UNIVERSITY OF NOVI SAD

MASTER THESIS

Semidefinite Programming Approach to Visual Storyline Creation

Author:
Kristina Licenberger

Supervisor:
PhD Dragana Bajović

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

at the

Faculty of Sciences
Department of Mathematics and Informatics

October, 2019

Declaration of Authorship

I, Kristina Licenberger, declare that this thesis titled, "Semidefinite Programming Approach to Visual Storyline Creation" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____________________________________

Date: ____________________________________

UNIVERSITY OF NOVI SAD

Abstract

Faculty of Sciences
Department of Mathematics and Informatics

Master of Science

Semidefinite Programming Approach to Visual Storyline Creation

by Kristina Licenberger

This thesis formulates the problem of finding minimal weight path in a graph, such that the individual weights along the path are as balanced as possible. The motivation for this problem comes from the recent work by G. Marcelino et al. that uses shortest path to find the most coherent sequence of images that illustrate a given event. Therein, the authors define a weight of an edge connecting two images as (1 - the quality of the transition) between the two images (the lower the better). In this thesis, we modify the preceding problem such that we seek for the path with best (lowest) transitions, but also where the transitions are mutually balanced. This is achieved by adding an extra cost term in the minimal path problem that penalizes differences between different transitions. To capture the differences in the transitions, we use a quadratic function. The preceding problem is a $(0, 1)$-quadratic program. As a solution, we propose a semidefinite programming (SDP) relaxation. We tested the algorithm on data containing flower images and on data from the Edinburgh Festival 2016. Although the SDP solution has higher complexity than the shortest path, in most cases it has proved to be worth finding a solution with our algorithm.

Acknowledgements

I would like to express gratitude to:

- *My advisor Prof. Dragana Bajović, for support, generously sharing of knowledge and advices. You inspired me to be better at what I do.*
- *Prof. João Magalhães, for all the suggestions, patience and encouragement. Working with you was a wonderful experience. Without your guidance nothing of this would be possible.*
- *David Semedo, for the time you spent helping me with my work. I learned a lot in computer science from you. You showed me how to make complicated things simple.*
- *Thales Parreira, for giving me the data about the Edinburgh festival.*
- *Lab P3.13 and Flávio Martins, for nice atmosphere during the work and for showing me the Lisbon. Foi um prazer fazer parte do seu laboratório.*
- *My friends and family, for helping me collect moments and chase dreams all these years. Thank you for always being there for me.*

Contents

List of Figures

List of Abbreviations

SP	**S**hortest **P**ath
SPS	**S**hortest **P**ath **S**olution
SPBC	**S**hortest **P**ath with **B**alanced **C**osts
SPBCS	**S**hortest **P**ath with **B**alanced **C**osts **S**olution
SDP	**S**emi**d**efinite **P**rogramming
SDPS	**S**emi**d**efinite **P**rogramming **S**olution
SDPS1	**S**emi**d**efinite **P**rogramming **S**olution Rounding Procedure **1**
SDPS2	**S**emi**d**efinite **P**rogramming **S**olution Rounding Procedure **2**
QP	**Q**uadratic **P**rogramming

Chapter 1

Introduction

1.1 Social media

An online source of information that is created, circulated and used by consumers is known as social media [35]. Social media content has become a part of everyday life. Its purpose is to help people both find new information and have fun. But also, it can help them to find new information in a fun-way. One way to obtain new information in a fun-way is to deduce it from the content of an image. It has been known since ancient times that an image is worth a thousand words, in truth some events are difficult to describe with a small amount of words and this is why images are a good tool to describe an event. The largest source of images is the Internet. People post daily on social networks images of all kinds of events, so every day the amount of images on the Internet is growing with tremendous speed.

In such a big collection, it is not so easy to choose a set of images which will best describe an event as a visually pleasant story. Relevance and many other factors influence the choice. It is also very important to be aware of similarities or dissimilarities between images, in a sequence describing one event. A bunch of images may describe the event as it is, but it does not necessarily mean that it will be pleasing to the human eye. For a set of images to represent the event nicely, the transition from image to image must be taken into account. Transitions are very important if we want to achieve the effect of a harmonious story. Finding a set of images that will best describe an event is a very challenging task for journalists.

One way to approach this problem is to build a graph out of the given images and search for the best path in the graph. This problem has already been discussed in [21]. The authors explained how a story from an event can be divided into segments. They considered what features might affect the relevance of the image and what were the possible ways to select the images that would best represent each segment.

The task addressed by this thesis is on given graph with segments and story illustration candidates for each segment to try a new way of selecting those images that will best represent the story. Images that we choose for solution should be as pleasant as possible to the human eye.

1.2 Stories as paths in Bipartite Graphs

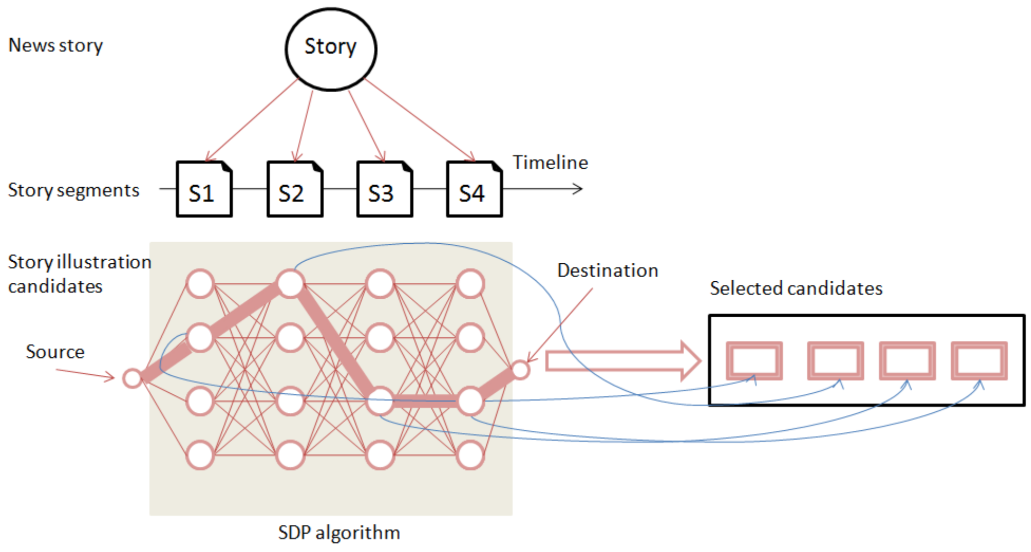


FIGURE 1.1: Selecting images [21]

From [21], we know that we can divide an event into stories and stories into several segments. For each segment we have a number of images that can represent it. One natural way is to use bipartite graphs as a tool to represent stories. A characteristic of a complete bipartite graph is that it consists of two partitions where all the nodes in different partitions are connected to each other and nodes from the same partition are not connected to each other. The point is that these complete bipartite graphs can be nicely applied to describe an event with images. Let's say the smallest example is that one story consists of two segments. For each of the segments we can have several images that describe them. We can represent an event as a bipartite graph and the segments in that case represent partitions where images are actually graph nodes. Edges in this kind of bipartite graph should represent some kind of relationship between images. The way we select images from partitions can be represented as the way we find a path in a graph. It is important to note that we can choose only one image per segment. If a story has more than two segments, then the graph representation generalizes to a sequence of bipartite graphs.[1]

1.3 Problem setup

Suppose we are given a set of images describing a certain event. The event is segmented into several parts, where the segmentation is defined either in terms of time, location etc., and our goal is to find a representative image for each segment, such that the sequence of chosen images faithfully represents the event as a whole. Suppose there are K segments, indexed by $k = 1, 2, ..., K$. As a first step, to each segment k is first assigned a representative set of images $\mathcal{I}_k$, extracted using machine learning tools from the textual information about the event and images of the event, for

[1]In [21] the authors consider sequence of complete bipartite graphs. To solve problem of describing event with images they used a variation of Dijkstra's minimum cost path algorithm.

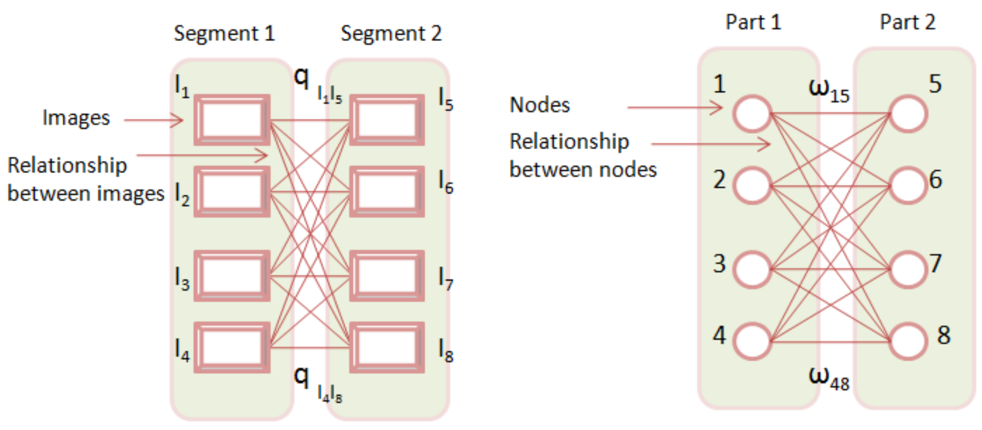


FIGURE 1.2: Bipartite graph

$k = 1, ..., K$. Second, for each pair of images I_k and I_l corresponding to different segments, $I_k \in \mathcal{I}_k$ and $I_l \in \mathcal{I}_l$, $k \neq l$, the "smoothness" of (visual) transition between I_k and I_l (the smoother the better) is defined through function $q : \mathbf{I} \times \mathbf{I} \mapsto [0, 1]$, where $\mathbf{I}$ denotes the set of all the images in a graph. The goal then is to find the sequence of images $I_k^\star$, $k = 1, ..., K$, such that the sum of consecutive transitions, $\sum_{k=1}^{K-1} q(I_k^\star, I_{k+1}^\star)$, is the smallest possible. Explained notations can be viewed in Fig.1.2.

The above problem can be casted as the (minimal) shortest path (SP) problem. To arrive at the SP formulation, we first explain how weighted graph G is built from the given set of images and function q. For each k, define the set of nodes $V_k = \mathcal{I}_k$, that is, each node in V_k corresponds to one image in $\mathcal{I}_k$. To model the transitions, we make all-to-all connections for any two consecutive segments, that is, $u \in V_k$ and $v \in V_l$ are connected by an edge if $l = k + 1, k = 1, ..., K - 1$[2]. Also, each of the edges (u, v) is assigned a weight w_{uv} equal to the value of function $q(I_u, I_v)$ between the images corresponding to nodes u and v.

