Maximizing color difference in metro maps

Bachelor thesis Mathematics
July 2014
Student: S.F. Griffioen
First supervisor: Dr. A.V. Kiselev
Second supervisor: Prof. dr. H. Waalkens
Source of frontpage image:
http://cdn.theatlanticcities.com/img/upload/2013/05/22/moscow_metro.jpg
RIJKSUNIVERSITEIT GRONINGEN

Abstract

Faculty of mathematics and natural sciences

Bachelor of Mathematics

Maximizing color difference in metro maps

by Simone Griffioen

In this thesis we consider the colors used in metro maps. Usually each line has its own color, and for obvious reasons the colors should be maximally different from each other. We present different methods to find an optimal color for a new line on the map. We discuss the physiology of the eye, different color spaces and color metrics. We work in the CIELAB space and we use both the simple Euclidean CIE76 color difference formula and the better but more complicated CIEDE2000 color difference formula. To find an optimal color for a new line, we must maximize the minimal distance between the new color and all the existing ones. We present different methods for this, based on our different color difference formulas. To maximize the minimal CIE76 distance, we calculate vertices of Voronoi cells. This method is improved by using the CIEDE2000 formula to select an optimal color from the candidate colors. To find an optimal color also random points can be used to come close to an optimal point. A method using Coulomb repulsion is suggested. For adding more than one color at the same time different methods are discussed. Adding colors one by one does not always give us an optimal result. The random point method is extended for adding more colors at the same time. Standard optimization algorithms can also be used for this purpose. We apply some of these methods to the metro map of Moscow. We suggest new colors for the new lines the Moscow metro is planning to build before 2020. We suggest ideas for follow up research that can be done on this subject.

Keywords: Optimization, Maximizing minimal distances, Voronoi diagrams, Color metrics, Moscow metro

MSC 2010 codes: 00A66, 65K10, 37C10, 37C50
Acknowledgements

I thank Arthemy Kiselev for being my preceptor and especially for suggesting this wonderful subject for my thesis. I have liked it very much to work on such a concrete subject that combines mathematics with other fields of science. Our brainstorm sessions about the subject and possible methods have often given me new insights which helped me get past some problems that seemed enormous to me. I also express my appreciation to professor Waalkens for being my second supervisor. Besides this, I thank all professors and other university teachers that have provided me with the mathematical knowledge and skills I could all combine in this thesis. I have learned a lot in the past three years and I hope to learn even more in my master’s. Finally I am grateful to everybody who came to listen to my talk about this thesis. I would never have expected to have such an audience!
Contents

Abstract ii

Acknowledgements iii

1 Introduction 1
  1.1 The Moscow metro ................................. 1
  1.1.1 Future developments ............................ 3
  1.2 The problem .................................. 3
  1.3 Outline of the thesis ............................ 3

2 Measuring color 5
  2.1 Seeing color .................................. 5
  2.1.1 Construction of the eye ....................... 5
  2.1.2 How do we see color? ......................... 6
  2.2 Color differences ................................ 7
  2.2.1 RGB space .................................. 7
  2.2.2 CIELAB space ............................... 8
  2.2.3 Color difference formulae ................. 8

3 Solution of the general problem 11
  3.1 Adding one point ................................ 11
  3.1.1 Mathematical formulation of the problem .... 11
  3.1.2 The Voronoi diagram ........................ 11
  3.1.3 Bounded Voronoi diagram .................... 12
  3.1.4 CIE76 Voronoi method ....................... 13
  3.1.5 CIE76-CIEDE2000 Voronoi method ........... 14
  3.1.6 Voronoi diagrams for non-Euclidean metrics 14
  3.1.7 Brute force method .......................... 15
    3.1.7.1 Probability of hitting an optimum .... 15
  3.1.8 Random walk method ........................ 16
  3.1.9 Coulomb repulsion method ................. 16
    3.1.9.1 Coulomb repulsion ........................ 16
    3.1.9.2 The model ............................... 17
  3.2 Adding more than one point .................... 18
  3.2.1 One by one Voronoi method ............... 19
  3.2.2 Nonsmooth optimization ................... 19
3.2.3 Brute force method ........................................... 21
3.2.4 Coulomb repulsion ........................................... 21

4 Application to the Moscow metro map .......................... 23
  4.1 The colors of the Moscow metro .............................. 23
  4.2 Finding optimal colors by the CIE76 Voronoi method ....... 23
    4.2.1 More new colors: One by one method ................... 27
  4.3 Finding new colors by the CIE76-CIEDE2000 Voronoi method........ 28
  4.4 Finding new colors by the brute force method ............... 28
    4.4.1 One by one ........................................ 28
    4.4.2 Adding two points simultaneously ..................... 29
  4.5 Finding two new colors by the build in Matlab function fmincon .... 29
  4.6 Finding new colors by the one by one Coulomb repulsion method .... 30
  4.7 Conclusions .............................................. 30
  4.8 Discussion ................................................. 31

A CIEDE2000 color difference formula .......................... 33

B Matlab code ..................................................... 35
  B.1 Calculate the sideplanes .................................... 35
  B.2 Calculate the minimal distance to all existing points ....... 36
  B.3 Calculate vertices with CIE76 method ....................... 37
  B.4 Calculate vertices with CIE76-CIEDE2000 method ............ 38
  B.5 Plot the colors in Cielab ................................... 40
  B.6 One by one brute force method ............................. 41
  B.7 Simultaneous brute force method ............................ 41
  B.8 Coulomb repulsion with CIEDE2000 ........................... 42
  B.9 Solve optimization problem with fmincon .................... 43

Bibliography ..................................................... 45
Chapter 1

Introduction

1.1 The Moscow metro

Opened in 1935 with 11 kilometer metroline and 13 stations, the Moscow metro was the first underground railway system in the Soviet Union [1]. Nowadays the Moscow metro has 194 stations and its total route length is 305.5 km. In 2012 the Moscow Metro was the world’s second most heavily used rapid transit system after the Seoul metropolitan subway.

The Moscow metro has 11 lines, schematically shown in Figure 1.2. This Figure is called the metro map of the Moscow metro. On the metro map, every line has a color. This way customers can easily recognise the lines on the map.

Figure 1.1: Mayakovskaya station. Photo: N Moulds.
Figure 1.2: The map of the Moscow metro.
1.1.1 Future developments

In Figure 1.3, the development plans for the Moscow metro up to 2020 can be found [2]. We see that the lines 1, 2, 3, 6, 7, 8, 10, 11 and 12 will be extended. We see that there will be probably four new lines. The problems concerning the metro map that will occur because of these plans will be described in Section 1.2.

1.2 The problem

For each city with a metro system, once every so many years the metro system is revised and a line is extended, a line disappears or a new line is added. When this occurs a new map has to be made, where the colors on the map should differ maximally from each other to make it easy to distinguish the lines from each other. However, the old lines should keep their color, because people are used to the colors they have. Therefore we need to find a color for the new line (or \( n \) new colors for \( n \) new lines), that is (or are) maximally different to all existing colors. How can we find such a color? And more specifically; what new color or new colors should Moscow use for their new line(s)?

1.3 Outline of the thesis

In this thesis, we will discuss different methods for obtaining a new color. In Chapter 2, we will discuss the physiology of the eye and we will give a summary of the developments in colorimetry over the years. In Chapter 3 we will describe different ways of finding a new color or multiple new colors. In Chapter 4, we will apply these methods to the Moscow metro map and we will discuss the results we obtained. All Matlab codes can be found in Appendix A. It is recommended to read a color-printed version of this thesis, because lots of colors are used.
Figure 1.3: Translation of the title: “Scheme for the future development of the Moscow metro 2012-2020.” The dashed lines depict the new parts of the route that are planned to be build before 2020.
Chapter 2

Measuring color

2.1 Seeing color

2.1.1 Construction of the eye

In this section, we will introduce some concepts of the working of the human eye and much of this section is adopted from Hunt [3]. A diagrammatic representation of a cross-section of the human eye is given in Figure 2.1. The cornea refracts light and accounts for approximately two thirds of the optical power of the eye. The lens also refracts light, but can alter the amount of refraction by changing its shape, being thinner for viewing distant objects and thicker for near objects. Together, the cornea and the lens project a small inverted image of the outside world on the retina, which is a light sensitive layer in the back of the eye. The iris can change its shape, having a central aperture that can differ from 2mm to 8mm in diameter. This aperture is called the pupil, and it is the area through which the light passes. By changing its diameter the iris compensates for

![Cross-sectional diagram of the human eye](image-url)
the changes in the illumination under which objects are seen. The retina covers most of the interior of the eye ball, however it is far from being uniform in sensitivity over its area. This means that our vision is very limited for things beyond about 40 degrees of the visual axis. In the retina there are different light receptors, namely cones and rods. The ratio of cones to rods varies continuously throughout the retina from all cones and no rods in the area near the visual axis to nearly all rods and very few cones beyond about 40 degrees from the visual axis. The individual cones and rods are connected to the brain by nerve fibres.

2.1.2 How do we see color?

The function of the rods in the retina is to give a monochromatic vision under low levels of illumination, such as moonlight and starlight. This form of vision is called scotopic. The function of the cones is to give color vision at normal levels of illumination, such as daylight and indoor artificial light, and is called a photopic form of vision. There is a gradual change from photopic to scotopic vision as the illumination level is lowered and in some intermediate area both cones and rods make significant contributions to our vision. This is called mesopic vision.

The rods and the cones are not equally sensitive to light of all wavelengths. In Figure 2.2 the spectral sensitivities of the rods and cones can be found. In scotopic vision no color can be seen, because for example a lightbeam of wavelength 500 nm will result in the same response as an approximately two times as strong beam of 550 nm. Scotopic vision, therefore only provides shades of white, gray and black as seen in moonlight.

