similarity. Perceptual similarity of some objects is based on its properties that humans are able to perceive with their senses. The traditional human senses are sight, hearing, taste, smell, and touch. There is no way how to precisely measure these properties.

Physical similarity. In contrast to perceptual ones, physical properties can be precisely measured using an appropriate measurement tool. On the basis of this measure, there can be also determined physical similarity.

System with humans-in-the-loop. System with humans-in-the-loop consists of the software part, and the human workers. Its usage is beneficial in cases, where a tight cooperation with human is necessary. Such system works iteratively: It presents a query to users and waits for her answer. Once the answer is submitted by them, the system updates its internal state and presents another query if needed. A scheme of such system is displayed in Figure 5.1.

Input objects. Let $Z = \{z_1, \ldots, z_n\}$ be the set of all input objects and $\mathbf{\Delta}$ be the following matrix of paired distances $\delta(i, j)$ of objects z_i and z_j:

$$\mathbf{\Delta} = \begin{pmatrix} \delta_{1,1} & \delta_{1,2} & \cdots & \delta_{1,n} \\ \delta_{2,1} & \delta_{2,2} & \cdots & \delta_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n,1} & \delta_{n,2} & \cdots & \delta_{n,n} \end{pmatrix}, \tag{5.1}$$

where $\delta(\cdot)$ is a general metric function. In most cases in this thesis, there is used as a metric function l_2-norm, also known as *Euclidean distance*.

Paired comparison. Paired (or pairwise) comparison may refer to a process of selecting one item from a pair of objects based on comparison with respect to some quantitative property.

In our case the term will be used for a process of comparing and sorting given two pairs of objects based on mutual similarity of objects in each pair, such that the objects in the first pair are "closer" with respect to some property, than objects from the second pair [1].

[1] Image taken from [23]

Formally, given input set of objects $Z = \{z_1, \ldots, z_n\}$ and some metric function δ, paired comparison π can be defined as

$$\pi = (i, j, k, l) | \, \delta(z_i, z_j) \leq \delta(z_k, z_l). \tag{5.2}$$

Triplets and the triplet universe. Triplet is a special case of paired comparison. For a paired comparison quadruple (i, j, k, l), there can be created a triplet placing $i = k$. Then, there is formed a tuple of 3 objects, where the first two objects are "closer" than the first and the third one [18]. Given input objects Z and metric function δ a triplet τ can be formally defined as

$$\tau = (i, j, l) | \, \delta(z_i, z_j) \leq \delta(z_i, z_l). \tag{5.3}$$

Inside triplet (i, j, l), the pair (i, j) will be referred to as a *short edge* and the pair (i, l) as a *long edge*.

Considering the definition of triplet and given set of inputs Z and its dissimilarity (distance) matrix $\mathbf{\Delta}$, let $\mathcal{T}_\Omega$ be a set of all existing triplets for the input set Z such that do not violate triplet consensus with respect to the matrix $\mathbf{\Delta}$. The total number of such triplets can be expressed as

$$\|\mathcal{T}_\Omega\| = \|Z\| \cdot \binom{\|Z\| - 1}{2}. \tag{5.4}$$

5.2 Triplet Retrieval Efficiency

The first question that probably emerges in relation to triplet retrieval from human workers is something like: "How many triplets are necessary to create a good-quality embedding?". It is apparent that, especially for a larger dataset, is impossible to ask workers about their judgment for the whole triplet universe (omitting the need of redundant triplets for quality assurance). Considering Equation 5.4 for the size of triplet universe, it is obvious that its asymptotic complexity is $O(n^3)$ and hence the number of triplets grows cubically. For real datasets containing thousands or more objects would be intractable to gather all triplets in this way.

This implies a need of some method to reduce the total number of triplets necessary for a good-quality embedding construction. In the following sections, there are proposed a concepts of *templates* and *screens*, as well as a way how to use them in order to increase the amount of triplets produced from workers' answers. Furthermore, there is a discussion about a required amount of triplets and the overview of algorithms usable for triplet selection.

5.3 Templates and Screens

The object similarity retrieval process from humans comprise a need to design an appropriate user interface. Using the *implicit crowdsourcing* for similarity retrieval would involve incorporation of the algorithm gathering similarities into some game or application, which would include a tight cooperation with designers of such games or applications.

Also in explicit tasks, where the workers are directly asked to solve some queries in order to gather similarities, there is still a need to present them the queries in a convenient way, such that the efficiency of similarity retrieval is maximal. The different types of these interfaces are referred to as *templates* in this work. Their usage will be demonstrated and compared using appropriate data – images. There will be presented two types of templates:

Triangle and *Grid with a probe*. The particular purpose of templates is to present input objects to workers in order to collect triplets from their answers.

Multiple-screen tasks. Most of similarity retrieval tasks are repetitive and in many cases it is suitable (for quality assurance) to present to a worker multiple easy subtasks wrapped in a larger task. The term *screen* is in this work used to define a single template with a task presented to a worker. HITs are usually composed of multiple *screens* which are also referred to as *set of screens* or *set*.

5.3.1 Triangle

The *triangle* template consists of three images placed in vertices of an equilateral triangle. Such template can be defined it as follow: Given a set of input objects Z, let

$$T_{\triangle} = \{z_i, z_j, z_k\} \tag{5.5}$$

be a *triangle* template where items z_i, z_j, z_k are items from the input set Z selected to the triangle. See Figure 5.2 for an example of the triangle template.

The task for workers is to select the edge, which connects the most similar pair of images in triangle. Given the triangle template T and a dissimilarity function δ, the worker's solution of the task S is defined as

$$S_{sel}(T) = (i, j) |\, \delta(z_i, z_j) < \delta(z_i, z_k) \wedge \delta(z_i, z_j) < \delta(z_j, z_k). \tag{5.6}$$

In case of the triangle template, two triplets can be generated from each answer obtained from a worker. Suppose the template $T_{\text{triangle}} = \{z_i, z_j, z_k\}$ and the worker's answer $S_{sel}(T) = (i, j)$, the set of triplets $\mathcal{T}_{\partial}$ generated from this particular answer is

$$\mathcal{T}_{\partial} = \{(i, j, k), (j, i, k)\}. \tag{5.7}$$

5.3.2 Grid with a probe

Grid with a probe is the second type of template presented in this work. It contains a *probe* – one image, which serves as a reference and a matrix (grid) of images. Formally, given a set of input objects Z, let

$$T_{\text{gp}} = (p, G, m) \tag{5.8}$$

be a *grid with a probe* template, where $G \subset Z$, $\|G\| = m$, $m \geq 2$ is a subset of input set selected to the grid and $p \in Z, p \notin G$ is the probe. For an example of this type of template see Figure 5.2. There are two following types of tasks defined for the *grid with a probe* template.

Selection task. In the selection task, the worker is asked to select s images from the grid, that are most similar (or dissimilar) to the probe. Given the template $T = (p, G, m)$, where $G = \{z_{c_1}, z_{c_2}, \ldots, z_{c_m}\}$ and a dissimilarity function δ, the selection task can be defined as

$$S_{sel}(T, s) = C, \tag{5.9}$$

where $C = \{c_1, c_2, \ldots, c_s\}$, $\forall i \in C, \forall z_j \in (G \setminus C) : \delta(p, z_i) < \delta(p, z_j)$. Given a template T, its selection $S_{sel}(T, s) = C$, and the set $R = \{c_{s+1}, c_{s+2}, \ldots, c_m\}$, the triplets generated from the selection are defined as a Cartesian product

$$\mathcal{T}_{\partial} = p \times S \times R \tag{5.10}$$

(a) Triangle

(b) Grid with a probe

Figure 5.2: Examples of templates.

in case of positive answers (most similar objects) or in case of the negative ones

$$\mathcal{T}_\partial = p \times R \times S. \tag{5.11}$$

The amount of triplets generated per selection is $\|\mathcal{T}_\partial\| = s(m-s)$.

Ordering task. This type of task prompts worker to order images in the grid from most similar to the less one with respect to the probe image. Given a template $T = (p, G, m)$, let $(G, <_\delta)$ be a total order on G, such that $\forall z_i, z_j \in G : z_i < z_j \iff \delta(p, z_i) < \delta(p, z_j)$. Then $(G, <_\delta) = z_{o_1} < z_{o_2} < \cdots < z_{o_m}$ and we denote $(O_G, <_\delta) = o_1 < o_2 < \cdots < o_m$ the order $(G, <_\delta)$ where the items from the grid are represented by their indices. Then the ordering task on template T is defined as

$$S_{ord}(T) = (O_G, <_\delta). \tag{5.12}$$

Given the template T and the ordering result $S_{ord}(T) = o_1 < o_2 < \cdots < o_m$, the triplets from this particular answer are generated according to the following algorithm:

$$\mathcal{T}_\partial = (p, o_i, o_j) | \, \forall i \in \{1, 2, \ldots, m-1\}, \forall j \in \{i+1, i+2, \ldots, m\}. \tag{5.13}$$

The number of triplets generated per ordering answer is $\|\mathcal{T}_\partial\| = \frac{m(m-1)}{2}$.

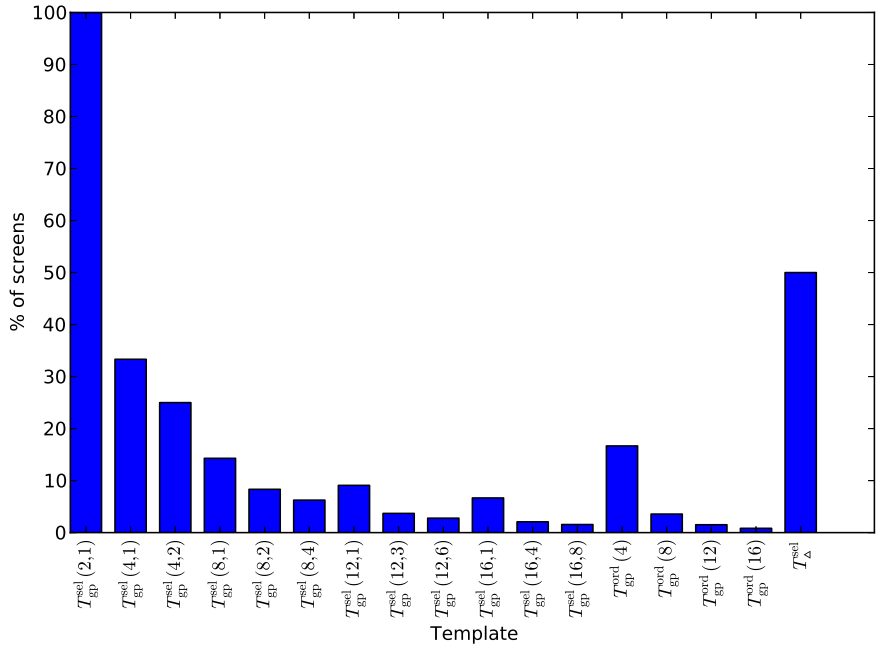
5.4 Template-specific Triplets

It is obvious from previous sections how to generate triplets from one worker's answer. If we are given multiple answers from templates with different images or even different templates, the resultant set of triplets $\mathcal{T}$ is

$$\mathcal{T} = \bigcup_\partial \mathcal{T}_\partial. \tag{5.14}$$

In the following text we denote $T_\triangle^{sel}$ a triangle template with a selection task performed on it and similarly $T_{gp}^{sel}(m, s)$ the template $T_{gp}(p, G, m)$ with the selection task $S_{sel}(T_{gp}, s)$ and $T_{gp}^{ord}(m)$ the template $T_{gp}(p, G, m)$ with the ordering task.