For the purpose of casting the defined problem in an optimization problem (see Chapter 3), we also introduce two distinct nodes, s (source) and d (destination), and we let $V_0 = \{s\}$ and $V_{K+1} = \{d\}$. We connect node s to each of the nodes in V_1, and connect d to each of the nodes in V_K; these connections are assigned zero weight. Summarising, the graph $G = (V, E)$ is formed with the set of nodes $V = \cup_{k=0}^{K+1} V_k$, the set of edges $E = \cup_{k=0}^{K} V_k \times V_{k+1}$, and each edge $e = (u, v)$ has the associated weight w_{uv}. Finally, we denote the number of nodes in each segment with $n_k = |V_k|$, for $k = 1, ..., K$, $N = |V|$ and number of edges with $M = |E|$. Then, $N = 2 + n_1 + ... + n_K$, and, by the structure of the graph G, it is easy to see that $M = n_1 + n_1 n_2 + ... + n_{K-1} n_K + n_K$.

[2]In [21] the authors also consider fully-multipartite graphs, where connections are formed even between the non-consecutive groups. In this model, the SP problem is replaced by the objective of finding the minimal all-to-all transitions sum (and not only consecutive transitions sum), across all possible sequences of images. For now we focus on consecutive connections only.

1.4 Contribution

- **Shortest path with balanced costs**:
  A novel approach for finding the path in a weighted graph such that the weights of the edges are as balanced as possible.
- **Public implementation**:
  https://github.com/451kica/StoryGraphs/blob/master/Solver.py
- **Real-life data application**:
  This thesis demonstrates how complex mathematical algorithms can be successfully applied to real-life data.

1.5 Document organization

Within the described optimization framework, Chapter 1 introduces and motivates the novel metric of balanced costs.

The next Chapter "Research Hypothesis" consists of sections Shortest path (SP) Problem and SP with balanced costs. We will explain how the shortest path problem can be reformulated as a linear program. Also we will explain our intuition and hypothesis - SP with SPBC reformulation.

The third Chapter named "Semidefinite Programming Relaxation" covers the theory used in the thesis. The transformation of the problem is explained and an illustrative example is given. The steps of implementing the algorithm are also explained. In this chapter we have sections: Semidefinite Proogramming, SPBC via SDP relaxation, Illustration Graph Example and Implementation.

The Chapter "Experiments" contains the three experiments and the results of the experiments. Experiments are divided in sections, so we have Syntetic Data section, Flowers dataset section and Social media storytelling section.

The final Chapter "Conclusion and Future Work" discusses the conclusion and flexibility of framework.

Chapter 2

Research Hypothesis

2.1 Shortest path (SP) problem

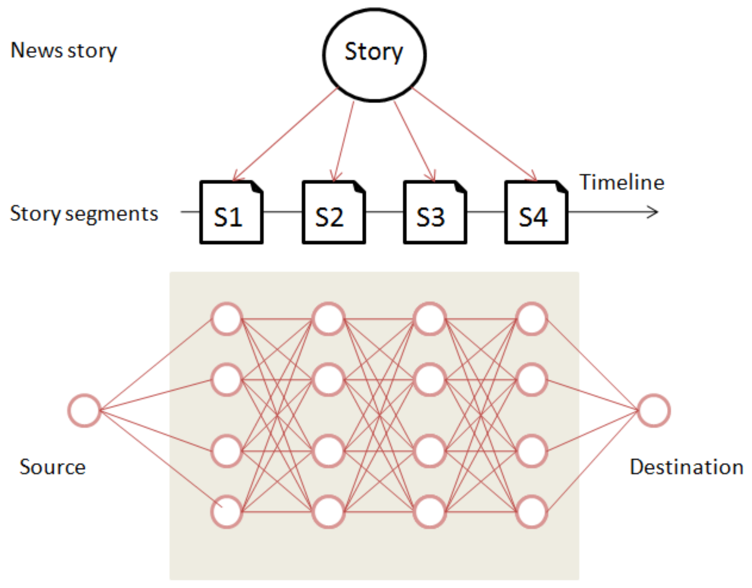


FIGURE 2.1: Graph with source and destination node

The goal is to find the sequence of nodes $v_k \in V_k$, $k = 1, ..., K$ (one from each k), such that the total cost along the path connecting the nodes along the sequence, $\sum_{k=1}^{K-1} w_{v_k,v_{k+1}}$ is minimal. Using nodes s and d and the fact that their connections have zero cost, the preceding objective can be equivalently written as $\sum_{k=0}^{K} w_{v_k,v_{k+1}}$.

For each edge $e \in E$, we introduce a new variable $x_e \in \{0, 1\}$ such that $x_e = 1$ if edge e is chosen and $x_e = 0$ otherwise. By x we denote the vector that stacks all the edge variables $x_e, e \in E$ by a given ordering (please see Chapter 3, section "SDP relaxation" for details on the ordering of edges used in this thesis). Then, the shortest path problem can be represented as the following boolean linear program [13]:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \sum_{e \in E} w_e x_e \\ \text{subject to} & \sum_{u \in V_1} x_{su} = 1 \\ & \sum_{v \in V_K} x_{vd} = 1 \\ & \sum_{u:uv \in E} x_{uv} = \sum_{u:vu \in E} x_{vu},\ v \in V \setminus \{s \cup d\} \\ & x_e \in \{0, 1\},\ e \in E \end{array} . \tag{2.1}$$

To explain the equivalence, suppose that a unit flow starts from the start node s, as given in the first constraint of (2.2). The flow can then choose the next edge by setting the corresponding variable x_{su} to 1, for $u \in V_1$ (note that, by the first constraint and the last constraint, if one of the edge variables connected to s is set to one, all the remaining edge variables connected to s must be equal to 0); for example, if node $u \in V_1$ is chosen, then $x_{su} = 1$ and $x_{sv} = 0$, for each $v \in V_1 \setminus \{u\}$. The third constraint is the flow conservation condition: at any node inside the graph, the total inflow must be equal to the total outflow. Since the edge variables x_u can be either zero or one, the flow cannot split into fractional flows. Also, since the inflow must be equal to the outflow, it also cannot branch out. Thus, at any given node v different than s and d only two situations can occur: 1) either both the inflow and outflow are zero (which happens in the case when $x_{uv} = 0$ for all u such that $(u, v) \in E$ and also $x_{vu} = 0$ for all $(v, u) \in E$), and v is not on the flow path; or 2) the inflow and outflow are equal to 1, which happens when exactly one of the x_{uv}'s equals 1, and exactly one of the x_{vu}'s equals 1, i.e., v is on the flow path.

The preceding problem is an integer linear program (ILP) [31]. ILPs are in general hard to solve optimally. For the shortest path (SP) problem with non-negative edge costs, it is well-known that if the relaxation from $x_e \in \{0, 1\}$ to $x_e \geq 0$ results in fact in an equivalent problem, and hence the optimal solution of (2.1) is found by:

$$\begin{array}{lc} \underset{x}{\text{minimize}} & \sum_{e \in E} w_e x_e \\ \text{subject to} & \sum_{u \in V_1} x_{su} = 1 \\ & \sum_{v \in V_K} x_{vd} = 1 \\ & \sum_{u:uv \in E} x_{uv} = \sum_{u:vu \in E} x_{vu},\ v \in V \setminus \{s \cup d\} \\ & x_e \geq 0,\ e \in E \end{array} . \tag{2.2}$$

2.2 SP with balanced costs (SPBC)

2.2.1 Intuition

We now explain the modification of the SP problem that we propose, that could maybe lead to an improved performance of the chosen sequence of images. To motivate the approach, we refer to Figure 2.1. taken from [21] (see Fig 6.11, p. 78 in [21]).

Specifically, the figure presents results of two (best performing) approaches of storyline creation [21], namely FullT and SeqT, represented by the first and the third image sequence in Figure 6.11 from [21]. First, we look at the FullT approach shown in the first image sequence in Figure 2.1. Looking at the sequence, we can observe that the first and the last pair of images are visually similar, whereas the transition between the second and the third image in the sequence is sharper. A similar effect, and maybe also more prominently occurring, can be observed with the SeqT approach, shown in the third sequence in Figure 6.11. With SeqT, the first two images in the sequence both show two performers on a stage (two musicians and two actors), and also in a very similar setup (e.g., size and orientation of the human figures, and also their mutual orientation are both very similar); the last two images both show one performer (circus actor, street performance actor) on the background with sharp edges (the layouts of the two images exhibit similarity when counterclockwise rotation is applied to the last image). The transition between the second and the third image is evidently sharp. An alternative sequence could possibly have a smoother middle transition, but this would certainly happen at the expense of increasing the

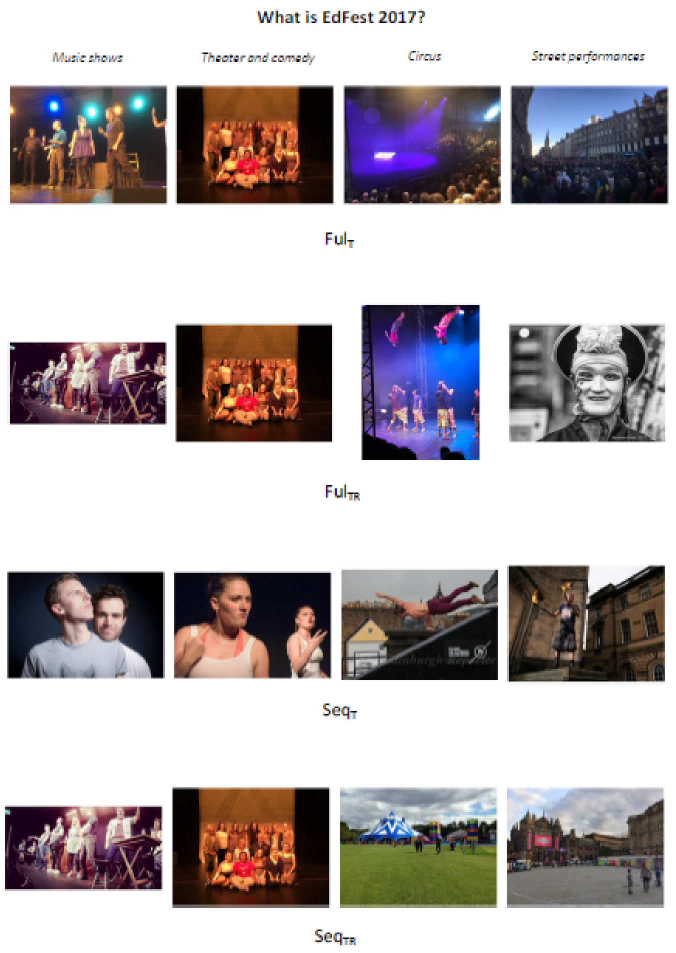


FIGURE 2.2: Fig. 6.11 [21]

two remaining transitions, since the sequence that SeqT approach outputs is guaranteed to have minimal total transition costs.