In the case of the cones, knowledge about them has been obtained by combining several measurements on cones removed from eyes together with deductions from data on color
Maximizing color difference in metro maps

detective vision. As a result of several studies, the set of curves given by the filled lines in Figure 2.2 were obtained. These curves represent the spectral sensitivities for light on the cornea. For convenience they have been plotted so that their maxima are all equal to 1. It is clear that we now have a basis for color vision, because every wavelength corresponds to a unique ratio of responses of the different cone-types. Hence in photopic vision, the ratio of the signal will indicate the wavelength and the strength will indicate the intensity. Most colors consist of a mixture of many wavelengths, but the above argument is quite general and also applies to these more complex colors.

The three different types of cone $\rho$, $\gamma$ and $\beta$ are distributed more or less randomly. The relative quantities of the three types vary per person, however there are always much fewer $\beta$ cones that $\rho$ and $\gamma$ cones. An estimate is that their average ratio is 40:20:1 for $\rho$, $\gamma$ and $\beta$ respectively. This is quite understandable, because the eye focuses cannot simultaneously focus sharply the three regions in which the $\rho$, $\gamma$ and $\beta$ cones are most sensitive. The eye focuses light of wavelength about 560 nm, so both the $\rho$ and the $\gamma$ responses correspond to images that are reasonably sharp. The $\beta$ cones receive an image that is much less sharp, so it is unnecessary to have much $\beta$ cones to detect the color.

2.2 Color differences

To solve our problem, we are interested in a metric for color difference. For that we first need a coordinate system to describe colors. This is also called a color space.

Definition 2.1. A color space is a mathematical system for describing colors as tuples of numbers, typically as three or four values or color components.

Because we are discussing color differences, we also need some kind of metric. It would be convenient if the Euclidean distance in the used color space would be proportional to the difference people perceive in colors. A color space in which this is true is called a perceptually uniform color space.

Definition 2.2. A colorspace is called perceptually uniform, if the Euclidean distance with respect to the coordinates, is proportional to the perceptual difference in color.

2.2.1 RGB space

A well known and widely used color space is the RGB-space. In this color space, a color is represented by its components of red, green and blue, based on the wavelengths of the three cones in the human eye. A disadvantage of the RGB space is that it is not
Maximizing color difference in metro maps

uniquely defined. This means that there are different RGB spaces, like Adobe RGB, Apple RGB and more. Also the RGB space is not perceptually uniform.

2.2.2 CIELAB space

In 1931, the Commission Internationale de l’Eclairage (CIE), did measurements on the visions of hundreds of humans. From those measurements first the CIEXYZ color space was created. In 1976 the CIE recommended the use of two approximately perceptually uniform color spaces and color-difference formulae. The spaces and formulae were chosen to stimulate uniformity of practice during the development of a space and a formula giving a substantially better correlation with visual judgements. The official names were the CIE 1976 (L*u*v*) space and the CIE 1976 (L*a*b*) space, but they have become known by their officially recommended abbreviations CIELUV and CIELAB, respectively [4]. In the CIELAB space, a color is described by three coordinates; L* (lightness), a* (red-green scale) and b* (yellow-blue scale). A color can also be expressed in terms of its polar coordinates lightness L*, chroma

$$C_{ab}^* = \sqrt{a^*^2 + b^*^2}$$

and hue angle

$$h_{ab} = \arctan \frac{b^*}{a^*}.$$  

The CIELAB space is not bounded itself, however based on what you want to measure, you should look at only part of the space, for example, only the colors that your printer can print. This set of colors is then called the gamut of your printer.

Definition 2.3. The gamut of a device, process or color space is the set of colors that can be represented or reproduced.

2.2.3 Color difference formulae

Because the CIELAB space was supposed to be almost perceptually uniform, the CIE 1976 color difference formula (CIE76) is given by

$$\Delta E_{ab} = \sqrt{(\Delta L)^2 + (\Delta a)^2 + (\Delta b)^2}.$$ 

In fact the CIELAB space is not really uniform. In particular, at high values of chroma, the simple CIE76 formula values color differences too strongly compared to experimental
results of color perception. An improved color difference formula was therefore recom-
mended in 1994. This formula also considered the difference in background conditions
[5]. In 2001 CIE developed a color-difference formula that did not only take hue and
chroma into account, but also an interactive term between chroma and hue differences
for improving the performance for blue colors and a scaling factor for improving the
performance for gray colors [6]. This formula is called the CIEDE2000 formula or also
$\Delta E_{00}$ and can be found in Appendix A.
Chapter 3

Solution of the general problem

3.1 Adding one point

3.1.1 Mathematical formulation of the problem

Mathematically our problem can be formulated as follows: Given a set of points \( P = \{p_1, \cdots, p_k\} \) in \( S \subset \mathbb{R}^3 \), how can we find a point \( x \in S \) such that

\[
D(x) = \min_{p_i \in P} ||x - p_i||
\]

is maximal? Here the points \( p_i \) are the colors represented in CIELAB, \( S \) is our (bounded) gamut in CIELAB and \( ||x - p_i|| \) is the Euclidean distance between \( x \) and \( p_i \) given by their CIELAB coordinates.

3.1.2 The Voronoi diagram

For this purpose we can use the Voronoi diagram, of which we will first give an intuitive definition. Given a set of points \( P \) in \( \mathbb{R}^3 \), we associate all locations in \( \mathbb{R}^3 \) to the closest member(s) of the point set \( P \), with respect to Euclidean distance. This results in a tessellation of the plane into a set of the regions associated with members of the point set. This tessellation is called the Voronoi diagram generated by the point set. The formal definition is given by

**Definition 3.1.** Let \( P = \{p_1, \cdots, p_k\} \subset \mathbb{R}^n \), where \( 2 < k < \infty \) and \( p_i \neq p_j \) for \( i \neq j \), \( i, j \in I_k \). We call the region given by

\[
V(p_i) := \{x \in \mathbb{R}^n : ||x - p_i|| \leq ||x - x_j|| \text{ for all } j \neq i, j \in I_k \}
\]
the \textit{n-dimensional Voronoi polytope} associated with \( p_i \) and the set \( \mathcal{V}(p) = \{V(p_1), \cdots, V(p_k)\} \) the \textit{n-dimensional Voronoi diagram} generated by \( P \).

We see that the Voronoi polytope of a point \( p_i \) is nothing more than the points that are closer to \( p_i \) than to any other member of \( P \). To find a way of calculating such a polytope, we first see how this works in \( \mathbb{R}^2 \). We consider the line perpendicularly bisecting the line segment \( p_i p_j \) joining the two points \( p_i \) and \( p_j \). We call this line the bisector between \( p_i \) and \( p_j \). Since every point on the bisector is equi-distant from both \( p_i \) and \( p_j \), we can use this to create the Voronoi polygon of \( p_i \). Generalized to \( \mathbb{R}^n \), this bisector \( b(p_i, p_j) \) is given by

\[
b(p_i, p_j) = \{x \in \mathbb{R}^n : ||x - p_i|| = ||x - p_j|| \} \ j \neq i.
\]

This bisector divides the space in two regions: a region of points closer to \( p_i \) and a region of points closer to \( p_j \). The \textit{dominance region} of \( p_i \) over \( p_j \) is given by

\[
H(p_i, p_j) := \{x : ||x - p_i|| \leq ||x - p_j|| \} \ j \neq i.
\]

We can see that any point \( x \) in the Voronoi polytope associated with \( p_i \) should be in \( H(p_i, p_j) \) for all \( j \neq i \) and vice versa. This proves the following theorem.

\textbf{Theorem 3.2.} Let \( P = \{p_1, \cdots, p_k\} \subset \mathbb{R}^n \), where \( 2 < k < \infty \) and \( p_i \neq p_j \) for \( i \neq j \), \( i, j \in I_k \). Then the Voronoi polytope associated with \( p_i \) is equal to

\[
V(p_i) = \bigcap_{j \in I_n \setminus i} H(p_i, p_j).
\]

\textbf{3.1.3 Bounded Voronoi diagram}

In Definition 3.1.2 we defined a Voronoi diagram in an unbounded plane. However the gamut of our vision or the gamut of a printer is bounded inside the CIELAB space. In Figure 3.1, obtained by using [7] we see that we can approach this gamut by a polyhedron, which we will do to simplify our problem.

We will call this gamut \( S \), which is a bounded region in \( \mathbb{R}^3 \). We must have that \( P \subset S \). Because we want our new color to be in the gamut, we are interested in the bounded Voronoi diagram \( \mathcal{V}_{\cap S} \) given by

\[
\mathcal{V}_{\cap S}(P) = \{V(p_1) \cap S, \cdots, V(p_n) \cap S\}.
\]

Because the boundary is a polyhedron as well, so will the bounded Voronoi cells \( V(p_i) \cap S \) be, which we will call the bounded Voronoi polyhedra for future reference. We can
construct the bounded Voronoi diagram of our set $P$ in our color space $S$ using only linear algebra. We can use this for finding our new color, because only vertices of the bounded Voronoi polyhedra are left as possible optimal points, which is a finite number of points.

**Theorem 3.3.** The $x \in S$ that maximizes the function $D(x) = \min_{p_i \in P} ||x - p_i||$ must lie on a vertex of a bounded Voronoi polyhedron associated with a $p_i \in P$.