If we take into account the template $T_{gp}^{sel}(2, 1)$ and we try to generate triplets according to the algorithm mentioned in Section 5.3.2, we will obtain just one triplet. This corresponds



(a) Percentage of screens needed to get the same amount of triplets from different templates.

Template	$T^{sel}_{gp}(2,1)$	$T^{sel}_{gp}(4,1)$	$T^{sel}_{gp}(4,2)$	$T^{sel}_{gp}(8,1)$	$T^{sel}_{gp}(8,2)$
Reduction factor	1	3	4	7	12
$T^{sel}_{gp}(8,4)$	$T^{sel}_{gp}(12,1)$	$T^{sel}_{gp}(12,3)$	$T^{sel}_{gp}(12,6)$	$T^{sel}_{gp}(16,1)$	$T^{sel}_{gp}(16,4)$
16	11	27	36	15	48
$T^{sel}_{gp}(16,8)$	$T^{ord}_{gp}(4)$	$T^{ord}_{gp}(8)$	$T^{ord}_{gp}(12)$	$T^{ord}_{gp}(16)$	$T^{sel}_{\triangle}$
64	6	28	66	120	2

(b) Reduction factor of screens for different templates with constant number of triplets.

Figure 5.3: Comparison of different templates with respect to the quantity of triplets.

to the classical triplet definition (see Section 5.1). This template will be used as a reference in future experiments. Since we already defined the triplet and the template which can generate it, what is the motivation to use different templates?

Quantity of Triplets. Lets compute the number of triplet produced from some particular templates and tasks. If we assign 100% to the number of screens needed to achieve some certain amount of triplets using the default template $T^{sel}_{gp}(2,1)$ generating just one triplet, we can compute the percentage of needed screens to reach the same amount of triplets for some other templates. The results are summarized in Figure 5.3.

Comparing the number of triplets produced from different templates, we can see that using some of them, we can significantly reduce the number of screens and hence the costs to achieve the same number of triplets. The highest number of triplets is produced by template $T^{ord}_{gp}(16)$, where the screen reduction factor w.r.t basic template $T^{sel}_{gp}(2,1)$ is 120:1.

However, is the highest quantity of triplets produced from a template the only priority that should be taken into account? There are also another important questions like: "Do the triplets produced by different templates have the same quality?", or "What is the limit size of the grid that are workers able to answer efficiently without significant error?".

Quality of triplets. The quantity of produced triplets is undoubtedly important, however, we have to consider also the quality of triplets produced by different templates. This statement is supported by the experiment described in Section 6.4.

In this experiment there were computed embedding errors for different templates and different embedding functions with fixed number of triplets. The best lowest error can be observed in embedding created from triplets using $T^{\text{sel}}_{\text{gp}}(2,1)$ template. There is also observed some correlation between number of triplets produced by a certain template and their embedding error (especially using t-STE or GNMDS embedding function). If the workers answering the tasks was paid per generated triplet, it would be best to use the default template $T^{\text{sel}}_{\text{gp}}(2,1)$ to get their answers. The comparison of these errors is is Figure 6.1a.

Fortunately, the workers are not paid per triplet, but rather per screen. More specifically they want to be paid for time they spend working on a task, but the reward is specified per task. So if we repeat the same experiment and instead of constant number of triplets we use constant number of screens, the results are dramatically different as one can see in Figure 6.1b. Since each default template $T^{\text{sel}}_{\text{gp}}(2,1)$ produces only one triplet, the total embedding error remains quite high. On the other hand using some templates that produces more triples, the resulting error is fairly low. However, the embedding error is not inversely related to the number of produced triplets per template.

5.5 Necessary Amount of Triplets

Once we figured out how to gather triplets from workers in an effective way, it is also important to know, how many triplets are necessary to build an embedding that would not suffer a significant error. As mentioned, it would be really expensive especially for a larger datasets to let workers to answer all triplets, even using some templates, which substantially reduces the costs. Therefore, the next task aims to determine, how many triplets or rather how many percent of all triplets do we really need and if the percentage remains still the same also for a different number of objects.

Assume that the necessary percentage for a good quality embedding is 1% for 100 objects, which means 4 851 triplets. If this 1% of triplet universe is required to build the same-quality embedding also for 1 000, 10 000 or 100 000 objects, then the task of getting such number of triplets will become extremely expensive and hence intractable already for a quite small number of objects. The number of required triplets and the costs are summarized in Table 5.1a.

Fortunately, according to the experiment described in Section 6.6, the required percentage is not the same and although the size of triplet universe grows cubically, the amount of necessary triplets seems to grow much slower, perhaps even linearly. If the growth is really linear, then the expenses are reduced by cube root. The situation in such case is summarized in Table 5.1b.

These results were obtained for synthetically generated triplets based on a ground truth distance matrix. The situation could be different in case of real data, because workers do not

Objects	All triplets	1% of triplets	Cost
100	485 100	4 851	$1.5
1 000	498 501 000	4 985 010	$1 558
10 000	499 850 010 000	4 998 500 100	$1 562 031
100 000	499 985 000 100 000	4 999 850 001 000	$1 562 453 125

(a) Cost estimation in case of 1% of all triplets is required.

Objects	All triplets	Necessary triplets	Cost
100	485 100	4 851	$1.5
1 000	498 501 000	48 510	$15
10 000	499 850 010 000	485 100	$152
100 000	499 985 000 100 000	4 851 000	$1 516

(b) Cost estimation in case of a linear growth of necessary triplets.

Table 5.1: Cost comparison (using $T_{\text{gp}}^{\text{sel}}(12, 4)$ paying workers $4/hour) in case when the percentage of all triplets required for the same-quality embedding is constant (and thus the amount of such triplets grows cubically) in terms of different number of objects, and in case when the amount of necessary triplets grows linearly (in practice the number might be multiplied by a constant).

answer always according to the ground truth and in some cases (e.g. perceptual similarity), it is also difficult or even impossible to find the ground truth function. Therefore, in practice it is better to collect rather larger amount of triplets in order to preserve the quality of the embedding.

5.6 Algorithms for Triplet Selection

Although we have already found the effective way, how to obtain triplets from the workers, and how to reduce the amount of required triplets, there are still some remaining possibilities, how to make this process even more effective. One of them is usage a different algorithm that chooses which triplets are displayed to workers. The triplets were so far chosen randomly, however, there is very likely a better algorithm, that could use a history of workers' responses.

Some research related to effective triplet selection has been done in [10, 17]. The idea of both these methods is to select the most informative triplets and display them to workers first, however these algorithms do not use bulk triplet collection and collect triplets one per a screen. Active MDS [10] considers all triplets as a partial ranking with respect to each point of the embedding and it places geometrical constraints to define the space where the points may lie. The adaptive selection algorithm presented in [17] models information gain for each triplet as a probability distribution over all points in the embedding.

Using bulk triplet collection approach instead of one-by-one gives room for exploring new methods. Although this investigation is not a part of this thesis, it can be a direction for a potential future work.

Chapter 6

Experiments on a Toy Dataset

The aim of experiments presented in this chapter is to determine an optimal setup for similarity retrieval process, that would lead to inexpensive data embedding without significant error. There are performed experiments to determine the necessary amount of triplets and to compare quality of the embeddings constructed by different methods, the efficiency of different templates used for triplets harvesting, and the embedding quality using different percentage of corrupted triplets (such that violate consensus).

USCA312 dataset. For the following experiments, there was chosen a dataset *USCA312*[1] of 312 cities in the US and Canada. The dataset contains names of the cities, their longitude and latitude, xy coordinates and paired distances. However, for the needs of experiments, there were used just the names of cities and the paired distances. There are several reasons why this dataset was chosen: there is no need to gather human responses for this dataset, because it contains the explicit ground truth (paired distances) based on which the triplets can be generated automatically. This dataset is large enough to show, how the mental matching algorithm works in practice, and also the whole embedding can be displayed in 2D without additional error.

To collect answers there was used instead of real workers a computer program, which, given set of cities, automatically provides answers based on comparison of paired distances. Such algorithm will be referred to as *artificial worker* in terms of experiments with the toy dataset.

6.1 Triplet Universe Generation

Considering the classical triplet definition the total number potentially non-violated triplets for this dataset is:

$$312 \cdot \binom{311}{2} = 15\,039\,960 \tag{6.1}$$

To avoid possibly duplicated triplets all 15 039 960 triplets were sequentially generated using nested `for` loops. At first, there were generated IDs of participating cities in each triplets in this form: $(i, \{j, k\})$. Then each triplet was created as a tuple $(i, j, k) | \delta(i, j) < \delta(i, k)$. In the end all triplets were randomly shuffled using Linux command `shuf`.

[1] http://people.sc.fsu.edu/~jburkardt/datasets/cities/cities.html

6.2 Templates and Artificial Worker

Some triplets generated in the following experiments come from specific templates. Using the definition of templates and screens from Section 5.3 there was created a program called *Artificial Worker*, which generates answers from the presented screens.

In the following experiments there were used 16 different *grid with a probe* templates with selection and ordering tasks (whereas the triangle template $T^{\text{sel}}_{\triangle}$ was omitted):
$T^{\text{sel}}_{\text{gp}}(2,1)$, $T^{\text{sel}}_{\text{gp}}(4,1)$, $T^{\text{sel}}_{\text{gp}}(4,2)$, $T^{\text{sel}}_{\text{gp}}(8,1)$, $T^{\text{sel}}_{\text{gp}}(8,2)$, $T^{\text{sel}}_{\text{gp}}(8,4)$, $T^{\text{sel}}_{\text{gp}}(12,1)$, $T^{\text{sel}}_{\text{gp}}(12,3)$, $T^{\text{sel}}_{\text{gp}}(12,6)$, $T^{\text{sel}}_{\text{gp}}(16,1)$, $T^{\text{sel}}_{\text{gp}}(16,4)$, $T^{\text{sel}}_{\text{gp}}(16,8)$, $T^{\text{ord}}_{\text{gp}}(4)$, $T^{\text{ord}}_{\text{gp}}(8)$, $T^{\text{ord}}_{\text{gp}}(12)$, $T^{\text{ord}}_{\text{gp}}(16)$.

Simulation Pipeline. In the selection task in this particular dataset the *artificial worker* is asked to select from the grid s closest cities with respect to the probe city. In ordering task, the *artificial worker* is supposed to sort the cities in the grid from the closest to the furthermost one with respect to the probe city.

The program that simulates this worker answering consist several parts:

1. Grid composer picks random objects from the input set Z and generates the virtual *grid with a probe* template. It also assures that there is no duplicated object in the template.

2. *Artificial Worker* is given a virtual template and simulates human answers on it. The images in the grid are ordered according to ground truth of paired distances. Once the grid is ordered, the program returns indices of the first s items in the case of selection task or returns indices of all ordered items in the grid in case of ordering task.

3. Triplet generator is also given a virtual template and the *artificial worker's* answer. It produces triplets from the answers depending on performed task (selection or ordering) using the algorithms defined in Section 5.3.2. All generated triplets are stored in a text file.

4. Embedding computation: There are performed three methods for embedding construction: GNMDS, CKL, and t-STE. Each method is performed 10 times for each triplet configuration, because all these methods are stochastic and produce different outputs every time.

5. Error measurement: Embedding error is computed for every produced embedding and the mean error over the 10 iterations is saved. The error functions are defined in Section 6.3.