There are two things that are interesting to note here. First, it would of course be better to smooth out any sharp transitions. And second, it could potentially be beneficial to actually increase the transition between images that are too similar (the higher the transition, the higher the amount of new information[1]). Motivated by this observation, we propose to relax the minimal total transition cost criterion by allowing transitions that are of higher cost, but more balanced, to win.

[1]We also had some ideas in this direction, that is, to explore alternative definitions of the trans function. Specifically, we were thinking of designing trans that is non-monotonous in certain features such that it forces some features (e.g., color histogram) to be close, but let others (e.g., semantic content) increase by an optimal amount, to allow for information gain. Defining the optimal point (for, e.g., semantic deviation) seemed very interesting to explore further.

2.2.2 Hypothesis - SP with SPBC reformulation

The preceding objective can be modelled by adding an extra term in the objective of (2.1) that penalizes the differences between the weights of adjacent edges. Before forming the objective, we introduce term of segment-adjacent edges. Segment-adjacent edges actually represent two edges which are adjacent and connecting different segment pairs. The formal definition of segment-adjacent pair of edges is as follows: edges e and f are segment-adjacent if $e = (u, v)$ and $f = (v, w)$, where $u \in V_{k-1}$, $v \in V_k$ and $w \in V_{k+1}$. To reformulate the objective function, we choose quadratic penalty and modify (2.1) as follows:

$$\begin{array}{ll}\underset{x}{\text{minimize}} & \sum_{e\in E} w_e x_e + \lambda \sum_{\substack{e,f\in E \\ e,f \text{ segment-adjacent}}} \frac{(w_e-w_f)^2}{2} x_e x_f \\ \text{subject to} & \sum_{u\in V_1} x_{su} = 1 \\ & \sum_{v\in V_K} x_{vd} = 1 \\ & \sum_{u:uv\in E} x_{uv} = \sum_{u:vu\in E} x_{vu},\ v \in V \setminus \{s \cup d\} \\ & x_e \in \{0,1\},\ e\in E\end{array}, \tag{2.3}$$

where λ is the penalty factor. We call problem (3.1) shortest path with balanced costs (SPBC) problem[2].

The first sum in the objective refers to the cost of the chosen path, computed as the sum of the weights along the path. It will be the sum of the weights of the selected edges in the graph. The second sum is a regularizer that compares transitions between all consecutive pairs of segments. Therefore, we added a difference of segment-adjacent edges to the objective function. The meaning of the regularizer in (3.1) is intuitive. Consider two fixed, adjacent edges $e, f \in E$. If both x_e and x_f are equal to 1, then these two edges lie on the chosen path and, since the product $x_e x_f$ equals 1, the appropriate penalty term $(w_e - w_f)^2$ is added to the objective. On the other hand, if either x_e or x_f equals zero, then the corresponding edge is not on the chosen path, and hence the penalty term $(w_e - w_f)^2$ should not participate in the objective. The latter is ensured by the product $x_e x_f$ which is zero in this case.

Namely, the condition $x_e \in \{0, 1\}$ can not be replaced with $x_e \geq 0$ because x is defined such that it can only take values 0 or 1 (edge is selected $x = 1$ or edge is not selected $x = 0$). However, the condition $x_e \in \{0, 1\}$ can still be reformulated in another way, if the problem itself is reformulated.

The SPBC problem (3.1) is a $(0, 1)$-quadratic problem and is hard to solve in general. In the next chapter we will define an approach based on semidefinite programming relaxation.

[2]An alternative way to define the penalty term in (3.1) would be to account for the differences between, not only the consecutive edges along the path, but between each pair of edges on the path. This can be achieved by enlarging the summation of the penalty term to go over all possible pairs of edges. As the two formulations seem to be very similar (i.e., with the same goal), for the present work, we considered only formulation in (3.1), but we remark that the generalization will be trivial.

Chapter 3

Semidefinite Programming Relaxation

3.1 Semidefinite programming

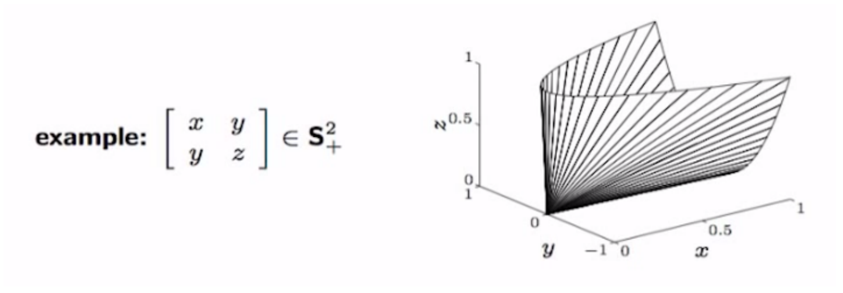


FIGURE 3.1: Positive semidefinite cone [30] - on left side is an example of matrix (of dimensions 2x2) where elements from a matrix are found in a positive semidefinite cone

This chapter introduces SDP relaxation and explains how it can be used to solve probem (3.1). In relaxation procedures, the aim is to approximate a difficult problem to a similar problem that is easier to solve. The solution of the relaxed problem will provide useful information about the solution of the original problem [12].

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \sum_{e\in E} w_e x_e + \lambda \sum_{\substack{e,f\in E \\ e,f \text{ segment-adjacent}}} \frac{(w_e-w_f)^2}{2} x_e x_f \\ \text{subject to} & \sum_{u\in V_1} x_{su} = 1 \\ & \sum_{v\in V_K} x_{vd} = 1 \\ & \sum_{u:uv\in E} x_{uv} = \sum_{u:vu\in E} x_{vu},\ v \in V \setminus \{s \cup d\} \\ & x_e \in \{0,1\},\ e \in E \end{array}, \tag{3.1}$$

Semidefinite programming is a type of convex programming defined over the (convex) set of positive semidefinite matrices[1] of a given dimension, and where the objective function and the constraints are linear [24]. Semidefinite programs are easier but slower to solve than linear programs and they are more general than linear programs. Minimizing a linear function of a matrix subject to linear equality and inequality constraints, where the inequalities include membership of the cone of positive semidefinite matrices is actually semidefinite programming [8].

[1]A matrix Z is positive semidefinite if $\forall v \in R^n$ holds that $v^T Z v \geq 0$.

Standard SDP has the form

$$\begin{array}{ll} \underset{Z}{\text{minimize}} & \operatorname{tr}(WZ) \\ \text{subject to} & \operatorname{tr}(A_i Z) = c_i,\ i = 1, ..., p \ \cdot \\ & Z \succeq 0 \end{array} \qquad (3.2)$$

where $W, Z, A_1, ...A_p \in S^n$, and where S^n is the cone of positive semidefinite $n \times n$ matrices. [33]

3.2 SPBC via SDP relaxation

SDP has wide applicability in many areas - convex optimization, combinatorial optimization, control theory... Many optimization problems have convex relaxations that are semidefinite programs and the optimal solution to the SDP relaxation can be converted to a feasible solution for the original problem [11]. The relaxation is a way of enlarging the original, usually nonconvex constraint set to achieve a convex one. An example of the use of SDP in combinatorial optimization is an SDP relaxation of the MAX CUT problem, for more information about this, the reader is refered to [25]. The constraints such as convex quadratic inequalities, lower bounds on determinants of symmetric positive semidefinite matrices, linear inequalities can be modeled in the SDP framework [11].

For the purpose of explanation, it is important to formally define quadratic form as $z^T Qz + 2z^T c$, where Q is a semidefinite matrix of a given dimension, and c is a vector of the same dimension. We propose a solution based on semidefinite relaxation, as explained in [27], see also [33]. In [27], they have shown that semidefinite relaxation are equivalent to quadratic relaxations. Also, they have shown that $(0, 1)$ quadratic programming form (of form $z^T Qz + 2z^T c$) is equivalent to $(-1, 1)$ quadratic programming problem via transformation $z = 2x - 1$. Inspired by that observation, we choose to reformulate our $(0, 1)$ quadratic programming problem to $(-1, 1)$ quadratic programming problem in order to apply semidefinite relaxation. The QP via SDP approach can be explained in three steps. So, first step is transformation, second step is SDP relaxation and third step is rounding. We detail each of the three steps in the following three sections. Thus, in the remainder of the chapter, we show how we solve the hard, integer QP problem 2.3 via a convex, SDP one 3.1.

We did not properly introduce optimization framework, neither we described how the quadratic cost arises. Our objective consist of four steps:

- First step is to formulate the problem as an integer quadratic program.
- Second step is to relax the problem to a semidefinite program.
- Third step is to solve the relaxation, obtaining a vector solution $z_1, ..., z_n$.
- And fourth step is to round the vector solution $z_1, ..., z_n$ to an integer solution $x_1, ..., x_n$ [24].

3.2.1 Transformation from $(0,1)$-QP to $(-1,1)$-QP

Recall that $(0,1)$-QP optimization problem is :

$$\begin{array}{ll}\underset{x}{\text{minimize}} & \sum_{e\in E} w_e x_e + \lambda \sum_{\substack{e,f\in E \\ e,f \text{ segment-adjacent}}} \frac{(w_e-w_f)^2}{2} x_e x_f \\ \text{subject to} & \sum_{u\in V_1} x_{su} = 1 \\ & \sum_{v\in V_K} x_{vd} = 1 \\ & \sum_{u:uv\in E} x_{uv} = \sum_{u:vu\in E} x_{vu},\ v\in V\setminus\{s\cup d\} \\ & x_e\in\{0,1\},\ e\in E\end{array}.$$

Transformation of the objectives

Transformation of the first term

We start by substituting the $(0,1)$ variables x_e by $(-1,1)$ variables obtained through the transformation $z_e = 2x_e - 1$, for each $e \in E$. Then, $x_e = \frac{z_e+1}{2}$. The first term of the objective (3.1) transforms into

$$\sum_{e\in E} w_e x_e = \sum_{e\in E} w_e \frac{z_e+1}{2} = \frac{1}{2}\sum_{e\in E} w_e z_e + \frac{1}{2}\sum_{e\in E} w_e$$

Now, we can see that $\sum_{e\in E} w_e$ is equal to number of edges M. Because of that, we got that first term of our new objective is

$$\frac{1}{2}\sum_{e\in E} w_e z_e + \frac{M}{2}, \tag{3.3}$$

where we note that $\frac{M}{2}$ is a constant and will not affect the optimization problem because z does not depend on M.