**Proof.** Suppose that $D(x)$ is maximal for $x = x_1 \in S$. Suppose that this $x_1$ is not in a vertex of a bounded Voronoi polyhedron. Because a Voronoi diagram is a tessellation of space, $x_1$ must belong to a bounded Voronoi polyhedron $V(p_i)$. Therefore $x_1$ is closer to $p_i$ than to any other $p_j \in P$, so $D(x_1) = ||x_1 - p_i||$. However, because $x_1$ is not in a vertex of $V(p_i)$, there must be an $\epsilon > 0$ and a nonzero vector $v$ (not every vector will work, because $x_1$ might be on the boundary), such that the segments that connects the points $x_2 = x_1 + \epsilon v$ and $x_3 = x_1 - \epsilon v$ is contained in $V(p_i)$. Than the line segment $x_1p_i$ is the median of the triangle $p_ix_2x_3$, so one of the two line segments $x_2p_i$ or $x_3p_i$ must be longer than $x_1p_i$, so $D(x)$ is not maximal for $x = x_1$. This is a contradiction so $x_1$ must be on a vertex of a bounded Voronoi polyhedron. \qed

### 3.1.4 CIE76 Voronoi method

We will call the method of calculating all vertices of the Voronoi diagram with respect to the Euclidean metric and choosing the one with maximal minimal distance the **CIE76 Voronoi method**, because it uses the CIE76 distance. But how can we calculate these
vertices? As said before, the bounded Voronoi polyhedron of \( p_i \) is given by \( V(p_i) \cap S = (\bigcap_{j \in I \setminus i} H(p_i, p_j)) \cap S \). Therefore a vertex is either a vertex of \( S \), an intersection of at least three bisectors, an intersection of two bisectors and the boundary of \( S \), or an intersection of two boundary planes and a bisector. Such a point \( x \) is only a vertex of \( V(p_i) \cap S \) if the line segment \( p_i x \) does not cross any other bisector of two points in \( P \) or the boundary of \( S \). Because we have a finite number of points, we also have a finite number of bisectors and thus a finite number of vertices. After finding all vertices \( x_{ij} \) of a bounded Voronoi polyhedron, we can calculate their distances to \( p_i \) and because they are vertices of \( V(p_i) \cap S \), we know that these distances are equal to \( D(x_{ij}) \). After doing this for all \( k \) Voronoi polyhedra, we have all vertices of bounded Voronoi polyhedrons and their corresponding minimum distances and we can see which one maximizes \( D(x) \).

### 3.1.5 CIE76-CIEDE2000 Voronoi method

Until now we used the CIE76 color difference formula, which is the Euclidean distance in CIELAB. The later developed CIEDE2000 formula \( \Delta E_{00} \) as given in Appendix A is more accurate in describing color differences. \( \Delta E_{00} \) is not continuous \(^8\). Therefore it is very hard to maximize this color difference. We can however improve the algorithm described above by using the CIEDE2000 difference formula to pick an optimal point after selecting candidate points using the Voronoi diagram obtained by using the Euclidean distances. We will call this method the \( \text{CIE76-CIEDE2000 Voronoi method} \).

### 3.1.6 Voronoi diagrams for non-Euclidean metrics

Using the CIEDE2000 formula (\( \Delta E_{00} \)) to pick an optimum improves our method, however that does not really maximize the minimal \( \Delta E_{00} \), because we use the Euclidean metric to select the candidate points. Could we somehow create a Voronoi diagram with a non-Euclidean metric, for example \( \Delta E_{00} \)? Such a non-Euclidean diagram is called a generalized Voronoi diagram with respect to a distance metric \( d \). A bisector of \( p_i \) and \( p_j \) is then defined as

\[
b(p_i, p_j) = \{ x : d(x, p_i) = d(x, p_j) \} j \neq i,
\]

and the dominance region of \( p_i \) over \( p_j \) as

\[
\text{Dom}(p_i, p_j) := \{ x : D(x, p_i) \leq d(x, p_j) \} j \neq i.
\]
The Voronoi region of $p_i$ with respect to the distance $d$ is now defined by

$$ V(p_i) = \bigcap_{j \neq i} \text{Dom}(p_i, p_j). $$

The collection of $V(p_1), \cdots, V(p_k)$ is called the Voronoi diagram for $P$ with respect to $\Delta E_{00}$. To obtain an optimal color we could look at the Voronoi diagram of $P$ with respect to $\Delta E_{00}$. However, in this case it is not clear whether an optimum should be in a vertex, because the argument of Theorem 3.3 does not hold here. Also, a bisector does not have to be a 2 dimensional manifold anymore. For example; if we look at the case of the discrete metric the bisector of two points is the whole space minus the two points. Because of this, it might not be possible to calculate the optimal point with respect to $\Delta E_{00}$ using Voronoi diagrams. Also, because $\Delta E_{00}$ is not even continuous, other ways of finding an optimum are also difficult, or require a lot of computational power. However, what we could do is use Coulomb repulsion to maximize not the minimal distance, but the weighted distance to all other points.

### 3.1.7 Brute force method

We could try to minimize $\Delta E_{00}$ by calculating this distance for a lot of randomly distributed points and picking the one with maximal minimal distance. The more random points we will use, the larger the chance of being very close to the optimal point.

#### 3.1.7.1 Probability of hitting an optimum

Using the brute force method, what is the probability of hitting an optimum with an error in the minimal $\Delta E_{00}$ smaller than $\epsilon$? We will shoot random points in a box with dimensions $[0, 100] \times [-100, 100] \times [-100, 100]$. The volume $V$ of this box is equal to $100 \cdot 200 \cdot 200 = 4,000,000$. What is the probability that such a point is in the gamut and close to the global optimum $p_{max}$ with minimum distance $D_{max}$?

To estimate this, we need an approximation of the volume of the area with a minimum distance $D \geq D_{max} - \epsilon$. We will call this volume $V_{\epsilon}$. Probability theory than tells us that the chance that we hit this area within $N$ shots, is equal to

$$ P = 1 - \left(\frac{V - V_{\epsilon}}{V}\right)^N. $$

Now how can we approximate $V_{\epsilon}$? Because $\Delta E_{00}$ is derived from $\Delta E_{76}$, we could use $\Delta E_{76}$ to do this. As we will see in Chapter 4, the minimum CIE 76 distance of an optimal point is roughly two to five times as high as the minimum CIEDE2000 distance.
of an optimal point. Therefore, if we want to have

$$\min \Delta E_{00} \geq D_{\text{max}00} - \epsilon,$$

we will approximately have that

$$\Delta E_{76} \leq D_{\text{max}76} - 3.5\epsilon,$$

where $D_{\text{max}00}$ and $D_{\text{max}76}$ denote the optimal CIEDE2000 distance and the optimal CIE 76 distance respectively. If we choose $\epsilon$ small enough, we can be sure that there is no local optimum with a minimum CIEDE2000 distance greater than $D_{\text{max}00} - \epsilon$. Therefore, $V_\epsilon$ can be approximated by the volume of a sphere with radius $3.5\epsilon$ around the optimum. This volume is equal to

$$V_\epsilon = \frac{4}{3}\pi(3.5\epsilon)^3.$$

If we shoot 100,000,000 random points, the probability of hitting a point closer than $\Delta E_{00} = 0.2$ to the optimum is than equal to

$$P = 1 - \left(\frac{4,000,000 - \frac{4}{3}\pi \cdot 0.7^3}{4,000,000}\right)^{100,000,000} = 0.97.$$

### 3.1.8 Random walk method

For the brute force method, we have to try a lot of random points. We could make the method faster by starting in a random point and try some points close around our this point. If one of the points is better than the starting point, we could try points close around this better point and take the best one of those. This method is called the random walk method, because you let your random point "walk" to a nearby optimum. We could even make this method faster by starting not in a random point, but in a corner of the Voronoi diagram, because these corners are already approximations of a local optimum.

### 3.1.9 Coulomb repulsion method

#### 3.1.9.1 Coulomb repulsion

Coulomb repulsion is the repulsive force between two positively or negatively charged particles (see Figure 3.2) as described in Coulombs law 3.1, which states that the repulsing force acting on the particles is proportional to the multiplication of the magnitudes of
Maximizing color difference in metro maps

Charges and inversely proportional to the square of the distance between them. Coulomb's law is given by

$$|F| = k_e \frac{q_1 q_2}{r^2},$$

(3.1)

Where $F$ is the repulsing force, $k_e$ is Coulomb's constant and $r$ is the distance between the two particles. Using Coulomb's law, one can for example model the behaviour of particles in gasses or liquids.

![Figure 3.2: Two charged particles and the forces acting on them.](image)

How could we use Coulomb repulsion to find a new color on a metro map? We could see all colors in the CIELAB space as charged particles that repulse each other according to Coulomb's law. We could use any distance measure we want, for example $\Delta E_{00}$. As stated before, such a system can be modelled and could also be used to find a new color for the Moscow metro map. This color would not have a maximal minimal distance, but it would also take into account the points that are farther away than the closest point. In our case, we could see the old lines as fixed points with a (unitary) charge and the new line as a moving particle with a (unitary) charge. We should give the boundary a very small charge to make sure that the particles do not cross it. We should also introduce a small friction force to make sure the new particle will stop at some time.

3.1.9.2 The model

We will take the following forces into account:

1. Repulsion between the old lines and the new line.
2. The boundary; a fictitious charge that repulses the new lines.
3. Viscosity (friction).