For each part except embedding computation was used programming language Python with `NumPy` library. In case of embedding computation algorithms there were used their implementations in Matlab [18].

6.3 Error Measurement

In order to measure the quality of the embedding, there is a need to have some error function. Since the ground truth is known to us, we can compare the embedding with real paired distances. There are presented two error functions for embedding quality evaluation. The comparison of both methods can be seen in Figure 6.2.

Embedding vs. Triplet Universe. This type of quality evaluation is based on comparison of real paired distances for all existing triplets that can be generated for the set of input cities with the matching distances in the embedding. Every single distance violation is accumulated to the total error counter and then normalized by number of comparisons. Since all used embedding algorithms are non-metric, it is more appropriate to compare just embedding distances with all triplets, instead of a proportional comparison of real distances directly with distances in the embedding.

Formally, given the set of inputs Z, its universe of triplets $\mathcal{T}_\Omega$ with respect to the paired distances $\mathbf{\Delta}_Z$, embedding E, and it's paired distance matrix $\mathbf{\Delta}_E$ the error ϵ_Ω can be expressed as follow:

$$\epsilon_\Omega = \frac{\sum_{(i,j,l)\in\mathcal{T}_\Omega:\delta_E(i,j)>\delta_E(i,l)} 1}{\|\mathcal{T}_\Omega\|} \tag{6.2}$$

One could argue that the more precise way how to measure embedding would be to compute the ratio between the *short edge* and the *long edge* of each triplet. This approach would be probably useful in metric methods like MDS, but although all examined methods form metric space for embedding, they are based on non-metric comparisons and thus measuring the magnitudes of error would be misleading.

Since this method compares distance ratios in embedding created just by a subset of triplet universe with ratios of real distances using the whole triplet universe, this error function indicates the generalization ability of the embedding algorithm and the capability of building whole embedding when just a part of comparisons (triplets) is provided.

Embedding vs. Supplied Triplets. The second possible way how to evaluate embedding quality is to count just the number of inconsistencies between triplets supplied to the embedding function and distances in created embedding. This method does not evaluate the ability of generalization, but more like the effort to satisfy provided triplets in the created embedding. The formal definition is very similar to the previous one, just instead of triplet universe is given just its subset $\mathcal{T} \subset \mathcal{T}_\Omega$ which was used for embedding creation.

$$\epsilon = \frac{\sum_{(i,j,l)\in\mathcal{T}:\delta_E(i,j)>\delta_E(i,l)} 1}{\|\mathcal{T}\|} \tag{6.3}$$

6.4 Experiment 1: Quality of Triplets

The goal of this experiment is to measure the quality of triplets produced by different templates. The assumption is that the highest-quality embedding can be created from the classical triplet definition using the template $T^{\text{sel}}_{\text{gp}}(2,1)$. This experiment should give the answer on the question how significant is the difference in quality of triplets produced by different templates and which template gives the lowest error for its embedding when a constant number of screens (and thus in some cases very different number of triplets) is used for each template.

Experiment Setup. In the first part of the experiment the number of triplets was fixed to 0.1% of all possible triplets, which is in this case 15 040 and for each template there was computed a necessary number of screens in order to reach the specified number of triplets. The resulting number of necessary screens are displayed in Table 6.1a. In the second part, there was fixed a constant number of screens to 1 000 (in terms of the size of dataset and

Template	$T^{\text{sel}}_{\text{gp}}(2,1)$	$T^{\text{sel}}_{\text{gp}}(4,1)$	$T^{\text{sel}}_{\text{gp}}(4,2)$	$T^{\text{sel}}_{\text{gp}}(8,1)$	$T^{\text{sel}}_{\text{gp}}(8,2)$
Screens	15040	5014	3760	2149	1254
$T^{\text{sel}}_{\text{gp}}(8,4)$	$T^{\text{sel}}_{\text{gp}}(12,1)$	$T^{\text{sel}}_{\text{gp}}(12,3)$	$T^{\text{sel}}_{\text{gp}}(12,6)$	$T^{\text{sel}}_{\text{gp}}(16,1)$	$T^{\text{sel}}_{\text{gp}}(16,4)$
940	1368	558	418	1003	314
$T^{\text{sel}}_{\text{gp}}(16,8)$	$T^{\text{ord}}_{\text{gp}}(4)$	$T^{\text{ord}}_{\text{gp}}(8)$	$T^{\text{ord}}_{\text{gp}}(12)$	$T^{\text{ord}}_{\text{gp}}(16)$	
235	2506	538	228	126	

(a) A necessary amount of screens for different templates to produce 15 040 triplets.

Template	$T^{\text{sel}}_{\text{gp}}(2,1)$	$T^{\text{sel}}_{\text{gp}}(4,1)$	$T^{\text{sel}}_{\text{gp}}(4,2)$	$T^{\text{sel}}_{\text{gp}}(8,1)$	$T^{\text{sel}}_{\text{gp}}(8,2)$
Triplets	1 000	3 000	4 000	7 000	12 000
$T^{\text{sel}}_{\text{gp}}(8,4)$	$T^{\text{sel}}_{\text{gp}}(12,1)$	$T^{\text{sel}}_{\text{gp}}(12,3)$	$T^{\text{sel}}_{\text{gp}}(12,6)$	$T^{\text{sel}}_{\text{gp}}(16,1)$	$T^{\text{sel}}_{\text{gp}}(16,4)$
16 000	11 000	27 000	36 000	15 000	48 000
$T^{\text{sel}}_{\text{gp}}(16,8)$	$T^{\text{ord}}_{\text{gp}}(4)$	$T^{\text{ord}}_{\text{gp}}(8)$	$T^{\text{ord}}_{\text{gp}}(12)$	$T^{\text{ord}}_{\text{gp}}(16)$	
64 000	6 000	28 000	66 000	120 000	

(b) The amount of triplets generated from 1 000 screens using different templates.

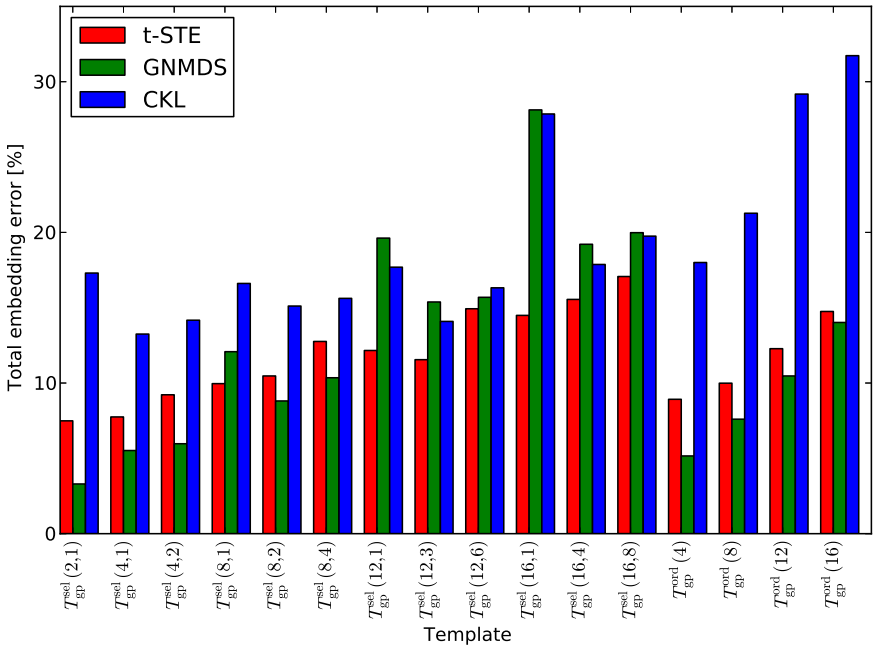
Table 6.1: A necessary amounts of screens to achieve the same amount of triplets and a number of triplets produced from a constant amount of screens for different templates.

the fact that some templates can produce up to 120 triplets per template). The resulting numbers of triplets produced by different templates are displayed in Table 6.1b.

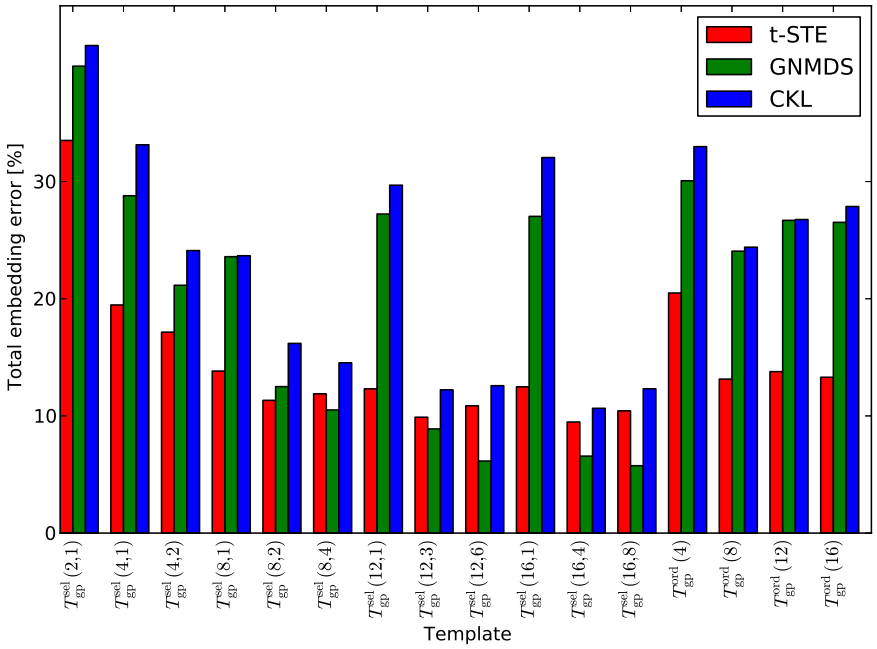
These numbers of screens along with the matching template were subsequently used as a input of the simulation pipeline, and there were computed embeddings and their mean errors for each part of this experiment separately. The results of the experiment can be seen in Figure 6.1.

Discussion. From the results of the first part of the experiment is obvious, that the lowest error rate and thus the highest-quality triplets are produced by template $T^{\text{sel}}_{\text{gp}}(2,1)$ which confirms the assumption (at least in the case of t-STE and GNMDS methods). Another interesting fact is, that for sorting tasks the error is highest among the templates with the same grid size when the task is to select just one item from the grid. Especially GNMDS and CKL methods suffer in such cases by significantly higher error. The triplets produced in this way have much more different *long edges* than *short edges*. In other words using the definition of the set of triplets $\mathcal{T}_\partial = p \times S \times R$ produced by a selection task on a *grid with a probe* template $T_{\text{gp}} = (p, G, m)$, then the set S contains just one item and $m - 1$ items are in the set R, so each produced triplet by this single template triplet has the same short edge.

In case of constant number of screens, the results are diametrically different. The fact that some templates produce incomparably higher amount of triplets than the basic template $T^{\text{sel}}_{\text{gp}}(2,1)$ surpasses its poorer quality and gives a noticeably lower embedding error. Also the higher error of GNMDS and CKL for the templates that generates much smaller number of *short edges* than *long edges* in the case of fixed number of screens is even amplified. The another observation is that while the GNMDS error was in the first case mostly lower than t-STE error, in the second case the situation is reversed. Especially in the cases of ordering tasks is the difference notable.



(a) Constant number of triplets.