Transformation of the second term

The second term transforms into

$$\begin{aligned}&\lambda \sum_{\substack{e,f\in E \\ e,f \text{ segment-adjacent}}} \frac{(w_e-w_f)^2}{2}\frac{z_e+1}{2}\frac{z_f+1}{2} \\ =&\lambda\cdot\frac{1}{8} \sum_{\substack{e,f\in E \\ e,f \text{ segment-adjacent}}} (w_e-w_f)^2(z_e+1)(z_f+1) \\ =&\lambda\cdot\frac{1}{8} \sum_{\substack{e,f\in E \\ e,f \text{ segment-adjacent}}} (w_e-w_f)^2(z_e z_f+z_e+z_f+1).\end{aligned}$$

We can notice the terms $\sum_{e\in E} z_e$ and $\sum_{f\in E} z_f$ in the preceding equation are actually equal. Hence, we have that $\sum_{e\in E} z_e + \sum_{f\in E} z_f$ is equal to $2\sum_{e\in E} z_e$, which then yields:

$$\begin{aligned}
&\lambda \cdot \frac{1}{8} \sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} (w_e - w_f)^2 (z_e z_f + 2z_e + 1) \\
&= \lambda \cdot \frac{1}{8} \sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} (w_e - w_f)^2 z_e z_f \\
&+ \quad \lambda \cdot \frac{1}{8} \sum_{e \in E} 2z_e \sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} (w_e - w_f)^2 \\
&+ \quad \lambda \cdot \frac{1}{8} \sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} (w_e - w_f)^{2.}
\end{aligned}$$

We can conclude in this case like in previous case for M, that

$$\lambda \frac{1}{8} \sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} (w_e - w_f)^2$$

will be constant and it will not affect the optimization problem, so we can ignore this part. At the end, for the second term we obtain

$$\lambda \cdot \frac{1}{8} \sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} (w_e - w_f)^2 z_e z_f + \lambda \cdot \frac{1}{4} \sum_{e \in E} z_e \sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} (w_e - w_f)^2. \quad (3.4)$$

Ignoring the constants, by summing up (3.3) and (3.4), we obtain that the new objective has the following form:

$$\frac{1}{2} \left(\sum_{e \in E} \left(w_e + \lambda \sum_{\substack{f \in E \\ e,f \text{ segment}-\text{adjacent}}} \frac{(w_e - w_f)^2}{2} \right) z_e + \lambda \sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} \frac{\left(w_e - w_f\right)^2}{4} z_e z_f \right).$$

Transformation of the constraints

We now express the constraints in (3.1) in terms of the variables z_e, $e \in E$. To this end, recall that $n_k = |V_k|$, $k = 1, ..., K$. For the first constraint, we have $\sum_{u \in V_1} \frac{z_{su}+1}{2} = 1$, which is equivalent to

$$\sum_{u \in V_1} z_{su} = 2 - \sum_{V_1} 1 = 2 - n_1.$$

Similarly, for the second and the third constraint, we, respectively, obtain

$$\sum_{u \in V_k} z_{ud} = 2 - \sum_{V_k} 1 = 2 - n_k,$$

and

$$\sum_{u \in V_{k+1}} \frac{z_{vu}+1}{2} - \sum_{u \in V_{k-1}} \frac{z_{uv}+1}{2} = 0,$$

implying

$$\sum_{u \in V_{k+1}} z_{vu} - \sum_{u \in V_{k-1}} z_{uv} = \sum_{u \in V_{k-1}} 1 - \sum_{u \in V_{k+1}} 1 = n_{k-1} - n_{k+1}$$

for each $v \in V_k$, $k = 1, ..., K-1$.

3.2.2 SDP relaxation

Summarizing, the $(0,1)$- QP (3.1) is transformed into the following $(-1,1)$- QP.

$$\begin{array}{ll} \underset{z}{\text{minimize}} & \sum_{e \in E} \left(w_e + \lambda \sum_{\substack{e,f \in E \\ e,f \text{ segment-adjacent}}} \frac{(w_e - w_f)^2}{2} \right) z_e + \lambda \sum_{\substack{e,f \in E \\ e,f \text{ segment-adjacent}}} \frac{\left(w_e - w_f\right)^2}{4} z_e z_f. \\ \text{subject to} & \sum_{u \in V_1} z_{su} = c_s \\ & \sum_{v \in V_K} z_{vd} = c_d \\ & \sum_{u \in V_{k-1}} z_{vu} - \sum_{u \in V_{k+1}} z_{uv} = c_v, \ v \in V_k, \ k = 1, ..., K \\ & z_e \in \{-1, 1\}, \ e \in E \end{array}, \tag{3.5}$$

where $c_s = 2 - n_1$, $c_d = 2 - n_K$, and $c_v = n_{k-1} - n_{k+1}$, for any $v \in V_k$, for $k = 1, ..., K$. We can notice that the new objective consists of two parts. The first part is linear, with weights equal to:

$$w_e + \lambda \sum_{\substack{e,f \in E \\ e,f \text{ segment-adjacent}}} \frac{(w_e - w_f)^2}{2}$$

and second part is quadratic[2] with weights equal to:

$$\lambda \sum_{\substack{e,f \in E \\ e,f \text{ segment-adjacent}}} \frac{\left(w_e - w_f\right)^2}{4}.$$

This is exactly what we wanted, because now we can define matrices which we need in order to make SDP formulation and then apply SDP relaxation. We know that quadratic form is defined as $z^T Q z + 2z^T c$. For matrix Q, we use integer indices (for edges), while in the objective we are still at the "edge-induced" notation $e, f \in E$. We should therefore bridge out these two notations. We introduce ordering of edges, e.g., from left to right, by segments, and from top to bottom, inside segments. To be precise, first we order collection of edges by segments - it proceeds west-east by segments, and north-south across segment pairs (edges that connect first and second segment, then edges that connect second and third segment and so on). After that we sort edges in every collection of edges - by nodes. First we put edges which are starting from first node in segment, then edges which are starting from second node in segment and so on (as we mention from top to bottom inside segments).

[2]We define summation over an empty set as zero. Just to be clear that some products $x_e x_f$ have zero weight (if not adjacent, then we don't need to compare them - at least in this formulation, if we wanted Full as opposed to Seq approach, we would compare all to all edges along a path)

A specific example of how Q is created is provided in Section "Illustration Graph Example". That leads us to formulate matrix Q as:

$$Q_{ij} = \begin{cases} 0, & \text{if } i = j \\ \lambda \frac{\left(w_i - w_j\right)^2}{8} & \text{if } i \text{ and } j \text{ are segment} - \text{adjacent} \\ 0, & \text{otherwise} \end{cases}, \tag{3.6}$$

Also we define c as

$$c = \begin{bmatrix} \frac{w_1}{2} + \lambda \sum_{\substack{1,f \text{ segment}-\text{adjacent}}}^{1,f \in E} \frac{(w_1 - w_f)^2}{4} \\ \vdots \\ \frac{w_M}{2} + \lambda \sum_{\substack{M,f \text{ segment}-\text{adjacent}}}^{M,f \in E} \frac{(w_M - w_f)^2}{4} \end{bmatrix}.$$

If vector c were 0, then we could conveniently represent the quadratic form as $z^\top Qz = trace(QZ)$, where $Z = zz^\top$ is a positive semidefinite matrix, exactly as in the objective of 3.1. When c is not 0, a similar transformation can be achieved, just with an enlarged vector z. Specifically, our goal is to define matrix W such that equality $z^\top Qz + 2z^\top c = \tilde{z}^\top W \tilde{z}$ holds. For that purpose we introduce $(M+1) \times (M+1)$ matrix W from the edge weights, as follows:

$$W = \begin{bmatrix} Q_{M \times M} & c \\ c^\top & 0 \end{bmatrix}, \tag{3.7}$$

$$W_{ky} = \begin{cases} 0, & \text{if } k = y \\ \frac{w_k}{2} + \lambda \sum_{k,j \text{ segment}-\text{adjacent}}^{j} \frac{(w_k - w_j)^2}{4}, & \text{if } y = M+1, k = 1,..M, j = 1,..M \\ \frac{w_y}{2} + \lambda \sum_{y,j \text{ segment}-\text{adjacent}}^{j} \frac{(w_y - w_j)^2}{4}, & \text{if } k = M+1, y = 1,..M, j = 1,..M \\ \lambda \frac{(w_i - w_j)^2}{8}, & \text{if } i \text{ and } j \text{ are segment} - \text{adjacent}, i = 1,..x-1, j = 1,..y-1 \\ 0, & \text{otherwise} \end{cases}, \tag{3.8}$$

where we assume that the edge weights w_i inherit the indexing of the edge variables.

Then, it can be shown that the objective in (3.5) equals $\sum_{i,j=1}^{M} W_{ij} Z_{ij} = \text{tr}(W^\top Z)$, where tr denotes the trace operator [29], and we now introduce the $(M+1) \times (M+1)$ matrix $Z = \tilde{z}\tilde{z}^\top$, where $\tilde{z} = \left(z^\top, 1\right)^\top$, and the vector $z = [z_1, z_2, ...z_M]$ is formed by arranging the edges $e \in E$ in a certain (arbitrarily defined) order.

The rank of a matrix is the dimension of the vector space generated by its columns [2], and actually that is identical to the dimension of the space spanned by its rows [20]. In different words, rank of matrix is equal to the maximal number of linearly independent columns (rows) of that matrix. It is easy to see that each column of matrix Z is obtained by multiplying the same vector $\tilde{z}$ by a different number (more specifically, by the corresponding entry of $\tilde{z}$). It follows that the number of linearly independent columns in Z is 1, hence Z is rank 1.