We will describe the position of the new color as a three-dimensional vector $x$ and the positions of the existing points $p_i \in P$ as three-dimensional vectors $p_i$. We know from Newton's second law that

$$F = m \cdot \ddot{x},$$
where $F$ is the force acting on the moving particle, $m$ is its mass and $\ddot{x}$ its acceleration.
We will take $m = 1$. We also know that the force acting on the moving particle is given by a sum of all Coulomb repulsions and the friction, that is

$$F = m \cdot \ddot{x} = \dot{x} = \gamma \cdot \dot{x} + \sum_{p_i \in P} \frac{k}{||x - p_i||^3} \cdot (x - p_i) + \frac{c}{||x - b||^3} (x - b),$$

where $\gamma$ denotes the magnitude of the friction force, $k$ is the charge of all colorpoints, $c$ is the charge at the closest boundary point boundary and $b$ is the closest boundary point.
If we would start somewhere within the color domain for $t = 0$, but not at any of the existing points, we could the route of our new point, by choosing $\dot{x} = 0$ and $\ddot{x} = 0$ for $t = 0$, and set

$$x_{t+1} = x_t + \dot{x}_t \cdot \Delta t,$$
$$\dot{x}_{t+1} = \ddot{x}_t \cdot \Delta t$$

and

$$\ddot{x}_{t+1} = \gamma \cdot \dot{x}_{t+1} + \sum_{p_i \in P} \frac{k}{||x_{t+1} - p_i||^3} \cdot (x_{t+1} - p_i) + \frac{c}{||x_{t+1} - b||^3} (x_{t+1} - b),$$

where $\gamma$ denotes the magnitude of the friction force, $k$ is the charge of all colorpoints, $c$ is the charge at the closest boundary point boundary and $b$ is the closest boundary point.
We should adapt some constants to make sure that we do not jump over the boundary. If we would let this model run for a long time, it is not certain that we would obtain an optimum. It is also possible that our new point is “trapped” between existing particles. However, if we would run our model many times with different starting points, we have a good chance of finding an optimum.

### 3.2 Adding more than one point

We showed how we can calculate which color we should use if we want to add one more line to the metro map. Suppose we want to add two new lines to the metro map.
The problem we want to solve can now be formulated as follows: Given a set of points $P = \{p_1, \cdots, p_k\} \in S \subset \mathbb{R}^3$ where $S$ is our (bounded) colorspace, how can we find two points $x, y \in S$ such that

$$M(x, y) = \min \left\{ \min_{p_i \in P} ||x - p_i||, \min_{p_i \in P} ||y - p_i||, ||x - y|| \right\}$$

(3.2)
is maximal?

### 3.2.1 One by one Voronoi method

What we could do is first find one new color with the CIE76 Voronoi method or the CIE76-CIEDE2000 Voronoi method, add that one, and after that find another new color the same way. We will call this the **one by one CIE76 Voronoi method** and the **one by one CIE76-CIEDE2000 Voronoi method** respectively. However, it is possible that this method does not give the optimal solution. This can easily be seen with a one dimensional example.

**Example 3.1.** Suppose you have a one dimensional color system with Euclidean distance, bounded by $[0, 1]$. Suppose you already have the two colors $x = 0$ and $x = 1$ and you want to find two new colors. If we would first find one new optimal color, this would be at $x = 0.5$. The second new color would be at either $x = 0.25$ or $x = 0.75$, which would give us a minimum distance of 0.25 between points. We immediately see that we could better have chosen our new colors at $x = \frac{1}{3}$ and $x = \frac{2}{3}$, which would have given us a minimum distance of $\frac{1}{3}$. This shows that applying the Voronoi method for one new point twice does not always give us the optimal result if we know in advance that we want to add two colors.

### 3.2.2 Nonsmooth optimization

The one by one method clearly does not always give us the optimal colors. Is it possible to maximize $M(x, y)$ by other techniques? We will first show that $M(x, y)$ is continuous.

**Proposition 3.4.** $M(x, y)$, as defined in 3.2 is a continuous function.

**Proof.** For a fixed point $p_i$ the function $f(x, y) = ||x - p_i||$, with $x, y \in \mathbb{R}^3$ is a continuous function on $\mathbb{R}^6$, because for every $\epsilon$ we can choose $\delta = \epsilon$ and then for $||(x_1, y_1) - (x_2, y_2)|| < \delta$ the reverse triangle inequality gives us

$$|| ||x_1 - p_i|| - ||x_2 - p_i|| || \leq ||x_1 - p_i - (x_2 - p_i)||$$

$$= ||x_1 - x_2||$$

$$\leq ||(x_1, y_1) - (x_2, y_2)||$$

$$< \delta$$

$$= \epsilon.$$
Also \( f(x, y) = \|x - y\| \) is a continuous function on \( \mathbb{R}^6 \), because for every \( \epsilon \) we can choose \( \delta = \frac{\epsilon}{2} \) and then for \( \|(x_1, y_1) - (x_2, y_2)\| < \delta \) the reverse triangle inequality and the triangle inequality give us

\[
\begin{align*}
\| \|x_1 - y_1\| - \|x_2 - y_2\| \| &\leq \|x_1 - y_1 - (x_2 - y_2)\| \\
&\leq \|x_1 - x_2\| + \|y_1 - y_2\| \\
&\leq 2\|(x_1, y_1) - (x_2, y_2)\| \\
&< 2\delta \\
&= \epsilon.
\end{align*}
\]

Now we will proof that the minimum of two continuous functions is continuous. We have \( h(x) = \min\{f(x), g(x)\} \), with \( f(x) \) and \( g(x) \) continuous and \( x \in \mathbb{R}^6 \). Because \( f \) and \( g \) are continuous, for every \( \epsilon \) there is a \( \delta_f \) and a \( \delta_g \) such that if \( \|x_1 - x_2\| < \delta_f \), then \( \|f(x_1) - f(x_2)\| < \epsilon \). The same holds for \( g \). It is trivial that \( h \) is continuous on points \( x \) where \( f(x) \neq g(x) \). We will show it is also continuous on a point \( x_0 \), where \( f(x_0) = g(x_0) \). For any \( \epsilon \), choose \( \delta \) as the minimum of \( \delta_f \) and \( \delta_g \). If \( \|x_0 - x\| < \delta \), which is smaller than or equal to \( \delta_f \) and \( \delta_g \), we have either

\[
\|h(x_0) - h(x)\| = \|f(x_0) - f(x)\| < \epsilon,
\]

or

\[
\|h(x_0) - h(x)\| = \|g(x_0) - g(x)\| < \epsilon.
\]

Therefore the minimum of two continuous functions is also continuous. Because \( M(x, y) \) is the minimum of the minimum (etc.) of continuous functions, \( M(x, y) \) is continuous itself. \( \square \)

We now want to maximize a continuous function that is not necessarily smooth. In [9], Rockafellar shows that we can rewrite our problem to a smooth one. We want to maximize \( 3.2. \) with \( (x, y) \in S \times S \subset \mathbb{R}^6 \). The maximization of \( M \) over \( S \times S \) can be approached by reformulating the given problem in a higher dimensional space. From \( (x, y) \in \mathbb{R}^6 \) and an additional variable \( x_0 \in \mathbb{R} \), we can put together vectors \( \tilde{x} = (x_0, x, y) \in \mathbb{R}^7 \) and look to maximizing \( M_0(\tilde{x}) = x_0 \) over all \( \tilde{x} \in \mathbb{R} \times S \times S \), that satisfy the constraints

\[
\|x - p_i\| - x_0 \geq 0 \text{ for each } p_i \in P, \]

\[
\|y - p_j\| - x_0 \geq 0 \text{ for each } p_j \in P, \]

\[
\|x - y\| - x_0 \geq 0.
\]
Here the constraint functions are smooth, so the problem is now in a standard setting. We can of course use additional constraints on \((x,y)\), to express our set \(S \times S\). This way we are dealing with a nonlinear constrained optimization problem, for which we could use standard optimization techniques, for example by the Nelder-mead (simplex) method.

### 3.2.3 Brute force method

For two new points we can also apply the brute force method. We could add the points one by one, but we could also add the two points simultaneously. If we do this, we are shooting two random points instead of one. Therefore the chance of hitting their combined optimum decreases. For 100.000.000.000 shots, the chance of having a mistake smaller than 0.2 will be approximately

\[
P = 1 - \left(1 - 2^{43} \cdot 0.73^{43} \cdot 0.73^3 \right)^{10.000.000.000.000} = 0.92.
\]

### 3.2.4 Coulomb repulsion

For adding more than one point, we could also use Coulomb repulsion, as described in Section 3.1.9. We should however add a term that takes into account the Coulomb repulsion between all new points. This would again give colors for which the sum of the weighted distances is maximised, instead of their minimum distance. In this case, for each new point \(x_i, x_{i(t+1)}\) and \(\dot{x}_{i(t+1)}\) are the same as in Section 3.1.9, but \(\ddot{x}_{i(t+1)}\) has a term that takes in account the forces between the new points. It is given by

\[
\ddot{x}_{i(t+1)} = -\gamma \cdot \dot{x}_{i(t+1)} + \sum_{p_i \in P} \frac{1}{||x_{i(t+1)} - p_i||^2} \cdot (x_{i(t+1)} - p_i)
+ \sum_{i \neq j} \frac{1}{||x_{i(t+1)} - x_{j(t+1)}||^2} \cdot (x_{i(t+1)} - x_{j(t+1)}) + \frac{c}{||x_{i(t+1)} - b||^2} (x_{i(t+1)} - b),
\]

where \(\gamma\) denotes the magnitude of the friction force, \(c\) is the charge of the boundary and \(b\) is the closest boundary point.
Chapter 4

Application to the Moscow metro map

4.1 The colors of the Moscow metro

The Moscow metro currently has 11 lines. There are three more colors used on the metro map, namely white for the background, black for the text and lightblue for the water. There is also a twelfth line, however, this looks different on the map and is not a real metro line, so it is not taken into account. In Table 4.1, the coordinates of the different colors in the CIELAB space can be found. The points are plotted in Figure 4.1. Figure 4.2 is a stereographical image of this plot, which enables you to see it in three dimensions. Where points fall on or slightly outside the boundary, we have shifted them a bit to prevent problems with the calculations.