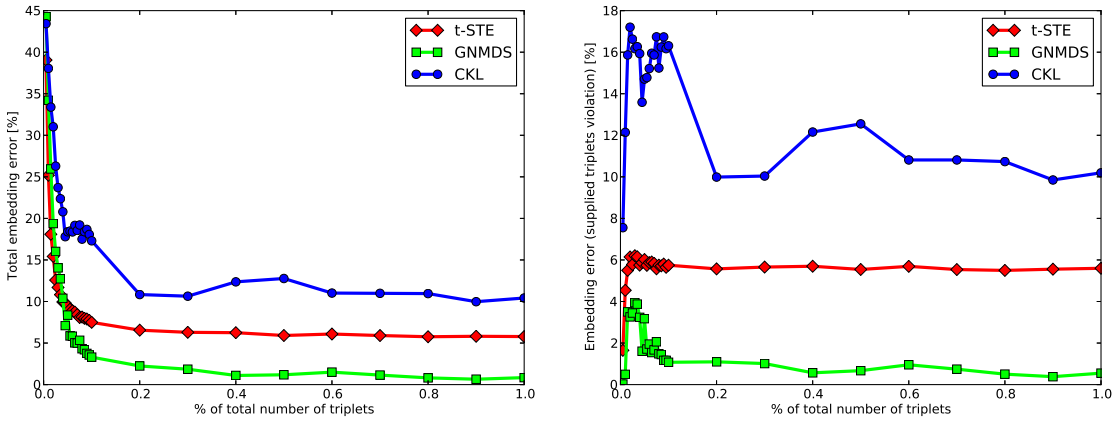


(b) Constant number of screens.

Figure 6.1: Comparison of embedding errors for a constant number of (a) triplets, (b) screens, using different templates.

%	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05
#	752	1 504	2 256	3 008	3 760	4 512	5 264	6 016	6 768	7 520
%	0.055	0.06	0.065	0.07	0.075	0.08	0.085	0.09	0.095	0.1
#	8 272	9 024	9 776	10 528	11 280	12 032	12 784	13 536	14 288	15 040
%	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
#	30 080	45 120	60 160	75 200	90 240	105 280	120 320	135 360	150 400	

Table 6.2: Percentages and their corresponding numbers of objects used in the experiment.



(a) Embedding vs. Triplet Universe

(b) Embedding vs. Supplied Triplets

Figure 6.2: Comparison of two embedding error functions from 6.3 w.r.t the percentage of triplets from $\mathcal{T}_\Omega$ provided to the embedding function.

6.5 Experiment 2: Embedding Error

The first part of this experiment aims to show the error of embedding constructed by different algorithms for a different number of triplets. It also shows the difference between the previously defined error functions (see Section 6.3). In the second part, there is examined the influence of corrupted triplets to the embedding error.

Experiment setup. The triplets were generated using the triplet template $T_{sel}(2, 1)$. For the selection of triplets there was used the shuffled list of triplet universe (see Section 6.1) and the triplets were selected from the beginning of the list. Table 6.2 shows the used percentages of triplets from the universe and the corresponding number of triplets.

There were performed all three embedding functions on each set of triplets and the mean embedding error was computed for both error functions both mentioned in Section 6.3. The resulting embedding errors for both function are presented in Figure 6.2. In the second part, the number of triplets was fixed to 1% from all possible triplets selecting first 15 040 ones from the shuffled list of all generated triplets. Then there were inverted last two items in triplet at $0\%, 5\%, 10\%, 15\%, \ldots 100\%$ of used triplets (triplet (i, j, l) was inverted to (i, l, j)). Subsequently, there were created 10 embeddings for each method (GNMDS, CKL, t-STE) and each percentage of corrupted triplets and the mean embedding errors were computed. The results are displayed in Figure 6.3.

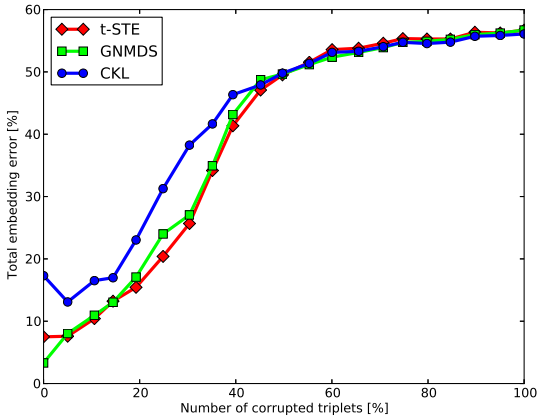


Figure 6.3: Embedding error with respect to percentage of corrupted triplets using 1% of total amount of triplets.

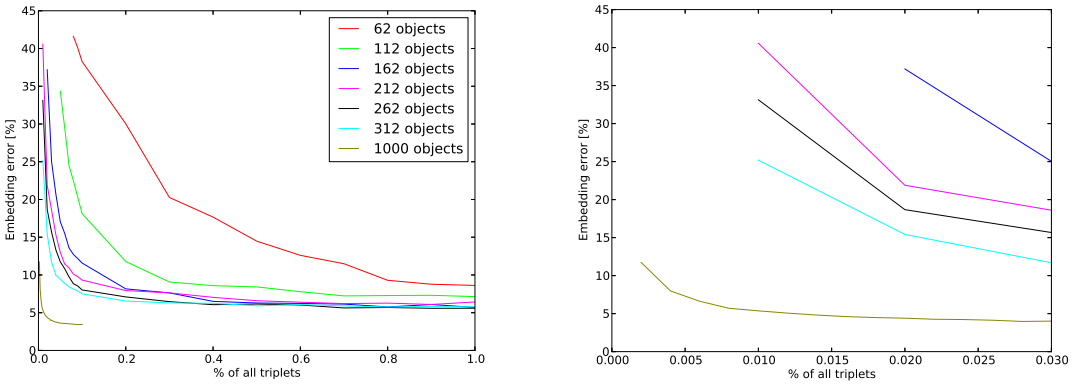
Discussion. In the case of the first error function (Embedding vs. Triplet Universe) which determines the overall embedding error is the error, it is evident from the results that the error is rapidly decreasing until about 0.1% of the total number of triplets in case of t-STE and GNMDS, and about 0.2% in case of CKL embedding function. From these threshold values the total error declines very slowly or remains nearly constant.

The second error function (Embedding vs. Supplied Triplets) returns low error for a very little number of supplied triplets and then the error increases and fluctuates until it reaches the similar threshold amount of triplets as in case of the first error function (0.1% for t-STE and GNMDS and 0.2% for CKL). From the threshold value the error remains almost constant. Unlike the first error function, which gives us the information about ability of embedding functions to generalize the embedding, this one shows the effort of embedding functions to satisfy given triplets.

6.6 Experiment 3: Necessary Amount of Triplets

From the previous experiment is apparent that the more triplets are used to build embedding the lower error is obtained. But what is the necessary amount of triplets needed to construct an embedding with the same error rate for different number of object? In other words, having 50 objects and the error of their embedding is e.g. 10% for some percentage of triplets from the universe, is the same percentage of triplets needed also for 100 or 200 objects? This experiment tries to answer these questions.

Experiment setup. As a measure of the embedding error there was used the error function, that compares distances in embedding with triplet universe (see Section 6.3). In the experiment, 6 different subsets of the cities were randomly chosen from the dataset. The sizes of subsets were determined to 62, 112, 162, 212, 262, and 312. There were also used a subset of 1 000 objects taken from a dataset of Californian post offices, that is useful for estimation of the number of necessary triplets at a food image dataset (see Section 8). Nevertheless, the fact that these 1000 objects come from a different dataset has no impact on reliability of the results, because triplets are generated on the basis of a ground truth,



(a) Original error functions.

(b) Zoomed detail.

Figure 6.4: Embedding error w.r.t percentage of used triplets for different number of objects.

which is in both cases available. With the following fixed percentages of triplet universe to $0.01\%, 0.02\%, \ldots, 0.09\%, 0.1\%, 0.2\%, \ldots, 0.9\%, 1.0\%$ and for all subsets of objects (except 1 000 objects subset) there was 10 times built an embedding using t-STE method, computed the embedding error, and saved the mean value over these 10 iterations. For the set of 1 000 object there were used just 2 iterations and a percentage range of triplets from the universe from 0.002% to 0.1% because of computational complexity. Notice that there are performed multiple iterations because the embedding function is stochastic and returns every time a different embedding.

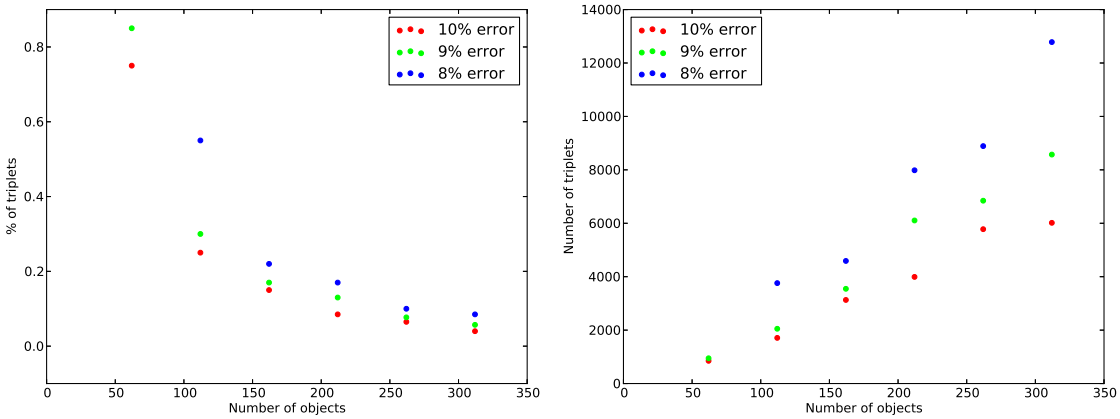
In practice, for each subset of cities a list of triplet universe was generated and shuffled according to the procedure described in 6.1. Then all used percentage of all triplets was the list of triplet universe sliced from the beginning up to the corresponding number of triplets.

Results and discussion. The embedding errors for each number of objects with respect to the used percentage of triplets from the universe are displayed in figure 6.4. The error function was created for each amount of objects by linear interpolation of resulting errors.

If the percentage of all triplets required to build the same quality embedding was the same for every number of objects, all the resulting error functions would collapse into only one. The fact that they did not collapsed into one error function implies that the percentage of triplets vary for distinct number of objects, however, from this figure is not really evident how much they vary.

Nevertheless, fixing the embedding error to some level, there can be displayed the dependency of percentage of all triplets or rather the concrete number of triplets with respect to the amount of objects (see Figure 6.5). Note that all percentages from triplet universe are just estimated from the resulting embedding errors. There is no way how to enforce exact embedding error since the embedding function is stochastic and the results from all iterations vary. Them the number of triplets was computed from the estimated percentage of the triplet universe size.

From Figure 6.5b is evident, that even though the amount of all triplets grows cubically, the necessary amount of triplets that preserves embedding quality seems to grow much slower, maybe even linearly. To figuring out the real complexity of the necessary amount of triplets would need a further research and it is not the subject of this thesis, however, it



(a) Percentage of triplets.

(b) Number of triplets.

Figure 6.5: Estimation of percentage and number of triplets needed to construct the same quality embeddings for different numbers of objects.