Our matrix Z is of the following form:

$$Z = \begin{bmatrix} zz^\top & z \\ z^\top & 1 \end{bmatrix} \tag{3.9}$$

Note that $Z_{ij} = z_i z_j$, for any $1 \leq i, j \leq M$. Since $z_i \in \{-1, 1\}$, we have that $Z_{ii} = 1$ for each i.

We now turn to the constraints in (3.5). For each node $v \in V_k$, introduce the set I_v^- that collects indices i of edges connected to node v to the left; similarly, I_v^+ collects indices of edges connected to v to the right. For each $v \in V$ (including s and d), denote by a_v the vector that defines the corresponding constraint in (3.5) by $a_v^\top z = c_v$. Each a_v has the same length as vector z and each a_v has has -1 on positions corresponding to the incoming edges,$e \in I_v^-$, and +1 on positions corresponding to the outgoing edges, $e \in I_v^+$. All other elements (edges which have no connection with v) in a_v have values equal to 0.

For general case a_s will have first n_1 elements of vector filled with ones and rest $M - n_1$ will be filled with zeros. That is, the generic structure of a_s is:

$$a_s = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

And then we have that the inner product of a_s and z is :

$$a_s^\top z = \begin{bmatrix} 1 & \cdots & 1 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{M-1} \\ z_M \end{bmatrix} = 1 + (-1)(n_1 - 1) = 1 - n_1 + 1 = 2 - n_1.$$

We note that the right hand side has a very intuitive meaning because $1 + (-1)(n_1 - 1)$ shows that we will choose exactly one edge (corresponding to the single +1 term) from collection of edges which connect two adjacent segments (in this case segment with source node and first segment). Because of that we will have $n_1 - 1$ number (corresponding to the -1 values in z) of not selected edges. A similar conclusion holds for other adjacent segments.

By a similar reasoning as with the objective function, it can be easily shown that the constraints can be encoded as

$$\mathrm{tr}(A_v Z) = c_v,$$

for each $v \in V$, where A_v is a symmetric matrix of the form

$$A_v = \begin{bmatrix} 0_{M \times M} & a_v/2 \\ a_v/2^\top & 0 \end{bmatrix}.$$

Summarizing, problem (3.5) can be represented as the following semidefinite program with rank 1 constraint:

$$\begin{array}{ll} \underset{Z}{\text{minimize}} & \operatorname{tr}(WZ) \\ \text{subject to} & \operatorname{tr}(A_v Z) = c_v,\ v \in V \\ & Z_{ii} = 1,\ i = 1, ..., M+1 \\ & Z \succeq 0 \\ & \operatorname{rank}(Z) = 1 \end{array} \quad . \tag{3.10}$$

By dropping the rank 1 constraint, we obtain a semidefinite program (SDP), which can be efficiently solved using numerical algorithms for SDP. The solution vector z can then be recovered by appropriate rounding procedures.

3.2.3 Rounding Procedures

After applying semidefinite relaxation we need to obtain the Z matrix. If the solution to the SDP relaxation (3.10) happens to be a rank one matrix $Z = yy^\top$, then, thanks to the constrains $Z_{ii} = 1$, we will have that y_i is either 1 or -1. Hence, vector y obtained through the optimal Z matrix belongs to the feasibility set of the SPBC problem. Since y is the solution to the relaxed problem it must therefore be the optimal solution of the original SPBC problem that we are trying to solve. This will however not always be the case. In general, the solution of the SDP relaxation will not be a rank one matrix, and we therefore need some rounding procedure to get back to the original, integer domain. We need to perform at least one of the rounding procedures to read the solution vector z. With rounding procedures, there is no right or wrong way to recover z vector. There are many methods by which the vector z can be recovered. Some of these are described in the paper [24]. Below is an explanation of the rounding procedure that we used. After the SDP algorithm, we get a matrix Z. In case that Z is matrix with rank equal to 1, Z will look like the following:

$$Z = \begin{bmatrix} z_1 z_1 & z_1 z_2 & \cdots & z_1 z_M & z_1 \\ z_2 z_1 & z_2 z_2 & \cdots & z_2 z_M & z_2 \\ \vdots & \vdots & & \vdots & \vdots \\ z_M z_1 & z_M z_2 & \cdots & z_M z_M & z_M \\ z_1 & z_2 & \cdots & z_M & 1 \end{bmatrix}$$

In order to recover the vector z, we look at the last row (without the last element being equal to 1). Note that it is also equivalent to look at the last column without the last element.

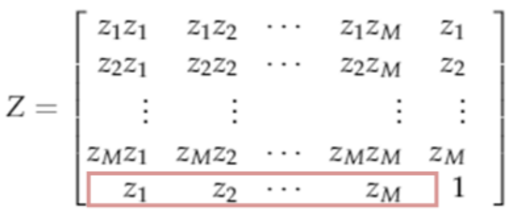


FIGURE 3.2: z vector for rounding procedures

Two rounding procedures were used in our work. In both rounding procedures the vector z is divided into as many parts as we have segments in an event + 1. This is because we have two artificial segments so we will have $K+1$ collections of edges (one collection of edges represents edges between two adjacent segments). The length of each collection of edges is the number which is equal to multiplication of number of elements in adjacent segments which are connected with exactly these edges. We introduce length of size of collection of edges like: M_1 length of first collection of edges, M_2 length of second collection of edges, M_3 length of third collection of edges and so on.

The following figure illustrates a collection of edges.

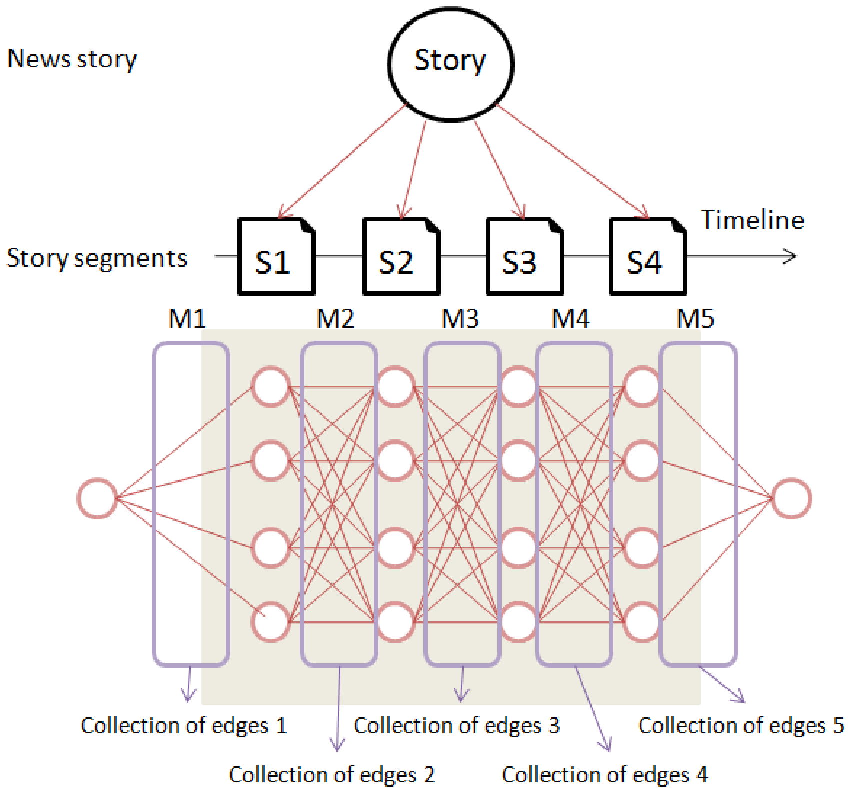


FIGURE 3.3: Collections of edges

$$z = [\overbrace{z_1, \ldots z_{M_1}}^{lengthofcollectionofedges1}, \overbrace{z_{M_1+1}, \ldots z_{M_2}}^{lengthofcollectionofedges2}, \ldots, \overbrace{z_{M_K+1}, \ldots z_{M_{K+1}}}^{lengthofcollectionofedgesK+1}]$$

Rounding Procedure 1

In the first rounding procedure, we first consider the part of the vector corresponding to the first collection of edges and select the maximum element. With this we have chosen the first edge, actually the first element in our path. Therefore, two images were selected (the start and end node of that edge). Next we look at the elements belonging to the second edge collection and select the maximum element. Then we repeat the process for all edge collections. That way we choose the path in our graph. The final output of the procedure is a sequence of edges, each connecting two consecutive segments, starting from the first until the last one. We note that this sequence might or might not be a path in the segments graph.

Rounding Procedure 2

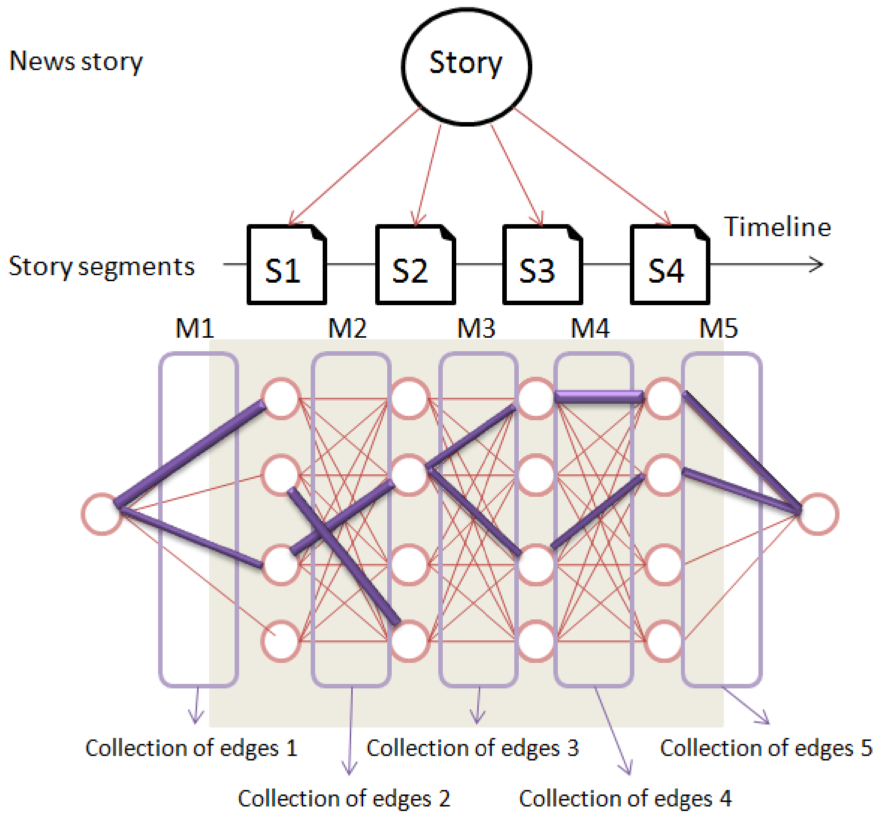


FIGURE 3.4: Example of chosen edges with Rounding Procedure 2

In the second rounding procedure, we again look at the first edge collection but this time we select two maximum elements. We then select the two maximum elements in the second edge collection and repeat this until we also select the two maximum elements from the last edge collection. We now have two edges between each adjacent pair of segments, which means we can have more than one path in the graph. We form all possible paths from the edges we have chosen. The next step is to insert the weights of the edges into the cost function (for each path in the graph we formed - separately) and finally select the path that gives the minimum cost function. The cost function is shown below.

$$\sum_{e \in E} w_e x_e + \lambda \sum_{\substack{e,f \in E \\ e,f \text{ segment-adjacent}}} \frac{(w_e - w_f)^2}{2} x_e x_f .$$

The rounding procedure 2 is more complex than the rounding procedure 1 and therefore takes longer to complete. However, in most cases where procedure 1 cannot find a feasible solution, there is a high possibility that rounding procedure 2 will find a feasible solution.