4.2 Finding optimal colors by the CIE76 Voronoi method

To find the optimal new color for the Moscow metro map with the CIE76 Voronoi method, we calculate all vertices of the bounded Voronoi polyhedrons. As stated before, the vertices of a bounded Voronoi polyhedron \( V(p_i) \) can be found by finding intersections of at least three bisector- or boundary planes, which can be done by solving a matrix equation \( Ax = b \). First we define the the gamut by the intersection of \( n \) halfspaces and check whether all colors present on the Moscow metro map are inside the gamut. We calculate the boundary planes and the bisector planes, by calculating a point on the plane and a normal vector to the plane. After finding all the intersections of three planes by solving a matrix equation \( Ax = b \), we need to check whether those intersections
Figure 4.1: The colors of the Moscow metro map in the CIEL*a*b* space. The exact coordinates can be found in Table 4.1.

Figure 4.2: A stereographical image of the colors of the Moscow metro map in the CIEL*a*b* space. View the left picture with your left eye and the right picture with your right eye at the same time to see a 3-dimensional picture. Scales and lables on the axes can be found in Figure 4.1. The colors of the colorpoints are only estimations of the real colors.
<table>
<thead>
<tr>
<th>Line</th>
<th>Color</th>
<th>L</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red</td>
<td>52</td>
<td>74</td>
<td>53</td>
</tr>
<tr>
<td>2</td>
<td>Ocean green</td>
<td>56</td>
<td>-45</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>Cobalt</td>
<td>35</td>
<td>7</td>
<td>-43</td>
</tr>
<tr>
<td>4</td>
<td>Sky blue</td>
<td>61</td>
<td>-16</td>
<td>-42</td>
</tr>
<tr>
<td>5</td>
<td>Olive brown</td>
<td>41</td>
<td>6</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>Peach</td>
<td>76</td>
<td>24</td>
<td>67</td>
</tr>
<tr>
<td>7</td>
<td>Pinkish purple</td>
<td>43</td>
<td>64</td>
<td>-24</td>
</tr>
<tr>
<td>8</td>
<td>Light mustard</td>
<td>86</td>
<td>4</td>
<td>85</td>
</tr>
<tr>
<td>9</td>
<td>Light grey</td>
<td>71</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>10</td>
<td>Greenish yellow</td>
<td>80</td>
<td>-26</td>
<td>68</td>
</tr>
<tr>
<td>11</td>
<td>Turquoise blue</td>
<td>56</td>
<td>-21</td>
<td>-29</td>
</tr>
<tr>
<td>Water</td>
<td>Pale blue</td>
<td>93</td>
<td>-8</td>
<td>-9</td>
</tr>
<tr>
<td>Background</td>
<td>White</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Text</td>
<td>Black</td>
<td>13</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1: Coordinates of colors on the Moscow metro map. The names of the colors come from [10]. The colors shown here are only an estimation of the real color.

are actually vertices of the bounded Voronoi polyhedron. Therefore, for each such an intersection \( x \), we check whether the line segment joining \( p_i \) and \( x \) intersects any of the boundary- or bisector planes.

**Proposition 4.1.** The line segment joining two points \( \vec{x} \) and \( \vec{y} \) in \( \mathbb{R}^3 \) intersects the plane \( f(x, y, z) = ax + by + cz + d = 0 \) if and only if \( f(\vec{x}) \) and \( f(\vec{y}) \) have a different sign.

**Proof.** The proof is trivial. \( \square \)

**Example 4.1.** To illustrate how we calculate vertices of the bounded Voronoi polyhedrons, we will give an 2-dimensional example. Suppose we are only dealing with three points, namely \( A, B \) and \( C \), in a 2-dimensional space bounded by a square. This is shown in Figure 4.3. If we want to calculate the vertices of the bounded Voronoi polygon of point A, we will first calculate the bisectors of \( AB \) and \( AC \). We now have a total of six edges, namely the two bisectors and four boundary edges. We can calculate their intersections, which gives us in this case nine intersections: \( \alpha, \cdots \) and \( \iota \). For each of these intersections we check whether it is a vertex of the bounded Voronoi polygon of A, by checking whether the line segment joining the intersection and A crosses an edge (boundary of bisector). In Figure ref:calculatecell, lines that do cross an edge are colored red and lines that do not cross an edge are colored green. We see that all green lines belong to a vertex of the bounded Voronoi polygon of A. This way we obtained all the vertices of the bounded Voronoi polygon of A, which are \( \alpha, \delta, \epsilon \) and \( \eta \).

The Matlab code of a program that calculates all Voronoi vertices of our 15 bounded voronoi cells and their corresponding minimal distances can be found in Appendix B.
The minimal distance of a vertex of $V(p_i)$ is equal to its distance to $(p_i)$. This gives us a reasonably small number of candidate points, which is bounded by $15 \cdot 26 \cdot 25 \cdot 24 = 23400000$, because we have 15 points, which all have 26 corresponding bisectors and boundary planes. From these candidate points, we can choose the one with the maximal minimal distance. To reduce the number of candidate points we can adapt the program such that it only shows points with a minimal distance greater than some chosen threshold. The color we obtain differs for a different choice of the gamut. In Figure 3.1 the points we used to approach the boundary of the gamut are marked $A, \cdots, H$ and their corresponding coordinates in CIELAB rounded to integers can be found in Table 4.2. Our boundary will be given by 12 planes each passing through three points, namely: ABG, AFG, ABD, BDC, ADF, DEF, HBC, HCD, HDE, HEF, HFG and HGB. We now used the gamut of Adobe, one could of course also make the same calculations using another gamut; for example the gamut of the printer used to print the metro maps.

With this choice of the gamut, we find many candidate colors for a new metro line. The best 15 candidates can be found in Table 4.3. The optimal color is given by $(83, -138, 91)$ in CIELAB coordinates, which is bright green ($\bullet$). This new color is a vertex of the
Table 4.2: Coordinates of the cornerpoints of an approached RGB gamut in CIELAB.

<table>
<thead>
<tr>
<th>Point</th>
<th>Color</th>
<th>L</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>White</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>Blue</td>
<td>33</td>
<td>80</td>
<td>-109</td>
</tr>
<tr>
<td>C</td>
<td>Fuchsia</td>
<td>67</td>
<td>105</td>
<td>-52</td>
</tr>
<tr>
<td>D</td>
<td>Red</td>
<td>61</td>
<td>90</td>
<td>75</td>
</tr>
<tr>
<td>E</td>
<td>Yellow</td>
<td>97</td>
<td>-23</td>
<td>105</td>
</tr>
<tr>
<td>F</td>
<td>Bright green</td>
<td>83</td>
<td>-138</td>
<td>91</td>
</tr>
<tr>
<td>G</td>
<td>Cyan</td>
<td>87</td>
<td>-78</td>
<td>-21</td>
</tr>
<tr>
<td>H</td>
<td>Black</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.3: Best 15 candidate points for the optimal color in the Moscow metro. All points are vertices of a bounded Voronoi cell. The best candidate is at the top of the table.

<table>
<thead>
<tr>
<th>Color</th>
<th>D(x)</th>
<th>L</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bright green</td>
<td>114.4</td>
<td>83</td>
<td>-138</td>
<td>91</td>
</tr>
<tr>
<td>Bright green</td>
<td>108.5</td>
<td>83</td>
<td>-133</td>
<td>83</td>
</tr>
<tr>
<td>Bright green</td>
<td>102.0</td>
<td>76</td>
<td>-126</td>
<td>84</td>
</tr>
<tr>
<td>Blue</td>
<td>87.1</td>
<td>33</td>
<td>80</td>
<td>-109</td>
</tr>
<tr>
<td>Blue</td>
<td>72.0</td>
<td>41</td>
<td>56</td>
<td>-96</td>
</tr>
<tr>
<td>Lime green</td>
<td>67.5</td>
<td>89</td>
<td>-93</td>
<td>61</td>
</tr>
<tr>
<td>Cyan</td>
<td>65.3</td>
<td>87</td>
<td>-78</td>
<td>-21</td>
</tr>
<tr>
<td>Cyan</td>
<td>64.8</td>
<td>87</td>
<td>-77</td>
<td>-21</td>
</tr>
<tr>
<td>Cyan</td>
<td>56.0</td>
<td>90</td>
<td>-64</td>
<td>-15</td>
</tr>
<tr>
<td>Bright pink</td>
<td>55.1</td>
<td>67</td>
<td>105</td>
<td>-52</td>
</tr>
<tr>
<td>Cyan</td>
<td>54.2</td>
<td>90</td>
<td>-62</td>
<td>-17</td>
</tr>
<tr>
<td>Rose pink</td>
<td>54.2</td>
<td>80</td>
<td>51</td>
<td>13</td>
</tr>
<tr>
<td>Rose pink</td>
<td>52.3</td>
<td>82</td>
<td>46</td>
<td>20</td>
</tr>
<tr>
<td>Bright pink</td>
<td>51.6</td>
<td>64</td>
<td>98</td>
<td>9</td>
</tr>
<tr>
<td>Indigo</td>
<td>51.0</td>
<td>20</td>
<td>49</td>
<td>-67</td>
</tr>
</tbody>
</table>

boundary. The point has smallest Euclidean distance to line number 10 in the CIELAB space, namely a distance of approximately 114.

### 4.2.1 More new colors: One by one method

In table 4.4, we can see the first six new colors obtained by the one by one CIE76 Voronoi method.
### Maximizing color difference in metro maps

#### 4.3 Finding new colors by the CIE76-CIEDE2000 Voronoi method

We select the vertices the same way, however now we check for each vertex the minimal \( \Delta E_{00} \) instead of the Euclidean distance as described in Section 3.1.5. The Matlab script for calculating \( \Delta E_{00} \) can be found in [11]. Because the difference in lightness is less important than the difference in hue and chroma in our case, we will use the weighing coefficients \( k_l : k_c : k_h = 2 : 1 : 1 \). In Table 4.5 the first 5 new colors we find by the one by one CIE76-CIEDE2000 Voronoi method are given. We see that the fifth color is has even a greater minimal distance than the fourth added color. The reason for this is that we only check the minimal distance of our candidate points: the vertices of the Voronoi diagram with respect to the Euclidean distance. After we added the fourth point; we obtain new vertices, of which one is even farther away from all points than the fourth point was. This shows that the optimal point with respect to \( \Delta E_{00} \) does not have to be in one of the vertices of the Voronoi diagram with respect to the Euclidean distance.