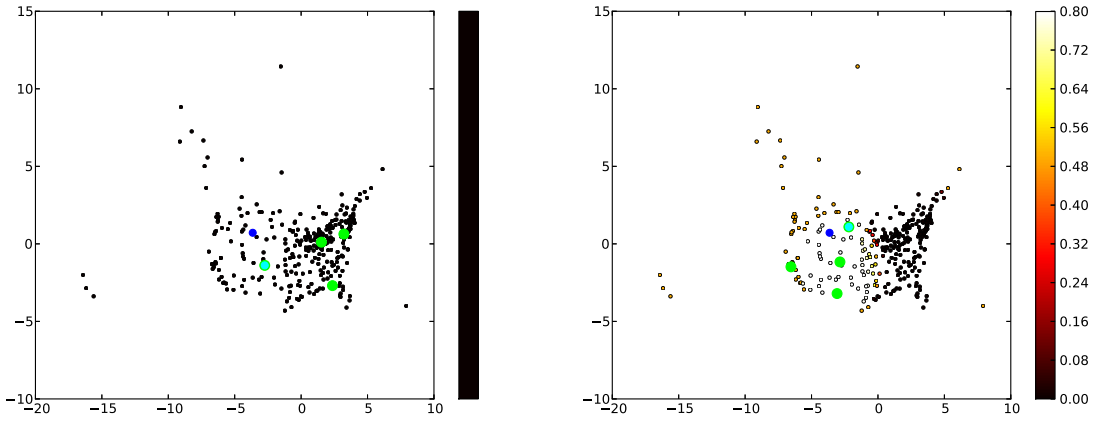
might be a challenging task for the future work.

6.7 Experiment 4: Navigation Through the Embedding

The last experiment on the US and Canada cities dataset concerns navigating through the space of embedding and finding a target object in the embedding. It is a demonstration of mental matching algorithm defined in Section 4.2 on this dataset. Although the main purpose of the algorithm is to find a target image in a set of images, it works with any data embedding.

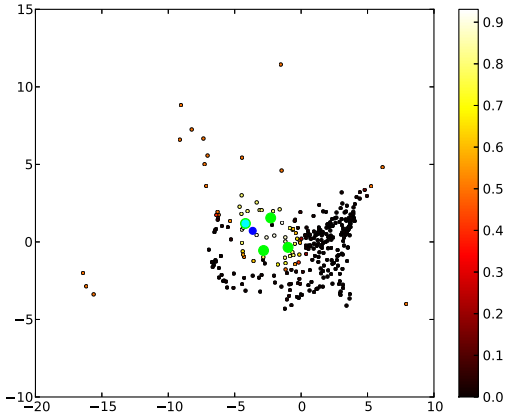
Experiment setup. In case of this toy dataset, there are always displayed all cities in the embedding as points, the target city is marked as a blue point and it is randomly selected in the beginning of each searching session. This searching method requires to specify at least two parameters, which were determined to $\theta_1^+ = 0.8$ and $\theta_2^+ = 0.001$. Since these parameters were estimated empirically for the embedding using its distance histogram, they are suboptimal. The searching session has been initialized 5 times. One example of the searching session is displayed in Figure 6.6.

Results. In each step of the searching process the cities chosen to the *display set* are colored green and the chosen city from display set is color cyan. The gradient color denotes the probability of each image to be selected into the display set: the lighter color the higher probability. The average number of steps necessary to hit the target object was 6 over the 5 iterations. There is no important conclusion from this experiment, which was added to this work just as a visual demonstration of the mental matching algorithm.

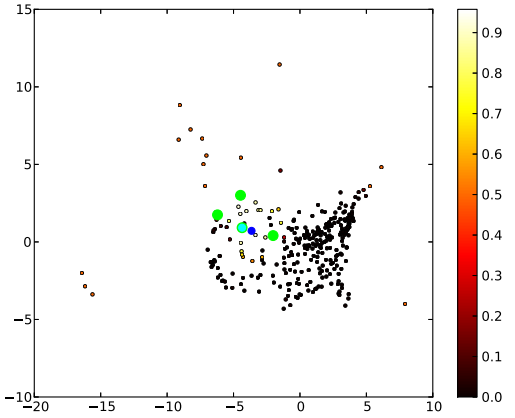


(a) step 0 - initial state

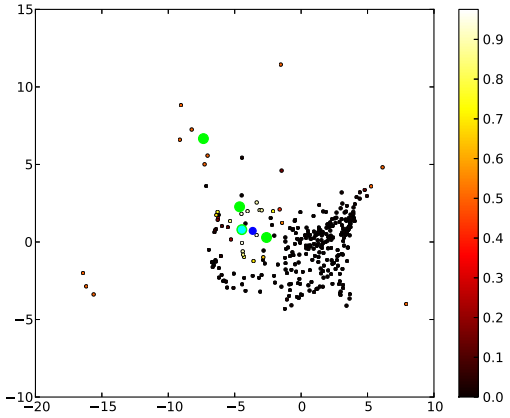
(b) step 1



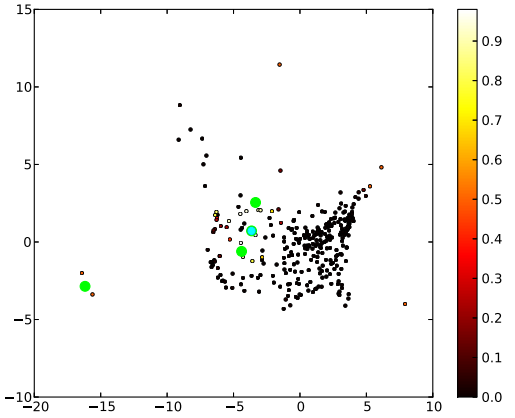
(c) step 2



(d) step 3



(e) step 4



(f) step 5 - target hit

Figure 6.6: The example of a searching session using the mental matching algorithm. The blue point represents the target city, green points cities selected to display set and the cyan one the city from display set that is closest to the target one. Gradient color determines the probability of eligibility to display set.

Chapter 7

Perceptual Similarity Evaluation on a Country Flags Dataset

The importance of this experiment is to demonstrate inconsistency and high divergence of responses when the workers are asked to numerically evaluate similarity of two objects. On the other hand, if they are asked to compare pairs of objects, their answers are much more consistent. The following experiments were performed on a country flags image dataset.

Country flags dataset. The country flags dataset contains 196 images of countries recognized by member states of United Nations, United Nations observer states, and Taiwan (according to Wikipedia[1]). The flags were in the following experiment used as objects that are perceptually similar and the ground truth similarity is not known.

For purposes of this experiment the problem formulation was taken from [18]. Assume we are given a set of inputs $\{z_1, \ldots, z_n\} \subset \mathcal{Z}$. There is a ground truth dissimilarity function $\delta(z_i, z_j)$, which evaluates the dissimilarity of two items z_i and z_j. This function $\delta(\cdot)$ is not known to us.

Hypothesis. In order to get the dissimilarity function δ we could ask users "how similar is object z_i to z_j" on Likert scale [12]. If multiple user are asked with the same question using the same objects z_i and z_j, their answers will be inconsistent and biased. The cause of this problem may arise from different respondents' experience and their different internal scales. Imagine two images with food. One with a seafood and another with a grilled chicken breast. If we showed these two images to a person who had never tried seafood and ask her how similar those foods on the images taste from 0 (completely differently) to 10 (identically), the score could be quite high, because she might think that both are meat (the origin is animal) and therefore it should taste similar. On the other hand, if we asked the same question somebody experienced by a seafood and grilled chicken breast, we would probably get much lower score than in the first case.

7.1 Experiment Setup

The aim of this experiment was to confirm the previously mentioned hypothesis. There were randomly sampled 10 triplets of flags from the dataset, which were subsequently used

[1] http://en.wikipedia.org/wiki/List_of_sovereign_states

(a) Numerical evaluation interface.

(b) Triplet interface.

Figure 7.1: The screen shots of the numerical evaluation of similarity and the triplet interface displayed to the workers in order to obtain their answers.

in the following two parts of the experiment.

In the first part, 10 randomly chosen pairs were taken from the triplets and presented to 10 workers that were asked to evaluate the similarity of each pair of flags. The same 10 pairs were shown to each worker, just the order of pairs was chosen randomly and workers were not allowed to return to previous pairs they had already answered. The aim of this random order was to demonstrate, that the results may be also affected by the order of pairs. Furthermore, each worker was allowed to answer just one set of these ten pairs. The workers were on each screen asked to evaluate the similarity of the displayed flags on a scale from 0 (completely different) to 10 (absolutely similar). A screenshot from the user interface presented to the workers is in Figure 7.1a.

In the second part of the experiment there were used all 10 sampled triplets. Each triplet was randomly divided to a probe image and a grid (remaining 2 images), and there were also 10 workers asked to select the most similar flag from the grid to the probe flag using the selection task on the default triplet template $T^{\text{sel}}_{\text{gp}}(2, 1)$. To all 10 workers were presented the same screens, just the order of screens was shuffled, and as well as in the first part, each worker was allowed to complete just one set of the 10 screens. User interface used in this part is shown in Figure 7.1b.

Both parts of the experiment were deployed to Amazon MTurk in order to receive answers from workers. The workers were in both cases paid \$0.05 per HIT (set of screens), and there were applied the following restrictions to assure the reliability of answers. To work on this task, there were allowed just workers, who are US residents, had already had at least 1 000 HITs approved, and at least 95% of their submitted HITs had been approved.

7.2 Results

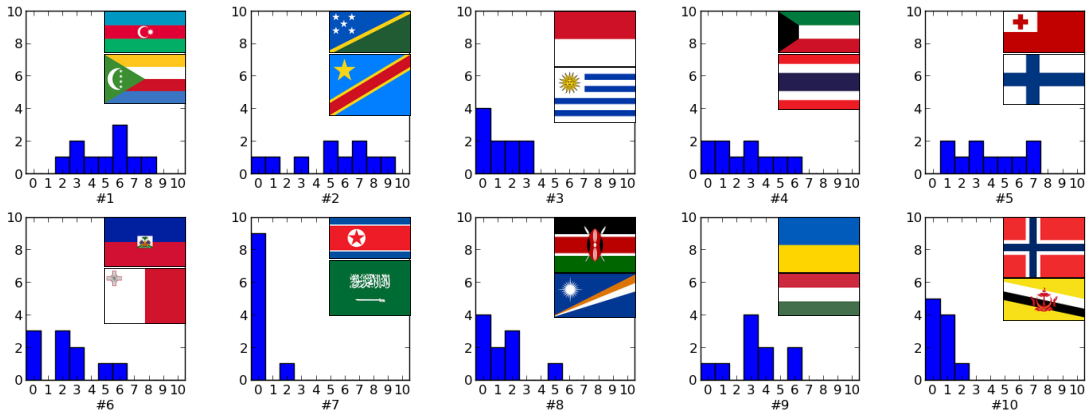
For the first part of the experiment there are plotted histograms of workers' answers separately for each screen. The results are displayed in Figure 7.2a. On the horizontal axis of each histogram there are numerical evaluations of similarity (from 0 to 10) and on the vertical axis the quantity of answers. Each histogram has also attached flags, which the workers were comparing. It is apparent from the histograms that the numerical evaluation of similarity is in some cases very outstretched and uniform (screens #1, #2, #4, and #5), however in a case of an ideal workers' answers, there would be just one value presented. Another obvious fact is that most of answers are pessimistic (the distribution of answers is

shifted to the lower values).