3.3 Illustration Graph Example

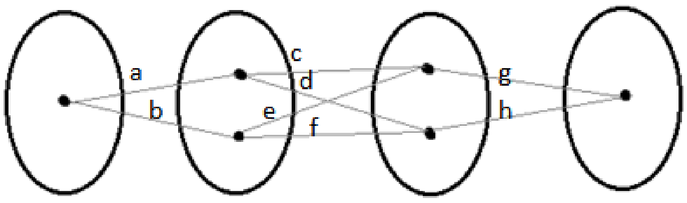


FIGURE 3.5: Graph example

In this section, we illustrate all the quantities used by our method with a simple example of two segments, shown in Figure 3.5. Applying the ordering defined in Section SDP relaxation. Before defining corresponding matrices and vectors, we set order of edges. For the given graph, our sorted weights of edges will look like:

$$\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

In our case, Q is :

$$Q = \begin{bmatrix}
0 & 0 & \lambda\frac{(c-a)^2}{8} & \lambda\frac{(d-a)^2}{8} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \lambda\frac{(e-b)^2}{8} & \lambda\frac{(f-b)^2}{8} & 0 & 0 \\
\lambda\frac{(c-a)^2}{8} & 0 & 0 & 0 & 0 & 0 & \lambda\frac{(g-c)^2}{8} & 0 \\
\lambda\frac{(d-a)^2}{8} & 0 & 0 & 0 & 0 & 0 & 0 & \lambda\frac{(h-d)^2}{8} \\
0 & \lambda\frac{(e-b)^2}{8} & 0 & 0 & 0 & 0 & \lambda\frac{(g-e)^2}{8} & 0 \\
0 & \lambda\frac{(f-b)^2}{8} & 0 & 0 & 0 & 0 & 0 & \lambda\frac{(h-f)^2}{8} \\
0 & 0 & \lambda\frac{(g-c)^2}{8} & 0 & \lambda\frac{(g-e)^2}{8} & 0 & 0 & 0 \\
0 & 0 & 0 & \lambda\frac{(h-d)^2}{8} & 0 & \lambda\frac{(h-f)^2}{8} & 0 & 0
\end{bmatrix}$$

Vector c is equal to :

$$c = \begin{bmatrix} \frac{a}{2} + \lambda\frac{(c-a)^2}{4} + \lambda\frac{(d-a)^2}{4} \\ \frac{b}{2} + \lambda\frac{(f-b)^2}{4} + \lambda\frac{(g-b)^2}{4} \\ \frac{c}{2} + \lambda\frac{(c-a)^2}{4} + \lambda\frac{(g-c)^2}{4} \\ \frac{d}{2} + \lambda\frac{(d-a)^2}{4} + \lambda\frac{(d-a)^2}{4} \\ \frac{e}{2} + \lambda\frac{(g-e)^2}{4} + \lambda\frac{(e-b)^2}{4} \\ \frac{f}{2} + \lambda\frac{(f-b)^2}{4} + \lambda\frac{(h-f)^2}{4} \\ \frac{g}{2} + \lambda\frac{(g-e)^2}{4} + \lambda\frac{(g-c)^2}{4} \\ \frac{h}{2} + \lambda\frac{(h-d)^2}{4} + \lambda\frac{(h-f)^2}{4} \end{bmatrix}$$

The complete matrix W is shown on equation 3.11.

$$
W = \left[\begin{array}{cccccccc|c}
0 & 0 & \lambda\frac{(c-a)^2}{8} & \lambda\frac{(d-a)^2}{8} & 0 & 0 & 0 & 0 & \frac{a}{2} + \lambda\frac{(c-a)^2}{4} + \lambda\frac{(d-a)^2}{4} \\
0 & 0 & 0 & 0 & \lambda\frac{(e-b)^2}{8} & \lambda\frac{(f-b)^2}{8} & 0 & 0 & \frac{b}{2} + \lambda\frac{(f-b)^2}{4} + \lambda\frac{(g-b)^2}{4} \\
\lambda\frac{(c-a)^2}{8} & 0 & 0 & 0 & 0 & 0 & \lambda\frac{(g-c)^2}{8} & 0 & \frac{c}{2} + \lambda\frac{(c-a)^2}{4} + \lambda\frac{(g-c)^2}{4} \\
\lambda\frac{(d-a)^2}{8} & 0 & 0 & 0 & 0 & 0 & 0 & \lambda\frac{(h-d)^2}{8} & \frac{d}{2} + \lambda\frac{(d-a)^2}{4} + \lambda\frac{(d-a)^2}{4} \\
0 & \lambda\frac{(e-b)^2}{8} & 0 & 0 & 0 & 0 & \lambda\frac{(g-e)^2}{8} & 0 & \frac{e}{2} + \lambda\frac{(g-e)^2}{4} + \lambda\frac{(e-b)^2}{4} \\
0 & \lambda\frac{(f-b)^2}{8} & 0 & 0 & 0 & 0 & 0 & \lambda\frac{(h-f)^2}{8} & \frac{f}{2} + \lambda\frac{(f-b)^2}{4} + \lambda\frac{(h-f)^2}{4} \\
0 & 0 & \lambda\frac{(g-c)^2}{8} & 0 & \lambda\frac{(g-e)^2}{8} & 0 & 0 & 0 & \frac{g}{2} + \lambda\frac{(g-e)^2}{4} + \lambda\frac{(g-c)^2}{4} \\
0 & 0 & 0 & \lambda\frac{(h-d)^2}{8} & 0 & \lambda\frac{(h-f)^2}{8} & 0 & 0 & \frac{h}{2} + \lambda\frac{(h-d)^2}{4} + \lambda\frac{(h-f)^2}{4} \\
\hline
\frac{a}{2} + \lambda\frac{(c-a)^2}{4} + \lambda\frac{(d-a)^2}{4} & \frac{b}{2} + \lambda\frac{(f-b)^2}{4} + \lambda\frac{(g-b)^2}{4} & \frac{c}{2} + \lambda\frac{(c-a)^2}{4} + \lambda\frac{(g-c)^2}{4} & \frac{d}{2} + \lambda\frac{(d-a)^2}{4} + \lambda\frac{(d-a)^2}{4} & \frac{e}{2} + \lambda\frac{(g-e)^2}{4} + \lambda\frac{(e-b)^2}{4} & \frac{f}{2} + \lambda\frac{(f-b)^2}{4} + \lambda\frac{(h-f)^2}{4} & \frac{g}{2} + \lambda\frac{(g-e)^2}{4} + \lambda\frac{(g-c)^2}{4} & \frac{h}{2} + \lambda\frac{(h-d)^2}{4} + \lambda\frac{(h-f)^2}{4} & 0
\end{array}\right] \tag{3.11}
$$

If we choose, for example, the path $a-c-g$, the corresponding vector z then equals:

$$z=\begin{bmatrix}1\\-1\\1\\-1\\-1\\-1\\1\\-1\end{bmatrix}.$$

We define matrix Z as $(M+1)\times(M+1)$ matrix $Z=\tilde{z}\tilde{z}^\top$, where $\tilde{z}=\left(z^\top,1\right)^\top$. So we have that

$$Z=\begin{bmatrix}zz^\top & z\\ z^\top & 1\end{bmatrix}.$$

In our case, Z is :

$$Z=\begin{bmatrix}1&-1&1&-1&-1&-1&1&-1&1\\-1&1&-1&1&1&1&-1&1&-1\\1&-1&1&-1&-1&-1&1&-1&1\\-1&1&-1&1&1&1&-1&1&-1\\-1&1&-1&1&1&1&-1&1&-1\\-1&1&-1&1&1&1&-1&1&-1\\1&-1&1&-1&-1&-1&1&-1&1\\-1&1&-1&1&1&1&-1&1&-1\\1&-1&1&-1&-1&-1&1&-1&1\end{bmatrix}.$$

From this example we can see that matrix that matrix Z satisfies $Z_{ii}=1$, for each $i=1,...,M+1$.

Now we will introduce corresponding vectors $a_v, v\in V$. In our example, we will have for the start node:

$$a_s=\begin{bmatrix}1\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}.$$

And then we calculate the product of a_s and z

$$a_s^\top z=\begin{bmatrix}1&1&0&0&0&0&0&0&0\end{bmatrix}\begin{bmatrix}1\\-1\\1\\-1\\-1\\-1\\1\\-1\end{bmatrix}=1+(-1)(2-1)=1-2+1=2-2=0.$$

In our case, A_s is :

$$A_s = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

For nodes between a_s and a_d, we assign value -1 to edges which are in I_v^- and we assign value $+1$ to edges which are in I_v^+. For example, for vertex v which is connected with edges a, c, d corresponding vector a_v and matrix A_v will look like :

$$a_v = \begin{bmatrix} -1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

And then we calculate the product of a_d and z to check if constraint condition is satisfied.

$$a_v^\top z = \begin{bmatrix} -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = (-1)1 + 1 \times 1 + (-1)1 = -1 + 1 - 1 = -1$$

This is exactly equal to constraint condition which says that this product should be equal to difference between number of nodes in previous segment and number of nodes in next segment. In our case it is $1 - 2 = -1$.