<table>
<thead>
<tr>
<th>Color</th>
<th>( D(x) )</th>
<th>L</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bright green</td>
<td>114,4</td>
<td>83</td>
<td>-138</td>
<td>91</td>
</tr>
<tr>
<td>Blue</td>
<td>87,0</td>
<td>33</td>
<td>80</td>
<td>-109</td>
</tr>
<tr>
<td>Sea green</td>
<td>67,7</td>
<td>85</td>
<td>-106</td>
<td>31</td>
</tr>
<tr>
<td>Cyan</td>
<td>65,3</td>
<td>87</td>
<td>-78</td>
<td>-21</td>
</tr>
<tr>
<td>Bright pink</td>
<td>56,1</td>
<td>60</td>
<td>100</td>
<td>-64</td>
</tr>
<tr>
<td>Rose pink</td>
<td>54,2</td>
<td>80</td>
<td>51</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 4.4: Colors chosen for the first 6 new lines using the one by one CIE76 Voronoi method.

<table>
<thead>
<tr>
<th>Color</th>
<th>( \min \Delta E_{00} )</th>
<th>L</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyan</td>
<td>23.9</td>
<td>87</td>
<td>-78</td>
<td>-21</td>
</tr>
<tr>
<td>Bright green</td>
<td>22.0</td>
<td>83</td>
<td>-138</td>
<td>91</td>
</tr>
<tr>
<td>Blue</td>
<td>19.8</td>
<td>33</td>
<td>80</td>
<td>-109</td>
</tr>
<tr>
<td>Pale lavender</td>
<td>19.7</td>
<td>86</td>
<td>23</td>
<td>-22</td>
</tr>
<tr>
<td>Rose pink</td>
<td>19.8</td>
<td>78</td>
<td>59</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 4.5: Colors chosen for the first 5 new lines using the one by one CIE76-CIEDE2000 Voronoi method using \( k_l : k_c : k_h = 2 : 1 : 1 \).

#### 4.4 Finding new colors by the brute force method

##### 4.4.1 One by one

For finding new colors with the brute force method, we are looking at random points with \( L \in [0, 100], a \in [-100, 100] \) and \( b \in [-100, 100] \). We first see whether the point is
inside the color gamut. For points inside the gamut we check their minimal distance to P. We only save the largest one. We do this for \( N = 10^5 \) points, which gives us a chance of approximately 0.04 of being closer than 0.2 to the optimum. This is not a very good chance, so when wanting to use this method for a new color, one should let the program run for a longer time. However, doing this for \( N = 10^5 \) points takes 3 minutes, so trying \( N = 10^8 \) points, would take approximately 50 hours. Using the one by one method, this gives us the new colors that can be found in table 4.6. Surprising here is that the color bright green is not in the table. But this is of course possible due to the small chance of hitting the optimum.

<table>
<thead>
<tr>
<th>Color</th>
<th>( \min \Delta E_{00} )</th>
<th>L</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyan</td>
<td>25.1</td>
<td>86</td>
<td>-80</td>
<td>-15</td>
</tr>
<tr>
<td>Baby pink</td>
<td>22.9</td>
<td>86</td>
<td>33</td>
<td>3</td>
</tr>
<tr>
<td>Dark teal</td>
<td>21.7</td>
<td>23</td>
<td>-17</td>
<td>-1</td>
</tr>
<tr>
<td>Blue violet</td>
<td>21.3</td>
<td>38</td>
<td>77</td>
<td>-95</td>
</tr>
<tr>
<td>Maroon</td>
<td>20.5</td>
<td>22</td>
<td>33</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 4.6: Colors chosen for the first 5 new lines using the one by one brute force method for \( n = 100.000 \).

### 4.4.2 Adding two points simultaneously

For this we will use \( N = 10^6 \), which gives us a chance of approximately \( 2.6 \cdot 10^{-7} \) of being closer than 0.2 to the optimum. This is of course a very small chance, but again, when we know in advance we want to have two new colors, we could try it for \( N = 10^{13} \), on a powerful computer to get a better solution. For \( N = 10^6 \) we obtained the two colors Cyan \( \bullet (84, -75, -15) \) and baby pink \( \bullet (81, 28, 6) \), with minimal distance 21.9. We see that this is not a better solution than the first two colors found by the one by one brute force method. This could be solved if we would use \( N = 10^{13} \), for which we need a lot of time or a powerful computer.

### 4.5 Finding two new colors by the build in Matlab function fmincon

For this we will again use the Euclidean distance, because \( \Delta E_{00} \) is not a smooth function. The build in function fmincon looks for local minima subject to some linear and some nonlinear constraint functions. The boundary of the gamut will give the linear constraints and the nonlinear constraint functions are given in Section 3.2.2. Because fmincon gives us only a local optimum, we try it for 1,000,000 random initial points.
(x₀, x, y) ∈ ℝ⁷, with x₀ = 0, to make sure that it satisfies the constraints. After selecting global optima this way, we maximize the minimal CIEDE2000 distance. This way we obtain the two colors Aqua blue • (85, -70, -11) and Rose Pink • (83,29,5). Their combined minimal CIEDE2000 distance is 22.1. This is more than the one by one CIE76-CIEDE2000 Voronoi method gives us.

4.6 Finding new colors by the one by one Coulomb repulsion method

We made a model for the Coulomb repulsion method for the CIEDE2000 distance. Here we chose the constants

\[ k = 10, \]
\[ \gamma = 0.1, \]
\[ c = 0.2 \cdot k, \]
\[ dt = 0.1 \]

and we use the color difference measure \( \Delta E_{00} \). We start with a random point and check whether it is inside the color gamut. If it is, we run the Coulomb repulsion model for 2000 steps. We check whether the endpoint is still inside the boundary and we calculate the minimum distance to \( P \). When doing this for 300-600 starting points, the results obtained vary a lot and will therefore not be discussed here. The matlab code for this method can be found in Appendix B.

4.7 Conclusions

Because the CIEDE2000 formula best describes the perceptual differences between colors, we will use the results that (at least partly) use the CIEDE2000 formula. We see that given the CIEDE2000(2:1:1) formula, the brute force one by one method gives us the greatest minimal distances. I would therefore suggest to use most of the colors from Table 4.6 for the new lines on the Moscow metro map, or run this method longer on a faster computer, which could give even better results. The results depend of course on the choices of \( k_L, k_C \) and \( k_H \), which should be chosen carefully. One should look at the metro map and choose the colors in a way that colors that are most similar to each other do not overlap. Because Cyan is most different from all old colors, this could be
Maximizing color difference in metro maps

used for the new circle-line (see Figure 1.3). Note that these results also depend on the
boundary we used for the CIELAB space; which is the boundary of the adobe 98 gamut.
Of course this boundary should depend on the gamut of the printer used for printing
the metro map. Some of the new colors that are suggested by different methods are on
the boundary or very close to it. For the new colors obtained by the CIE76 CIEDE2000
Voronoi method and the brute force method, this is true for Bright green, Cyan, Baby
pink and Pale lavender. When choosing new colors for the Moscow metro, one should
carefully define the boundary of the color space, based on the gamut of the printers
used. This could change some results.

4.8 Discussion

For this particular problem, the CIE76 Voronoi method does not give very significant
results. We see for example that the first new color “Bright green” and the third new
color “Sea green” have only a CIE2000 distance of 13.2. However, for similar problems
where we can use the Euclidean distance, this method not only works very fast; it also
gives you the exact optimum and is therefore a very good method. The Coulomb method
is interesting because it does not only take the nearest point in account, however it
should be improved or ran for more random beginning point to give good results. Using
nonlinear constraint optimization methods could work very well for adding more than
one point. However, it should run longer (for more initial points) or on a faster computer.
This would also improve our results for the brute force methods. Implementing the
random walk method could reduce the computing time as well.

Other research that could be done to improve the choice of the new colors is research
on the choice of $k_L$, $k_C$ and $k_H$ for this particular use: use on maps which are often read
in poor light.
Appendix A

CIEDE2000 color difference formula

Given the CIELAB coordinates $L^*, a^*$ and $b^*$ of two colors 1 and 2, their CIEDE2000 color difference can be calculated the following way.