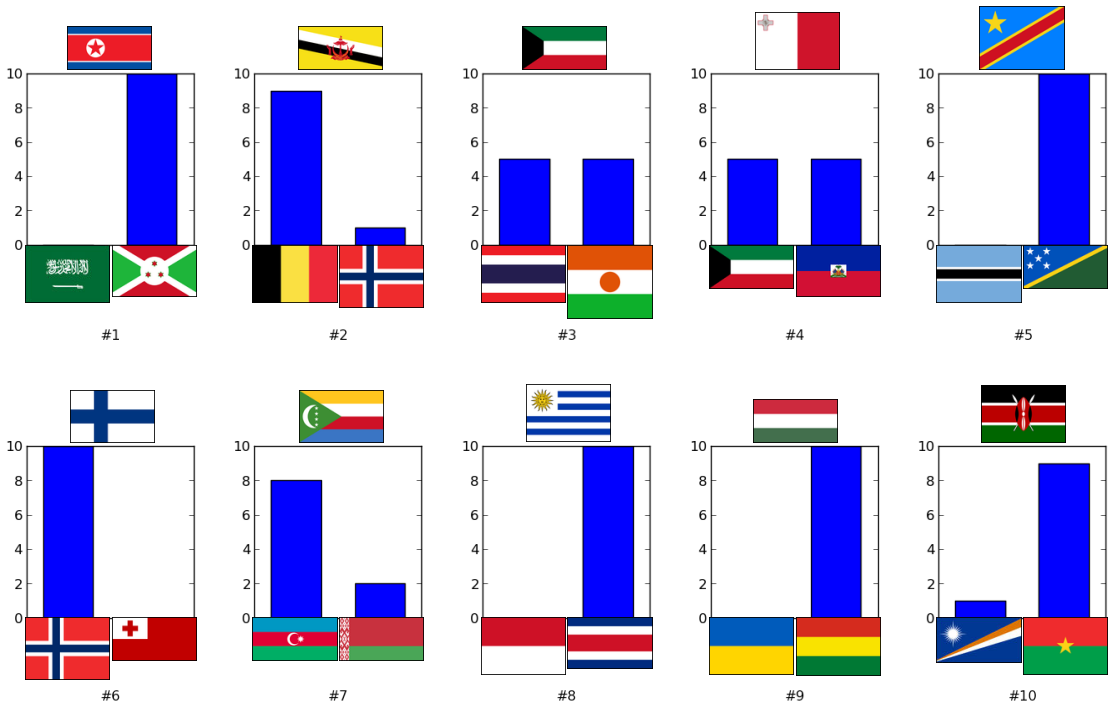
The resulting histograms of the worker's answers for the second part of the experiment are displayed in figure 7.2b. The flag images displayed on the top of each histogram are the probe images of triplets presented to the workers, the images at the bottom are the another two images from triplets that could be selected as "more similar" to the probe. On the "y" is again displayed the quantity of answers. In the half of triplets presented to the workers are their answers uniform and there are just two triplets, where the answers are balanced and hence is not possible to decide, which of the flags is more similar to the probe.

Discussion. As we assumed in the hypothesis, if the users are asked to evaluate some perceptual similarity of two objects numerically, their answers will vary noticeably. This statement is supported by the results of the first part of this experiment, where 10 workers on MTurk were asked to evaluate the similarity of two given flag images. The answers were in the majority of flags pairs distributed over more then half of the scale range. In most cases the distribution of answers also does not correspond to the normal distribution.

The second part of the hypothesis assumed that usage of triplets instead of numerical evaluation should improve the consistency and reliability of answers. This assumption was confirmed in the second part of the experiment, where the workers were presented triplets with flag images and asked to select the most similar flag to the probe flag. Their answers were in 80% of triplets very clear and in the remaining 20% there were balanced and thus it is not possible to deduce a "correct" answer in those cases.



(a) Results of numerical evaluation of flags similarities: Histogram of answers for each pair of flags displayed to workers.



(b) Results of flag triplets: Histogram of answers for each triplet displayed to workers. Flags on the top of sub-figures were the query images (probes) and from the flags at the bottom the user chose the most similar one to the probe.

Figure 7.2: Obtained results from the both parts of the experiment: The results from numerical evaluation are displayed in the top row, results for triplet comparison in the bottom one.

Chapter 8

Experiments on Food Images

This chapter describes application of acquired knowledge on a real dataset of food images. There is performed a complete pipeline starting with similarity retrieval process, continuing through triplet filtration and embedding creation and ending with evaluation of searching performance in the space of embedding.

Food-1000 dataset. Food-1000 is an image dataset containing 1 000 randomly chosen images from Yummly[1]. Along with these images, the dataset contains several meta data as names of meals, ingredients, and tastes.

8.1 Similarity Retrieval

The images from the Food-1000 dataset were uploaded to Visipedia: Crowdwork application (see Section 3.3.2) in order to obtain similarities among them, in particular triplet comparisons. The target number of needed triplets is based on the results of the experiment described in Section 6.6. The goal was to obtain at least 0.015% of all triplets, which roughly corresponds to 5% of embedding error.

According to the investigation of template performance (see Section 6.4) and the research done by authors in [25], one of the most effective templates for triplet generation, which was also used in this experiment, is the template $T^{\text{sel}}_{\text{gp}}(12, 4)$. In other words, a probe image and a grid of 12 images were presented to workers and their task was to select 4 images from the grid that are most similar to the probe image. There can be produced 32 triplets from every such answer. The goal was to obtain at least 0.015% of all triplets, which is in this particular case

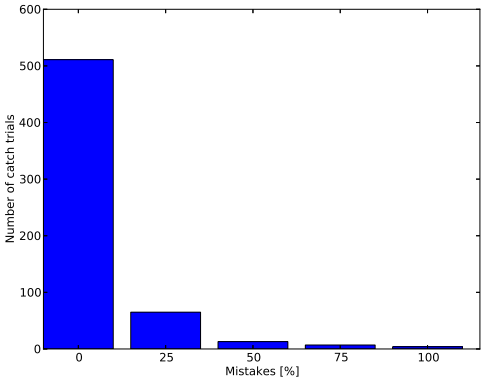
$$0.00015 \cdot 1\,000 \cdot \binom{999}{2} \doteq 74\,775 \tag{8.1}$$

triplets, so the necessary amount of screens presented to workers is

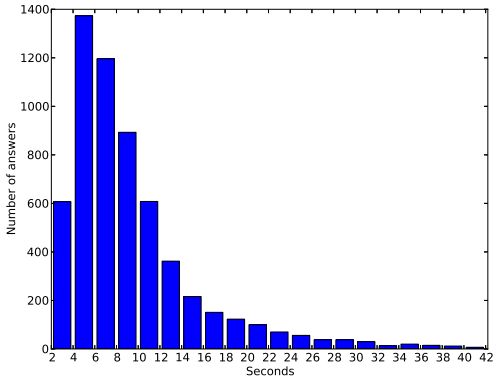
$$\left\lceil \frac{74\,775}{32} \right\rceil = 2\,337. \tag{8.2}$$

The screens were divided into sets of 20: 18 regular screens and 2 catch trials for quality assurance reason. Adding catch trials arose the necessary amount of screens by 10%. Also

[1] http://yummly.com/



(a) Mistakes in catch trials.



(b) Time histogram with the median time 7.64 s.

Figure 8.1: Statistics from the workers' answers: Mistakes in catch trials and a screen timer histogram.

all answers are based on a perceptual property and they are produced by human workers, so it is very likely that some inconsistencies appear in the answers and hence some redundant screens should be added. One possibility would be to display the same screens to multiple users and subsequently merge their answers, but this solution would increased the costs per triplet. Since the t-STE method deals with inconsistencies quite well and from the partial conclusion that quality of triplets can be compensated by quantity (see Section 6.4), instead of repetitive screens there were just added more screens. So, in the end the goal was determined to 6 000 screens (300 sets). Each set was deployed to MTurk as a separate HIT. Eligible workers were those, who were US residents, had already had at least 1 000 HITs approved and at least 95% of their submitted HITs had been approved. The reward for a HIT was determined to \$0.2 per HIT.

There are some important statistical data from workers' answers, that might be used for their filtration. In Figure 8.1 there is a histogram of mistakes in catch trials and a histogram of time needed to answer a single screen. The median time was 7.64 s. A research in [25] subsumed a similar data collection on a different dataset, where the authors achieved the median 8.67 s per screen using the same template $T^{\text{sel}}_{\text{gp}}(12, 4)$. This difference is probably caused by usage of different datasets. Next statistics about the similarity retrieval task are summarized in the following table:

Number of images	1 000
Number and type of used template	6 000 screens (300 sets) $T^{\text{sel}}_{\text{gp}}(12, 4)$
Quality assurance	10% of screens were catch trials (totally 10 distinct)
Number of obtained triplets	192 000 (0.0385% of all triplets)
Total costs	\$66 (\$60 rewards and \$6 Amazon commission)

8.2 Embedding Construction

Once the answers were collected, the next step was to construct an embedding of the food images. From the possible embedding algorithms that could be potentially used was chosen t-STE, which gives best results on noisy data with some contradicting triplets.

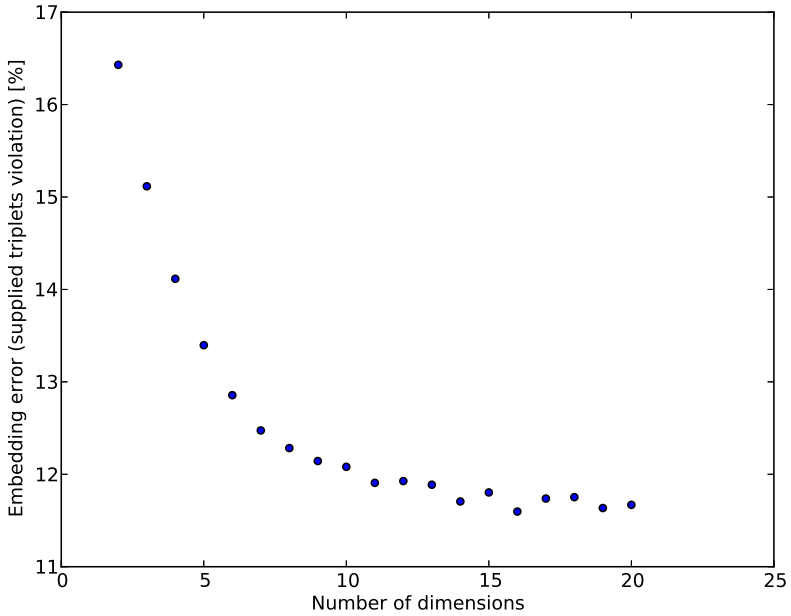


Figure 8.2: Error of food image embeddings with different number of dimensions. The embedding errors was computed from supplied triplets.

At first, it was necessary to determine a number of dimensions. For each dimension $d = 2, 3, \ldots, 20$ there were created 10 embeddings and computed a mean embedding error from supplied triplets (see Section 6.3) over these 10 iterations. The resulting mean errors for all dimensions are displayed in Figure 8.2. According to results it seems reasonable to select any dimension around 10. In fact, there are two factors affecting this choice: computational complexity and embedding error; higher dimensionality decreases embedding error, but increases the computational complexity. These led to choose the number of dimensions $d = 8$, because the embedding error decreases very slowly for higher numbers.

The next step was to select which triplets to use for the embedding construction. There were collected 192 000 triplets, which is 0.0385% of the triplet universe, but, obviously, there are some triplets eligible to be filtered out, or rather the answers from which these triplets were generated. The reasons why it is advantageous to filter out some answers are as follow:

Duplicates. Since 10% of all screens presented to workers were catch trials and the amount of distinct catch trials is relatively small, there is a lot of identical catch trial screens and hence a lot of identical triplets. Using t-STE method which pushes together and pulls apart objects even when the presented triplet is already satisfied, the food images that are part of triplets generated from catch trials would be disproportionately close together or far away with respect to the rest of images.