So for this vertex v, A_v is :

$$A_v = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

When we want to run our algorithm, we need to give as input matrices A, a constraints vector, and a matrix W. As a solution, we get a matrix Z, which, as we explained in this case, is:

$$Z = \begin{bmatrix} 1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \end{bmatrix}.$$

After matrix Z is obtained, rounding procedures are applied. Our vector z we can read from matrix Z (last column or row without last element which is equal to 1). So, $z = [1, -1, 1, -1, -1, -1, 1, -1]$.

In the Rounding Procedure 1 we look at the length of collection of edges and then choose maximum element from each collection. First collection of edges is formed of edges a, b. We look at length of that collection (in this case it is equal to 2) in vector $z = [1, -1, *, *, *, *, *, *]$ and we choose max element which is 1. We choose first element from z. Then we consider second collection of edges which is equal to c, d, e, f. Now we choose max element from $z = [*, *, 1, -1, -1, -1, *, *]$, which is third element from z. At the end we choose edge from collection of edges g, h which is equal to max element in $z = [*, *, *, *, *, *, 1, -1]$.

To sum up, we choose edges $z = [1, *, 1, *, *, *, 1, *]$. If we recall, edges are sorted in this order $[a, b, c, d, e, f, g, h]$. So, we conclude that chosen path is $a - c - g$.

Rounding procedure 2 is similar to rounding procedure 1 but selects two max elements from each collection of edges, after which all paths from the selected edges are formed. The value of cost function for one path is calculated by inserting the values of weights of edges into the definition of the cost function. After all the paths that the selected edges can form are formed, the path which have the lowest value of the cost function is chosen.

3.4 Implementation

All implementations are done in Python programming language [9]. In our case, for the input data we have the adjacency matrix, number of segments and lengths of segments. With this information and using the NetworkX [4] library, we form a graph. In order to arrive at the SDP formulation, we first need to define the necessary matrices and constraints. NumPy library [5] was used to construct the matrices and vectors. The diagram in Fig 3.6 presents the skeleton of our implementation. In the first step, we construct the graph, including the edge weights. The weights are defined through the adjacency matrix for each image pair, which is predefined and not part of our consideration. It is important to mention that in adjacency matrix we have an order by segments and then by nodes in segments. After making the graph, we define lambda (the value for lambda may change at run of the algorithm). The next step is to define the matrix Q and the vector c. We define vector c from two parts because the first part refers only to the weights of the edges while the second part is shown in the previous section. When we define matrix Q and vector c, we can easily define matrix W. Then we define edge collections, a_v, A_v for all nodes including source and destination and after that we define constraints vector.

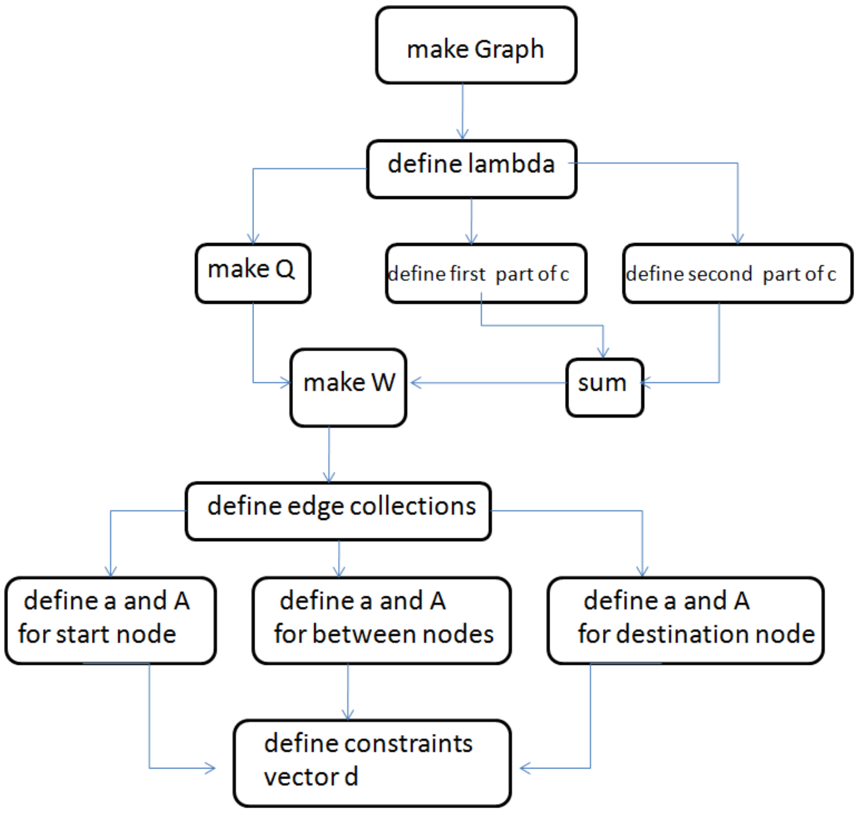

make Graph
define lambda
make Q
define first part of c
define second part of c
make W
sum
define edge collections
define a and A for start node
define a and A for between nodes
define a and A for destination node
define constraints vector d

FIGURE 3.6: Steps for SDP form implementation 1

The SDP problem formulation is defined with the CVXPY [1] library while the SDP problem is solved with the CVXOPT [22] library. Rounding procedures are implemented with NumPy [5] library. Images for stories are plotted with PIL [10], skimage [32] and matplotlib [17] libraries. When we define the matrices W, A and the constraints vector d, we can apply the form SDP and then solve the SDP. As a result, we obtain the matrix Z from which we observe the vector z (the last column or row of Z without the last element equal to 1). Then we apply rounding procedures and get the selected path.

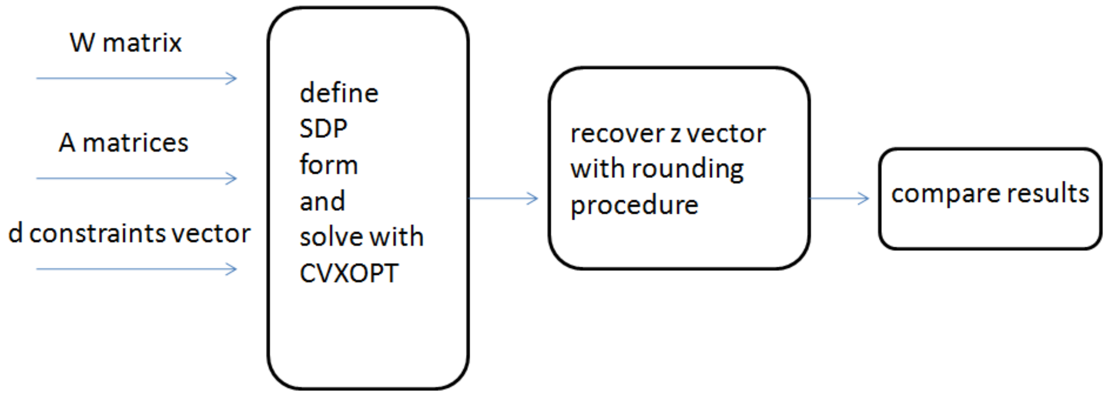

W matrix
A matrices
d constraints vector
define SDP form and solve with CVXOPT
recover z vector with rounding procedure
compare results

FIGURE 3.7: Steps for SDP form implementation 2

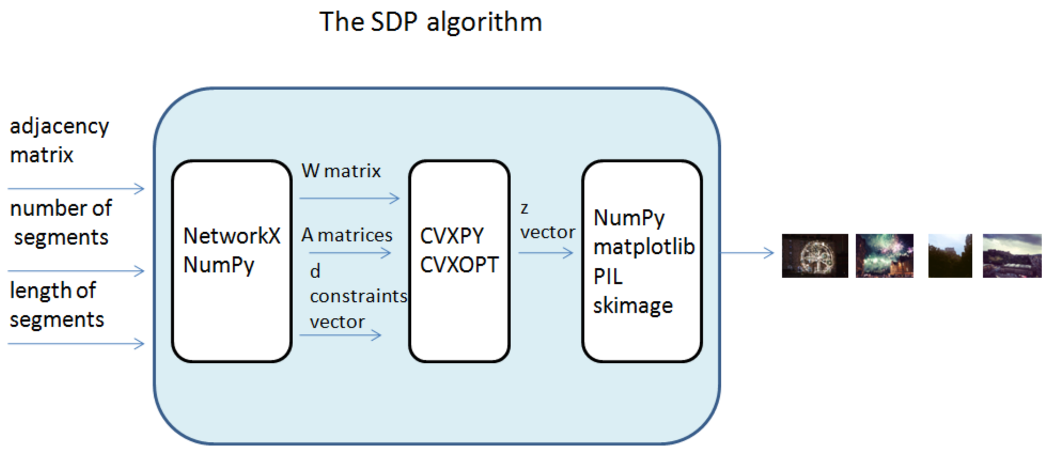


FIGURE 3.8: The SDP algorithm implementation

Chapter 4

Experiments

Three types of experiments were performed. The first experiment is about synthetic data. The goal was to establish how our cost function behaves when one of the edge weights of the selected path in the graph changes. The second experiment was performed on a specific type of data, namely on images containing different families of flowers. The flowers dataset is downloaded from [7]. In this example, we had 4 types of different graph structures and 10 graphs were created for each of these types. The third type of experiment relates to data of stories from The Edinburgh Festival 2016.

4.1 Synthetic Data

In this experiment, we tested how the function would behave under different conditions. We have observed a vector of length 5 which is an illustration of the path in the graph. This graph is hypothetically composed of 4 segments and the source and destination nodes are added. We tested how the function would behave if one edge changes and the other two edges connecting adjacent segments were fixed. So in this path, we fix weights of two edges, and vary third one in order to see how the cost function will behave. Also we tested how the function would behave if the smoothing parameter changes. The red line in Fig4.2 (and similarly for any other similar appearances) represents values of the cost function, the green represents the value of term 1 from cost function and the blue line represents term 2 from cost function as shown in Fig4.1. The following equation represents term 1 and term 2 in our cost function.

$$\overbrace{\sum_{e \in E} w_e x_e}^{Term1} + \lambda \overbrace{\sum_{\substack{e,f \in E \\ e,f \text{ segment}-\text{adjacent}}} \frac{\left(w_e - w_f\right)^2}{2} x_e x_f}^{Term2} \tag{4.1}$$

4.1.1 Unbalanced edge weights

In the following figures we show the results obtained by varying the weight of each edge. In the first example we vary the weight of first edge, in the second example we vary the weight of the second edge and in the third example we vary the weight of the third edge. In each of these three examples, we fix the smoothing parameter λ to be equal to 1, and also in each of the examples we fix weights of the other two edges in path. We have 5 cases of fixing two edges in path. They are respectively equal to $0.01, 0.25, 0.5, 0.75$ and 0.99.