Step 1. Calculate $C^*$ for both colors:

$$C^* = \sqrt{a^{*2} + b^{*2}}.$$

Step 2. Calulate $a', C'$ and $h'$ for both colors:

$$L' = l^*,$$
$$a' = (1 + G)a^*,$$
$$b' = b^*,$$
$$C' = \sqrt{a'^2 + b'^2},$$
$$h' = \arctan \left( \frac{b'}{a'} \right),$$

where

$$G = 0.5(1 - \sqrt{\frac{\bar{C}^{*2}}{\bar{C}^{*4} + 25}}),$$

where $\bar{C}^*$ is the arithmetic mean of the $C^*$ values for a pair of samples.
step 3. Calculate $\Delta L', \Delta C'$ and $\Delta H'$:

\[
\Delta L' = L_2^* - L_1^*, \\
\Delta C' = C_2^* - C_1^*, \\
\Delta H' = 2\sqrt{C_2'C_1' \sin \left( \frac{\Delta h'}{2} \right)},
\]

where

\[
\Delta h' = h'_2 - h'_1.
\]

Step 4. Calculate $S_L, S_C, S_H$ and $R_T$:

\[
S_L = 1 + 0.015 \frac{(\bar{L}' - 50)^2}{\sqrt{20 + (\bar{L}' - 50)^2}}, \\
S_C = 1 + 0.045 \bar{C}', \\
S_H = 1 + 0.015 \bar{C}'T, \\
R_T = -\sin(2\Delta \theta)R_C,
\]

where $\bar{L}', \bar{C}'$ and $\bar{h}'$ are the arithmetic means of the $L', C'$ and $h'$ values for a pair of samples,

\[
T = 1 - 0.17 \cos(\bar{h}' - 30^\circ) + 0.24 \cos(2\bar{h}') + 0.32 \cos(3\bar{h}' + 6^\circ) - 0.20 \cos(4\bar{h}' - 63^\circ), \\
\Delta \theta = 30 \exp\{-[(\bar{h}' - 275^\circ)/25]^2\}
\]

and

\[
R_C = 2\sqrt{\frac{\bar{C}'^2}{\bar{C}'^2 + 257}}.
\]

Step 5. Calculate the CIEDE2000 color difference $\Delta E_{00}$:

\[
\Delta E_{00} = \sqrt{\left(\frac{\Delta L'}{k_L S_L}\right)^2 + \left(\frac{\Delta C'}{k_C S_C}\right)^2 + \left(\frac{\Delta H'}{k_H S_H}\right)^2 + R_T \left(\frac{\Delta C'}{k_C S_C}\right) \left(\frac{\Delta H'}{k_H S_H}\right)},
\]

where $k_L, k_C$ and $k_H$ are chosen weighting factors.
Appendix B

Matlab code

B.1 Calculate the sideplanes

```matlab
% The cornerpoints of the gamut
Cornerpoints = [0 0 0; 33 80 -109; 67 105 -52; 61 90 75; 97 -23 105; 83 -138 91; 87 -78 -21; 100 0 0];

% Calculate the formulas for the sideplanes of the gamut. Here the i-th row
% of Sideplanes generates the formula the following way:
% "Sideplanes(i,1:3) dot (x,y,z)=Sideplanes(i,4)"
Sideplanes = zeros(12,4);
%1: ABG
normal = cross((Cornerpoints(1,:) - Cornerpoints(2,:)),(Cornerpoints(1,:) - Cornerpoints(7,:)))
Sideplanes(1,:) = [normal dot(normal,Cornerpoints(2,:))];
%2: AFG
normal = cross((Cornerpoints(1,:) - Cornerpoints(6,:)),(Cornerpoints(1,:) - Cornerpoints(7,:)))
Sideplanes(2,:) = [-normal -dot(normal,Cornerpoints(1,:))];
%3: ABD
normal = cross((Cornerpoints(1,:) - Cornerpoints(2,:)),(Cornerpoints(1,:) - Cornerpoints(4,:)))
Sideplanes(3,:) = [-normal -dot(normal,Cornerpoints(1,:))];
%4: BDC
normal = cross((Cornerpoints(2,:) - Cornerpoints(3,:)),(Cornerpoints(2,:) - Cornerpoints(4,:)))
Sideplanes(4,:) = [-normal -dot(normal,Cornerpoints(2,:))];
%5: ADF
normal = cross((Cornerpoints(1,:) - Cornerpoints(4,:)),(Cornerpoints(1,:) - Cornerpoints(6,:)))
Sideplanes(5,:) = [-normal -dot(normal,Cornerpoints(1,:))];
%6: DEF
```
Maximizing color difference in metro maps

normal = cross((Cornerpoints(4,:) - Cornerpoints(5,:)),(Cornerpoints(4,:) - Cornerpoints(6,:)));
Sideplanes(6,:) = [-normal -dot(normal,Cornerpoints(4,:))];

% 7-12: HBC, HCD, HDE, HEF, HFG, RGB
for i=7:12
    normal = cross(Cornerpoints(8,:) - Cornerpoints(1+(i-6,:),:), Cornerpoints(8,:) - Cornerpoints(2+mod((i-6),6),:));
    Sideplanes(i,:) = [normal dot(normal,Cornerpoints(8,:))];
end

B.2 Calculate the minimal distance to all existing points

function [minimumafstand, error, number] = minafstand(x, soort)

% The coordinates of the color-points in CIElab
L = [52 56 35 61 71 40 56 99 3 80 56 99 13]; % 86 33 86.5];%
a = [74 -45 7 -16 6 24 64 4 0 -26 -21 -8 0 2]; % -77 -137.5 79.5 22.5];%
b = [53 26 42 27 64 23 8 72 5 68 -29 -9 0 0 ]; % -20.5 90.5 -180.5 -21.5];%
Points = [transpose(L) transpose(a) transpose(b)];

% calculate the sideplanes
sideplanes;

% Test whether x is in the gamut
error = 0;
for i=1:12
    if sign(dot(x, Sideplanes(i,1:3)) - Sideplanes(i,4)) ~= sign(dot([50 0 0], Sideplanes(i,1:3)) - Sideplanes(i,4))
        error = 1;
    end
end

% set an initial value for "distance", this should be bigger than the
% maximal distance
distance = 100;
for m=1:length(a)
    % "soort 1" is ciede2000
    if soort == 1
        distance1 = deltaE2000(x, Points(m,:), [2 1 1]);
    % "soort 2" is the Euclidean norm
    elseif soort == 2
        distance1 = norm(x - Points(m,:));
    end
    % See if the distance is smaller than the distance to all points before
    if distance1 <= distance
        distance = distance1;
        % denote the closest point
        number = m;
    end
end
minimumafstand = distance;
return
B.3 Calculate vertices with CIE76 method

```matlab
% The coordinates of the color points in CIElab
a = [74 -45 7 -16 6 24 64 4 0 -26 -21 -8 0 2 -137];
b = [53 26 -43 -42 27 67 -24 85 -2 68 -29 -9 0 0 91];
L = [52 56 36 61 41 76 43 86 71 80 56 93 99 13 83];
Points = [transpose(L) transpose(a) transpose(b)];
% Calculate the sideplanes
sideplanes;

% Test whether all points are in the gamut
for n = 1:14
    for i = 1:12
        if sign(dot(Points(n,:), sideplanes(i,1:3)) - sideplanes(i,4)) ~= sign(dot([50, 0, 0], sideplanes(i,1:3)) - sideplanes(i,4))
            n
            i
        end
    end
end

% Ask the user for which point he/she wants to calculate the corner points
% of the voronoi cell
prompt = 'For which point do you want to calculate the corner points of the Voronoi Cell? (number 1-14) 
';
n = input(prompt);
for n = 1:length(a)
    while n<1 || n>14
        n = input(prompt);
    end
    % Calculate the bisectors and store them in Planefunction the same way as Sideplanes.
    Planefunction = zeros(length(a), 4);
    for i = 1:length(a)
        if i ~= n
            middel = (Points(n,:) + Points(i,:))/2;
            normal = Points(n,:) - Points(i,:);
            Planefunction(i,:) = [normal dot(normal, middel)];
        end
    end
    format long g

% put together Planefunction and Sideplanes
Planes = [Planefunction ; sideplanes];

% calculate all intersecting points of three planes
for i = 1:(length(a)+12-2)
    if i ~= n
        for j = (i+1):(length(a)+12-1)
            if j ~= n
                for k = (j+1):(length(a)+12)
                    if k ~= n
```
A = [Planes(i,1:3); Planes(j,1:3); Planes(k,1:3)];
D = det(A);
if abs(D) >= 10^-10
b = [Planes(i,4); Planes(j,4); Planes(k,4)];
x = A \ b;
error = 0;
l = 1;

% Check whether the point is really a
% cornerpoint of the voronoi cell (i.e. xn does
% not cross another bisector or boundary plane).
while (l <= length(a) + 12) && error <= 1

if l ~= n

errorx = sign(round(100*(dot(Planes(l,1:3), x) - Planes(l,4))));
errorn = sign(dot(Planes(l,1:3), Points(n,:)) - Planes(l,4));
error = abs(errorx - errorn);
end
l = l + 1;
end

% Print the points that are cornerpoints and
% their distance to n

distance = norm(x - transpose(Points(n,:)));
if error <= 1 && distance >= 80

error; x
distance
end

end

end

end

end

end

B.4 Calculate vertices with CIE76-CIEDE2000 method

% The coordinates of the color-points in CIELab
L = [52 56 35 61 76 43 86 71 80 56 93 99 13];
% 86 83 33 86.5];%

a = [74 -45 7 -16 6 24 64 4 0 -26 -21 -8 0 2];%
% -77 -137.5 79.5 22.5];%
b = [53 26 -43 -42 27 67 -24 85 -2 68 -29 -9 0 0];
% -20.5 90.5 -108.5 -21.5];%

Points = [transpose(L) transpose(a) transpose(b)];