Incorrect Catch Trials. The next filtration criteria concerns these sets, in which the worker answered catch trials incorrectly, and hence also answers on regular screens can be very likely biased. On the other hand, filtering out all sets with any mistake would reduce

ID	Filtration criteria		Filtered out objects		Error
N	No filtration		0	(0%)	12.3%
D	Duplicated answers		18 487	(9.63%)	13.29%
C1	Incorrect catch trials	any mistake in a set	50 560	(26.33%)	10.17%
C2		2+ mistakes in a set	18 560	(9.66%)	11.51%
C3		3+ mistakes in a set	7 040	(3.67%)	12.06%
T1	Timer-based filtration	< 3 s	3 264	(1.7%)	12.1%
T2		< 4 s	19 520	(10.17%)	11.75%
T3		< 5 s	42 112	(21.93%)	11.23%
T4		> 10 s	61 760	(32.17%)	10.38%
T5		> 20 s	15 040	(7.83%)	12.11%
T6		> 30 s	5 408	(2.82%)	12.2%
X1	Selected combinations	D, C2	35 383	(18.42%)	12.52%
X2		D, C2, T1	37 569	(19.57%)	12.36%
X3		D, C2, T1, T6	42 436	(22.1%)	12.22%
X4		D, C2, T2	48 047	(25.02%)	11.86%
X5		D, C2, T2, T6	52 914	(27.56%)	11.73%

Table 8.1: Selected filtration options and their impact to a resulting number of triplets and the 8-dimensional embedding error.

significantly the amount of triplets (see Table 8.1), so there should be given some tolerance.

Time-based Filtering. According to the time histogram of workers' answers (see Figure 8.1b) there are some screens which were answered quite fast or slow. The fast answers can be explained quite simply: workers on MTurk just try to make as much money as they can, but sometimes it can degrade the quality of answers. The slow answers can be caused by different reasons: Workers may think a long time about the best answer or they might be focused on something else (even though this situation should be eliminated by stopping a timer when the worker is idle).

Although the filtration of some answers reduces the amount of triplets used for embedding construction, it should increase the quality of the remaining triplets. Several filtering criteria and their combination are summarized in Table 8.1.

In this particular dataset the ground truth function is not known, so there is no way how to exactly measure the quality of an embedding or how to compare two embeddings (besides the embedding error function, that measures a percentage of satisfied triplets from with respect to all provided triplets, but is does not tell much about the ground truth function). One way how to compare embeddings at least visually is to display nearest neighbors to selected objects and compare their relevance. Since all embeddings contains the same objects, the nearest neighbors of a particular object can be compared over several different embeddings. There were chosen 10 images trying to cover a wide spectrum of different types of food and computed 5 nearest neighbors for each of them. A comparison of such nearest neighbors for filtration options N and X4 is displayed in Figure 8.3.

Comparing the nearest neighbors with the reference object can give us an idea about about quality of localization in the embedding. It is apparent from the previously mentioned figure, that lot of similar meals ale located close to each other, but also on the other hand there are some types of meals, that are placed in the embedding quite randomly. There are

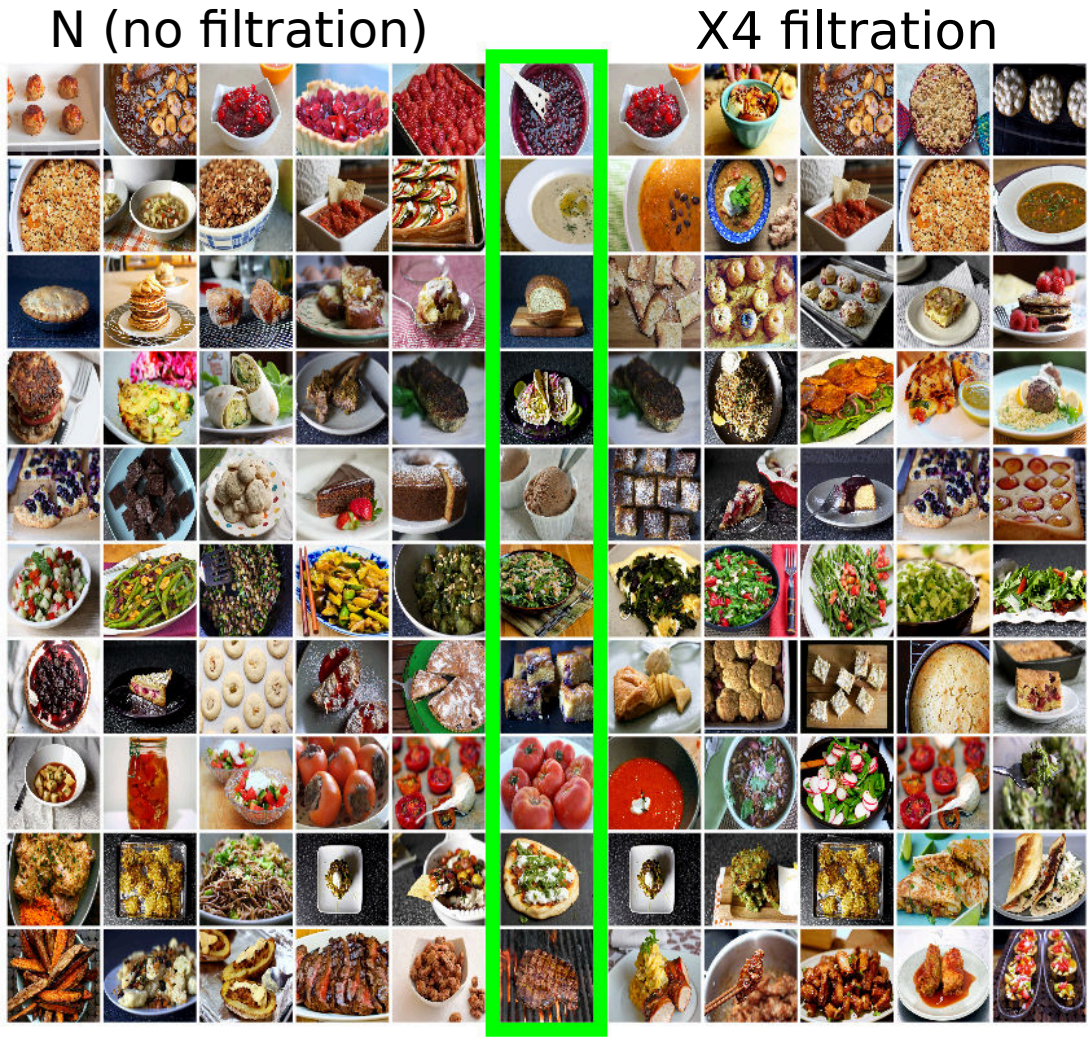


Figure 8.3: Nearest neighbor meals in embeddings. The left section comes from an embedding without filtration and for the right one was used filtration option X4 (see Table 8.1). Then each line shows the 5 nearest neighbors from the embedding to the image in the green box. The farther from the green box, the less similar image.

some reasons that might caused, that some objects are embedded better and some worse:

- The images were selected randomly to the grid during the similarity retrieval, so some of them appeared in more screens than the others and hence there are presented in more triplets.
- There is a wide range of numbers of similar meals to some reference one. For example, there is a lot of salads or cakes, but just a few representatives of sushi or French fries.

The other interesting fact is, that comparing nearest neighbors of the embedding created from filtered triplets and of the embedding from all triplets, there is no notable improvement, even the embedding from all triplets could be considered as a better one, because on some lines (1, 8) the nearest neighbors seem to be more relevant then in case of the other embedding.

(a) θ_1^+ determination.

(b) θ_2^+ determination.

Figure 8.4: Screenshots of the layouts used for determination of parameters θ_1^+ and θ_2^+ in mental matching algorithm.

8.3 Searching in Embedding

Searching procedure in an embedding is based on the mental matching algorithm (see Section 4.2). The method takes an embedding on its input along with four parameters affecting a behavior of $\phi_+(d)$ and $\phi_-(d)$ functions and a parameter specifying the size of the display set. The optimal size of display set, that is not too large for users, was determined by authors of [9] to 8, but it can be accommodated to a particular use case or a screen size. The parameters for negative model $\phi_-(d)$ can be fixed to $\theta_1^- = 0$, $\theta_2^- = 1$. For the positive model, the parameters was estimated according to the algorithm presented in Section 4.2.2.

8.3.1 ϕ_+ Parameters Determination

The algorithms for object selection for both parameters θ_1^+ and θ_2^+ were implemented into *Visipedia: Crowdwork* application. At first, a task to collect data required to compute θ_1^+ was created and deployed on MTurk. Once the θ_1^+ parameter was computed, the second task for θ_2^+ that already required θ_1^+ was created and deployed to MTurk.

Both previously mentioned algorithms require to pick a target class S as well as one object $k \in S$ randomly. Although the Food-1000 dataset contains some meta data like meal name, ingredients, etc., there are no classes of these meals. Also if they were, the aim of this thesis is not to use a deterministic number of classes, but rather a continuous space. Therefore, the issue was how to determine a target class. Considering Figure 8.3 with nearest neighbors, many similar meals that could be even classified to the same category are grouped together. And since the algorithm shows workers the overview of the target class instead of a single image, it handles also a situation, when a really different food is located among the nearest neighbors. The target group was hence formed from a randomly picked food image and its 8 nearest neighbors in the embedding, so the size of the target group was 9. This size was selected because it can be ordered in a 3×3 grid which fits the layout pretty well. Then, the image k was randomly chosen from the target class. The layouts of the tasks deployed to MTurk are diplayed in Figure 8.4. It is worth to mention that the embedding space is generally unbounded, but the mental matching algorithm works with

θ	0.1	0.15	0.2	0.25
$N(\theta)$	76	50	27	13
$p(\theta)$	9.96×10^{-8}	0.5	≈ 1	≈ 1

(a) No triplet filtration (100 samples).

θ	0.2	0.25	0.3	0.35	40
$N(\theta)$	43	30	29	27	18
$p(\theta)$	0.0004	0.5	0.6	0.78	≈ 1

(b) X4 triplet filtration (60 samples).

Table 8.2: θ_1^+ parameter determination for two embeddings: with no triplet filtration and with X4 triplet filtration. We choose $\theta_1^+ = 0.1$ in case of no triplet filtration and $\theta_1^+ = 0.2$ for X4 filtration.

normalized distances $\delta \in [0, 1]$ between objects. Therefore all distances from the embedding are at first divided by the largest distance.

The procedure of parameter adjustment was repeated for both embeddings: with no triplet filtration and with X4 triplet filtration. These embedding will be denoted E_N and E_{X4}. The whole procedure consists of two steps. In the first one, there was determined parameter θ_1^+, which was subsequently used in the second step to get θ_2^+. In Table 8.2 there are presented results obtained from workers for the first step. For E_N, there was chosen $\theta_1^+ = 0.1$ and for E_{X1} parameter $\theta_1^+ = 0.2$. These parameters represent the largest values, where the null hypothesis is rejected at 0.05 significance level (see Section 4.2.2). In the second step, there were obtained the following results:

- For E_N, the images i has been chosen in 204 cases from 400 answers, which gives $\theta_2^+ = 0.137$,
- in case of E_{X4}, i has been chosen 221 times from 400 answers, hence $\theta_2^+ = 0.116$.