% Calculate the sideplanes
sideplanes;
for n=1:14
    for i=1:12
        if sign(dot(Points(n,:),Sideplanes(i,1:3))-Sideplanes(i,4)) ~= sign(dot([50 0 0],Sideplanes(i,1:3))-Sideplanes(i,4))
            n
        end
        end
    end
% Ask the user for which point he/she wants to calculate the corner points
% of the voronoi cell
for n=1:length(a)
    % Calculate the bisectors and store them in Planefunction the same way as
    % for Sideplanes.
    Planefunction= zeros(length(a),4);
    for i=1:length(a)
        if i~=n
            middle=(Points(n,:)+Points(i,:))/2;
            normal = Points(n,:)-Points(i,:);
            Planefunction(i,:)=[normal dot(normal,middle)];
        end
    end
    format long g
    % put together Planefunction and Sideplanes
    Planes = [Planefunction ; Sideplanes];
    % calculate all intersecting points of three planes
    for i=1:(length(a)+12-2)
        if i~=n
            for j=(i+1):(length(a)+12-1)
                if j~=n
                    for k=(j+1):(length(a)+12)
                        if k~=n
                            A=[Planes(i,1:3); Planes(j,1:3) ; Planes(k,1:3)];
                            D=det(A);
                            if abs(D)>10^-10
                                b=[Planes(i,4); Planes(j,4) ; Planes(k,4)];
                                x=A\b;
                                error=0;
                                l=1;
                                % check whether the point is really a
                                % cornerpoint of the voronoi cell (i.e. xn does
                                % not cross another bisector or boundary plane).
                                while (l<=(length(a)+12) && error<1)
                                    if l~=n
                                        errorx=sign(round(100*(dot(Planefunction(l,1:3),
                                            x)-Planefunction(l,4))));
                                        errorn=sign(dot(Planefunction(l,1:3),Points(n,:))
                                            -Planefunction(l,4));
                                    end
                                    end
                                    l=l+1;
                                    error=error+abs(x(l-1));
                            end
                        end
                    end
                end
            end
        end
    end
end
error = abs(errorx - errorn);
end
l = l + 1;
end

% print the points that are cornerpoints and
% their distance to n
if error <= 1
    distance1 = 100;
    distance = 100;
    for m = 1:length(a)
        distance1 = deltaE2000(transpose(x), Points(m,:),[2 1 1]);
        if distance1 <= distance
            distance = distance1;
            number = m;
        end
    end
end

B.5 Plot the colors in Cielab

a = [74 -45 5 -16 6 24 64 4 0 -26 -21 -8 0 2 -138]
b = [53 26 -49 -42 27 67 -24 85 -2 68 -29 -9 0 0 91]
L = [52 56 34 61 41 76 43 86 71 80 56 93 100 13 83]
C = [0.95 0.07 0.174; 0 0.6 0.36; 0 0.33 0.61; 0 0.62 0.84;
     0.46 0.36 0.22; 1 0.66 0.3; 0.73 0.13 0.54; 1 0.82 0.28;
     0.67 0.68 0.69; 0.69 0.82 0.33; 0 0.58 0.71; 0.82 0.94 0.98; 0 0 0; 0.13 0.12
     0.13; 0 1 0]
scatter3(a,b,L,60,C,'fill')
title('points in Cielab space');
xlabel('a');
ylabel('b');
zlabel('L');
B.6 One by one brute force method

```matlab
% Calculate the sideplanes
sideplanes; % Set the initial value for the maximal distance.
bestdistance = 0;

% Shoot 100,000 times 2 random points
for k = 1:100000
    buiten = 0;
    x1 = [100*rand 200*rand -100 200*rand -100];
    % Check whether x1 is in the gamut
    for i = 1:12
        if sign(dot(x1,Sideplanes(i,1:3)) - Sideplanes(i,4)) != sign(dot([50 0 0], Sideplanes(i,1:3)) - Sideplanes(i,4))
            buiten = 1;
        end
    end
    if buiten == 0
        % Calculate the minimal distance to P
        [min, fout, nummer] = minafstand(x1,1);
        if fout == 0
            % See whether it is better than the current minimal distance
            if min >= bestdistance
                bestpoint = x1
                bestdistance = min
                nummer
            end
        end
    end
end
```

B.7 Simultaneous brute force method

```matlab
% Calculate the sideplanes
sideplanes; % Set the initial value for the maximal distance.
bestdistance = 0;

% Shoot 1,000,000 times 2 random points
for k = 1:1000000
    buiten = 0;
    x1 = [100*rand 200*rand -100 200*rand -100];
    x2 = [100*rand 200*rand -100 200*rand -100];
    % Check whether they are in the gamut
    for i = 1:12
        if sign(dot(x1,Sideplanes(i,1:3)) - Sideplanes(i,4)) != sign(dot([50 0 0], Sideplanes(i,1:3)) - Sideplanes(i,4))
            buiten = 1;
        end
    end
end
```
Maximizing color difference in metro maps

if \( \text{sign}(\dot{z}_2, \text{Sideplanes}(i,1:3)) - \text{Sideplanes}(i,4)) = \text{sign}(\dot{\text{dot}}([50 0 0], \text{Sideplanes}(i,1:3)) - \text{Sideplanes}(i,4))\)

\( \text{buiten}=1; \)

end
if buiten==0

% Calculate the minimal distance to \( P \) and to eachother
[\text{min1}, \text{fout1}, \text{nummer1}]= \text{minafstand}(x1,1);
[\text{min2}, \text{fout2}, \text{nummer2}]= \text{minafstand}(x2,1);
tussen = \text{deltaE2000}(x1,x2,[2 1 1]);
A=[\text{min1}; \text{min2}; \text{tussen}];
minimum=min(A);
if \text{fout1}==0 && \text{fout2}==0

% See whether it is better than the current minimal distance
if minimum>=\text{bestdistance}
bestpoint1=x1
bestpoint2=x2
bestdistance=minimum
end
end
end

B.8 Coulomb repulsion with CIEDE2000

L=[52 56 34 61 41 76 43 86 71 80 56 93 99 13];
a=[74 -45 5 -16 6 24 64 4 0 -26 -21 -8 0 2];
b=[53 26 -49 -42 27 67 -24 85 -2 68 -29 -9 0 0];
Points=[\text{transpose}(L) \text{transpose}(a) \text{transpose}(b)];

% Calculate the sideplanes
sideplanes;

% set the initial values for our new point
dx1= [0 0 0];
ddx1= [0 0 0];

% Set the values of the constants
k=10;
\gamma = 0.1;
c=0.2;
dt=0.1;
\text{bestdistance}=0;
bestpoint= [50 0 0];

% Run the algorithm for 300 random startingpoints
for \( k=1:600 \)
buiten=0;
x1= [100*\text{rand} 200*\text{rand}-100 200*\text{rand}-100];

% See whether the starting point is within the gamut
for i=1:12
    if sign(dot(x1,Sideplanes(i,1:3))-Sideplanes(i,4)) ~= sign(dot([50 0 0], Sideplanes(i,1:3))-Sideplanes(i,4))
        buiten=1;
    end
end
if buiten==0
    % If the starting point is in the gamut, let the coulomb repulsion
    % algorithm run for 2000 timesteps
    for t=1:2000
        x2 = x1+dx1*dt;
        dx2 = dx1+ddx1*dt;
        % calculate the contribution of the old points to ddx2
        for i=1:length(a)
            sum = (1/deltaE2000(x2,Points(i,:),[2 1 1])^3)*(x2-Points(i,:));
        end
        % calculate the closest boundary point
        mindistance=300;
        for j=1:12
            time = (dot(Sideplanes(j,1:3),x2)+Sideplanes(j,4))/dot(Sideplanes(j,1:3),Sideplanes(j,1:3));
            b=x2+time*Sideplanes(j,1:3);
            distance=deltaE2000(x2,b,[2 1 1]);
            if distance<=mindistance
                mindistance=distance;
            end
        end
        ddx2 = -gamma*dx2 +k*sum +k*c/deltaE2000(x2,b,[2 1 1])^3*(x2-b);
        % set the new values for x, dx and ddx
        x1=x2;
        dx1=dx2;
        ddx1=ddx2;
    end
    % calculate the minimal distance of the point obtained.
    [min, fout, nummer] = minafstand(x1,1);
    if fout==0
        if min>=bestdistance
            bestpoint=x1
            bestdistance=min
            nummer
        end
    end
end
end
bestpoint
bestdistance

B.9 Solve optimization problem with fmincon
sideplanes
A = [zeros(24, 1) Sideplanes(1:12, 1:3); zeros(12, 3); Sideplanes(1:12, 1:3)];
b = [Sideplanes(1:12, 4); Sideplanes(1:12, 4)];
Aeq = []; beq = [];
lb = []; ub = [];
bestdistance = 0;

% Calculate the sideplanes
sideplanes;
% Set the initial value for the maximal distance.
bestdistance = 0;

% optimlib: fmincon: SwitchingToMediumScale
options = optimset('Display', 'off', 'Algorithm', 'active-set');

% Shoot 1,000,000 times 2 random points
for k = 1:1000000
    buiten = 0;
    x1 = [100 * rand 200 * rand -100 200 * rand -100];
    x2 = [100 * rand 200 * rand -100 200 * rand -100];
    % Check whether they are in the gamut
    for i = 1:12
        if sign(dot(x1, Sideplanes(i, 1:3)) - Sideplanes(i, 4)) ~= sign(dot([50 0 0], Sideplanes(i, 1:3)) - Sideplanes(i, 4))
            buiten = 1;
        end
        if sign(dot(x2, Sideplanes(i, 1:3)) - Sideplanes(i, 4)) ~= sign(dot([50 0 0], Sideplanes(i, 1:3)) - Sideplanes(i, 4))
            buiten = 1;
        end
    end
    if buiten == 0
        % Calculate the minimal distance to P and to eachother
        x = fmincon(@x - x(1), [0 x1 x2], A, b, Aeq, beq, lb, ub, @nolcon, options);
        [min1, fout1, numer1] = minafstand(x(2:4), 1);
        [min2, fout2, numer2] = minafstand(x(5:7), 1);
        tussen = deltaE2000(x(2:4), x(5:7), [2 1 1]);
        Q = [min1; min2; tussen];
        minimalm = min(Q);
        if fout1 == 0 && fout2 == 0
            % See whether it is better than the current minimal distance
            if minimalm > bestdistance
                bestpoint1 = x1
                bestpoint2 = x2
                bestdistance = minimalm
            end
        end
    end
end
Bibliography