8.3.2 Performance Evaluation

The performance of searching in the space of embedding was measured in the similar way as authors of the used mental matching method [9]. The only difference is that they used labeled datasets, so they determined the target class on a basis of its label. In the case of Food-1000 dataset, the target class was determined by a cluster of to a randomly selected item (target image) from the embedding and its 8 nearest neighbors, which gives together 9 images in target class S. There were collected and compared results for three different user models:

- *Ideal user* always chooses the closest image to the target set.
- *Random user* acts in every step randomly.
- *Real user* represents a sample of human users.

Whereas the ideal and random user model, which were computed programmatically, the real user model was calculated on MTurk workers. There was used the identical template as in case of θ_2^+ parameter determination process (see Figure 8.4b). The target class was displayed all the time and in every iteration there was shown the new display set and the worker was asked to keep selecting the closest image from display set to the target class until at least one food image from the target class appeared in display set or exceeded maximum number of iterations.

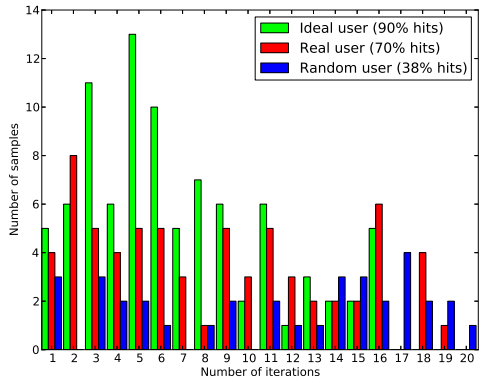
The experiment was repeated 100 times for all three user models and both embeddings (E_N and E_{X4}) and in each run, there was wrote down the necessary number of iteration

when any image from the target set appeared in the display set. The maximum number of iterations for each run was limited to 20. When no image from the target set was reached within 20 iterations, the searching was marked as unsuccessful. The resulting histograms of necessary number of steps as well as cumulative distribution functions for all user models and both embeddings are displayed in Figure 8.5.

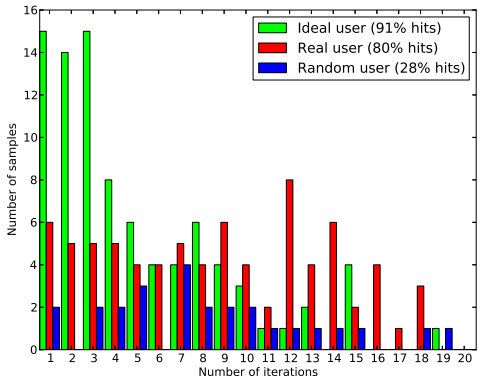
Discussion. Although the E_N embedding looked probably better than E_{X4} in terms of the nearest neighbors (see Figure 8.3), according to the results from mental matching experiment it gives a worse performance. Besides the quality of embedding it could be caused also by suboptimal determination of parameters for positive answer model, however, the determination process was identical for both embeddings so this assumption might be misleading.

Comparing the histograms of samples from ideal user, it is apparent that in case of E_{X4} the high values on the vertical axis are shifted towards the smaller values on horizontal axis with respect to the E_N embedding. This implies, that more searching sessions finished earlier, which indicates a better performance. This is also apparent from the shape of cumulative distribution function. In case of real user, the difference is not so noticeable for early steps, but it become apparent in later steps of the searching session. There is 10% improvement in number of tasks which finished within 20 iterations.

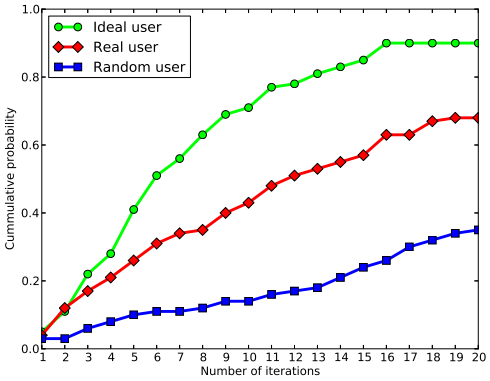
The searching performance achieved on the food dataset can be also compared with the experiments presented in [9]. Even though the performance achieved on food image dataset is not as good as in their experiments on Corel and Alinati databases, it does not indicate failure, because in their cases, the embeddings were created using computer vision methods, in our case they were constructed from humans' answers which were based on perceptual similarity.



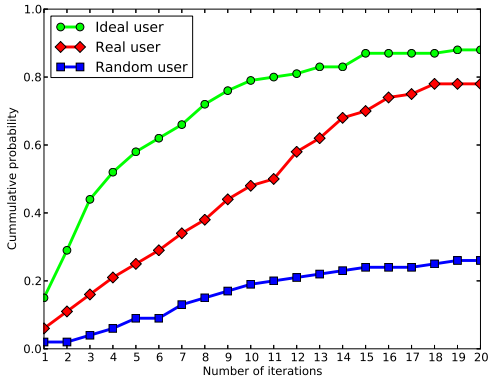
(a) Histogram of samples for E_N.



(b) Histogram of samples for E_{X4}.



(c) Cumulative distribution function for E_N.



(d) Cumulative distribution function for E_{X4}.

E_N	20%	40%	60%	80%
Ideal user	3	5	8	13
Real user	4	9	16	N/A
Random user	14	N/A	N/A	N/A

E_{X4}	20%	40%	60%	80%
Ideal user	2	3	6	11
Real user	4	9	13	N/A
Random user	11	N/A	N/A	N/A

Figure 8.5: Performance of the mental matching algorithm achieved on Food-1000 image dataset for ideal user, real user and random user expressed by the sample histogram, cumulative distribution function, and the number of steps required to finish searching in 20, 40, 60, and 80 percent of cases. The left column refers to E_N embedding, the right one to E_{X4} embedding.

Chapter 9

Conclusions

The aim of this work was to propose an efficient way of searching among perceptually similar objects, which subsumed an investigation in perceptual similarity retrieval from humans, choosing an appropriate method for embedding learning, and adaptation of the used searching algorithm to the particular embedding, in order to extend the Visipedia project. This target has been met.

In the work, there are described and divided types of relations between objects. A special focus is put on investigation of effective ways, how to obtain objects comparison on the basis of their perceptual properties. Since this step includes collaboration with humans, there is presented the idea of crowdsourcing and there are proposed recommendations how to get the most from this interaction. There are also presented concepts of triplets and paired comparisons, together with techniques of humans' feedback transformation into them. In the theoretical part, there are introduced and compared methods for embedding construction and navigation in such space, which are subsequently applied on real data. There are presented three datasets on which the experiments has been performed: A toy dataset of cities in the US and Canada, a country flags dataset, and a food image dataset. The resultant searching process is demonstrated on a simple application.

In the presented experiments, there were achieved several remarkable results. Although the amount of triplets grows cubically w.r.t number of objects, it seems that the number of triplets required to build the embedding with constant error grows much slower – maybe even linearly. The next one measures the necessary amount of rounds while searching in the perceptual embedding. The searching session finishes for average human user in 9 steps for 40% and in 13 steps for 60% of all searching sessions using triplet filtration.

There are several possible directions for a future work. Since the performance of searching in an embedding is strongly correlated with its quality, some research should be focused on embedding quality improvement and its measurement. In particular it comprise an investigation in alternative ways of interaction with humans in order to obtain object similarities as well as in methods determining which triplets are important to improve embedding quality. Adding a new object into embedding requires its rebuilding, so there is also room for finding ways how to force en embedding to be locally-oriented and hence simplify the process of a new object incorporation. It would be also interesting to determine a complexity of the function expressing a growth of necessary amount of triplets with respect to a number of objects in the embedding preserving its quality, as was sketched out in this work. There is also a lot of work on *Visipedia: Crowdwork* application which functionality should be extended and the application seamlessly connected to the whole Visipedia system.

Bibliography

[1] S. Agarwal, J. Wills, L. Cayton, G. Lanckriet, D. Kriegman, and S. Belongie. Generalized Non-metric Multidimensional Scaling. In *AISTATS*, San Juan, Puerto Rico, 2007.

[2] Amazon Web Services, Inc. *Amazon Mechanical Turk: Getting Started Guide*, March 2012.

[3] P. Belleflamme, T. Lambert, and A. Schwienbacher. Crowdfunding: tapping the right crowd. CORE Discussion Papers 2011032, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2011.

[4] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer Series in Statistics. Springer, 2005.

[5] M. Buhrmester, T. Kwang, and S. D. Gosling. Amazon's Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data? *Perspectives on Psychological Science*, 6:3–5, 2011.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image. In *CVPR09*, 2009.

[7] E. Estellés-Arolas and F. G.-L. de Guevara. Towards an integrated crowdsourcing definition. *Journal of Information Science*, 38:189–200, 2012.

[8] Y. Fang and D. Geman. Experiments in mental face retrieval. In *Proceedings AVBPA 2005, Lecture Notes in Computer Science*, pages 637–646, 2005.

[9] M. Ferecatu and D. Geman. A statistical framework for image category search from a mental picture. *IEEE Trans. PAMI*, 31:1087–1101, 2009.

[10] K. G. Jamieson and R. D. Nowak. Low-dimensional embedding using adaptively selected ordinal data. In *Allerton Conference on Communication, Control, and Computing*, 2011.

[11] J. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Information Science and Statistics. Springer, 2007.

[12] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 140:1–55, 1932.

[13] W. Mason and D. J. Watts. Financial incentives and the „performance of crowds“. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '09, pages 77–85, New York, NY, USA, 2009. ACM.

[14] T. Matera, J. Jakes, M. Cheng, and S. Belongie. A User Friendly Crowdsourcing Task Manager. 2014.

[15] G. Paolacci, J. Chandler, and P. G. Ipeirotis. Running experiments on Amazon Mechanical Turk. *Judgment and Decision Making*, 5:411–419, 2010.

[16] P. Perona. Visions of a Visipedia. In *Proceedings of the IEEE August 2010*, volume 98, pages 1526–1534, August 2010.

[17] O. Tamuz, C. Liu, S. Belongie, O. Shamir, and A. T. Kalai. Adaptively Learning the Crowd Kernel. *CoRR*, abs/1105.1033, 2011.

[18] L. van der Maaten and K. Weinberger. Stochastic Triplet Embedding. In *2012 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING*, Santander, Spain, 2012.

[19] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI '04 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 319–326, 2004.

[20] L. von Ahn and L. Dabbish. Designing Games With a Purpose. *Communications of the ACM*, 81:58–67, 2008.

[21] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science 12*, 321:1465–1468, 2008.

[22] C. Wah. Crowdsourcing and its applications in computer vision. UCSD CSE Research Exam, UC San Diego, 2011.

[23] C. Wah, S. Branson, S. Maji, P. Perona, and S. Belongie. Similarity Comparisons for Interactive Fine-Grained Categorization. 2014.

[24] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-ucsd birds 200. Technical Report CNS-TR-201, Caltech, 2010.

[25] M. J. Wilber, I. S. Kwak, and S. J. Belongie. Cost-Effective HITs for Relative Similarity Comparisons. 2014.

[26] S. K. M. Yi, M. Steyvers, M. D. Lee, and M. J. Dry. The wisdom of the crowd in combinatorial problems. *Cognitive Science*, 36:452–470, 2012.

Appendix A

CD Content

The CD attached to this thesis contains the following file structure:

- `/data` – datasets, answers from workers, triplets, and embeddings,
- `/demo` – simple application demonstrating searching in the embedding space,
- `/scripts` – scripts used in the experiments,
- `/thesis` – PDF file of this thesis and its LaTeX source code,
- `/README` – the CD file structure.