We can conclude from the figures that varying the last or first edge weight gives the similar results, but varying middle one is different. In order to see what happens

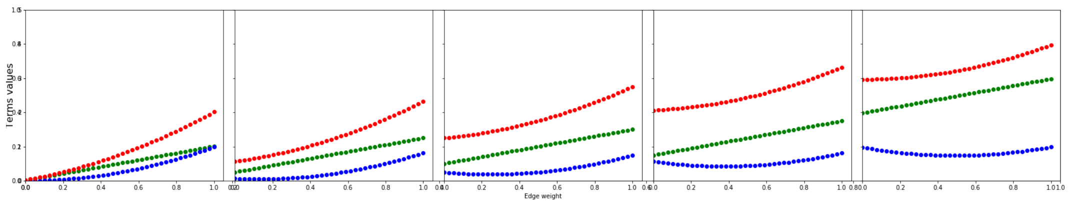


FIGURE 4.1: First edge changes

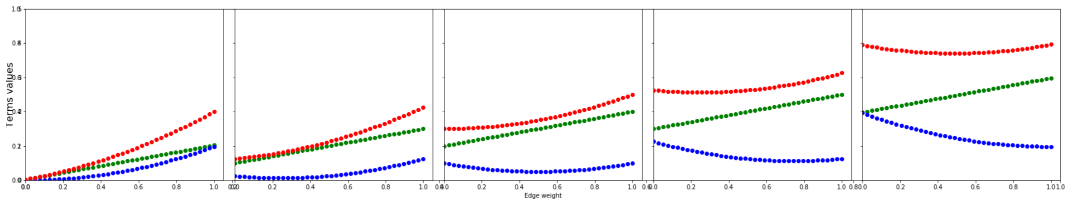


FIGURE 4.2: Second edge changes

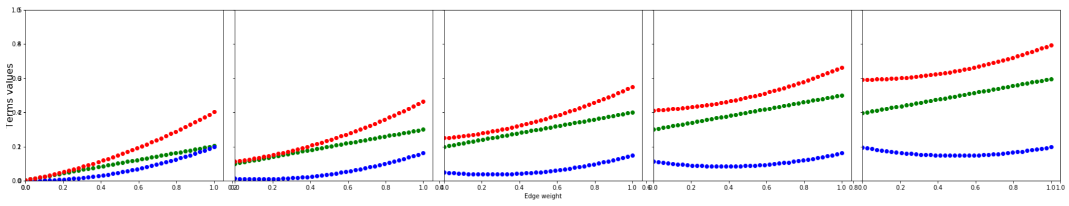


FIGURE 4.3: Third edge changes

when we change the values of λ, we consider a few cases ($\lambda = 1, \lambda = 3, \lambda = 5, \lambda = 8$) which will be discussed in the next section.

4.1.2 Regularization

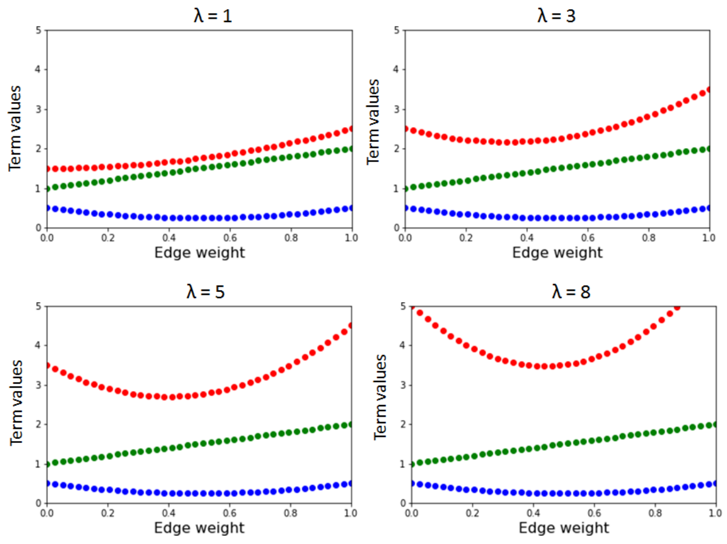


FIGURE 4.4: λ value changes

In the Fig.4.4 we can see how the function behaves if we change the smoothing parameter - the parameter that is responsible for regularization. In the Fig.4.4 we can see an example where the first and the third edge are fixed to value 0.5 and the second edge is the edge whose weight value varies. As in the previous figures, the red line represents the cost function, the green line represents the value of term 1 and the blue line represents the value of term 2. The higher the λ, the greater the possibility that we can find a path that will be different than the shortest path.

4.1.3 Synthetic Graph

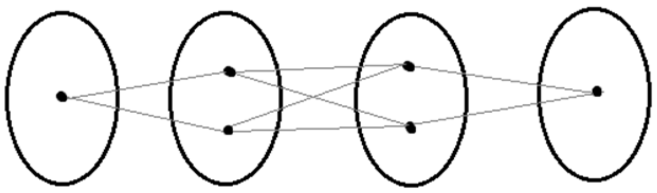


FIGURE 4.5: Synthetic graph structure

After testing how the function would behave with different values of λ and with different values for edge weights, we implemented how to create matrices, set up an optimization problem and solve it and then apply rounding procedures. We made a graph consisting of two segments with two nodes in each segment where the values for edge weights were chosen by a random function from the interval (0,1). Except the nodes in the segments, the source and destination nodes are also included. The edges with one of the source or destination nodes have a weight equal to 0. The algorithm was run multiple times on the graph of this structure but nothing of what interests us could be deduced from the results that were numerically only. We know that for λ value equal to 0 the same path would be obtained as and the shortest path in the graph (which is logical because the second sum in our equation is equal to 0 and only the sum of the weight of the edges in the path remains), and for λ values greater than 1, in most cases the solution is different from the shortest path in the graph. The higher the value for λ, the greater the probability that the solution obtained by the algorithm and the shortest path in the graph will differ. Since we couldn't deduce anything from the numerical results about the visual "smoothness" effect, we decided to do an experiment on dataset which contains images.

4.2 Flowers Dataset

In order to examine how the framework works, we have selected a dataset containing 1360 images of 17 species of different flowers [7]. The 17 flower types are divided into 8 flower families. To define edge weights for flower graphs in experiments we use adjacency matrix which is made with ResNet-50 [14] convolutional neural network, pre-trained on the ImageNet [3] dataset for object detection. The softmax layer (classification layer) is removed and feature maps are average pooled. Each image is then given as input to the network and the activations of the last convolutional layers were extracted, average pooled, resulting in vectors of dimension 2048. While a challenging problem in itself, obtaining the adjacency matrix was not part of our

work, i.e. we used a predefined adjacency matrix.
We chose this dataset because we know that we can have complete control over the results, specifically we can define the metric nicely. We decided to evaluate the results by considering the "correct" result to be where all the images in the path are images of the same flower or all the images in the path are images from the same flower family.

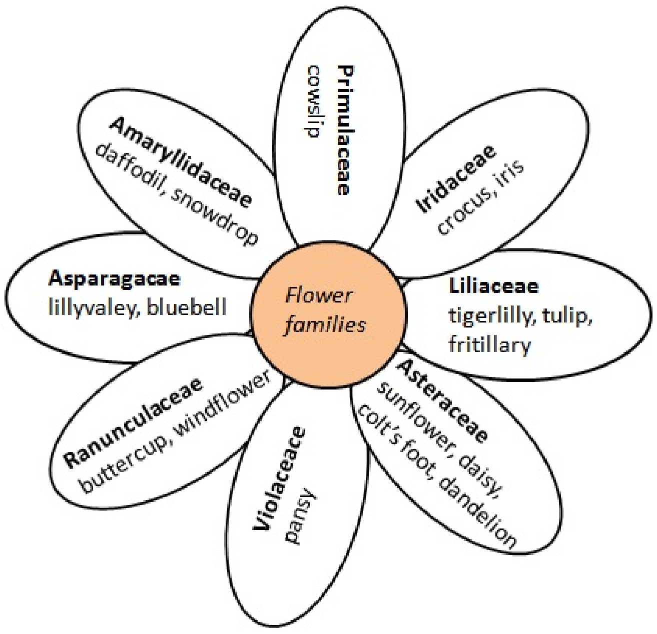


FIGURE 4.6: Flowers families

4.2.1 Experimental setup

To test how the algorithm works, we examined 4 types of graphs where each has 4 segments with 3 images each, but with the images in segments chosen in different ways:

- **Fine grain correct solutions**: In the first type of graphs in each of the segments are mixed families of flowers, but there is one same flower in each of the segments - so we consider that the correct solution is the path that connects those 4 images of the same flower.
- **Coarse grain correct solutions**: In the second type of graphs, in the first and the last segments there is the same flower, while in the middle two segments there is a flower that is from the same flower family as mentioned flower from the first and the last segment. In this case, the path we are looking for will be just that path where all the flowers are from the same family.
- **Multiple correct solutions**: In the third type of graphs, in the first and the last segments there is the same flower, while in the middle two segments there is the same flower but also a flower that is from the same flower family. There are two correct solutions here - one that will find a path of four same flowers, and another one that will find a path of 4 flowers from the same flower family.

- **No correct solution**: In the fourth type of graphs, families of flowers as well as flowers are mixed in segments. In the middle segments (the second and the third) there are no flowers from the same flower specie or from the same family. In this type of graphs, we have two flowers from the same family in the first two segments and in the last two segments are two flowers from the same family but that family is different from the family that appears in the first two segments. In this case we consider that solution is correct if on the chosen path we have first two flowers from one family and third and fourth flowers from another family.

4.2.2 Baselines

- **SDPS1**: This indicates that after we have obtained a solution to the relaxed problem, we apply the rounding procedure 1. (Explained in previous chapter in section "Rounding procedures".)
- **SDPS2**: This indicates that after we have obtained a solution to the relaxed problem, we apply the rounding procedure 2. (Explained in previous chapter in section "Rounding procedures".)
- **SPS**: This means that we apply the shortest path algorithm - find all paths in the graph such that only one edge is selected between two adjacent segments and then chose the path that has the smallest value of sum of edge weights.

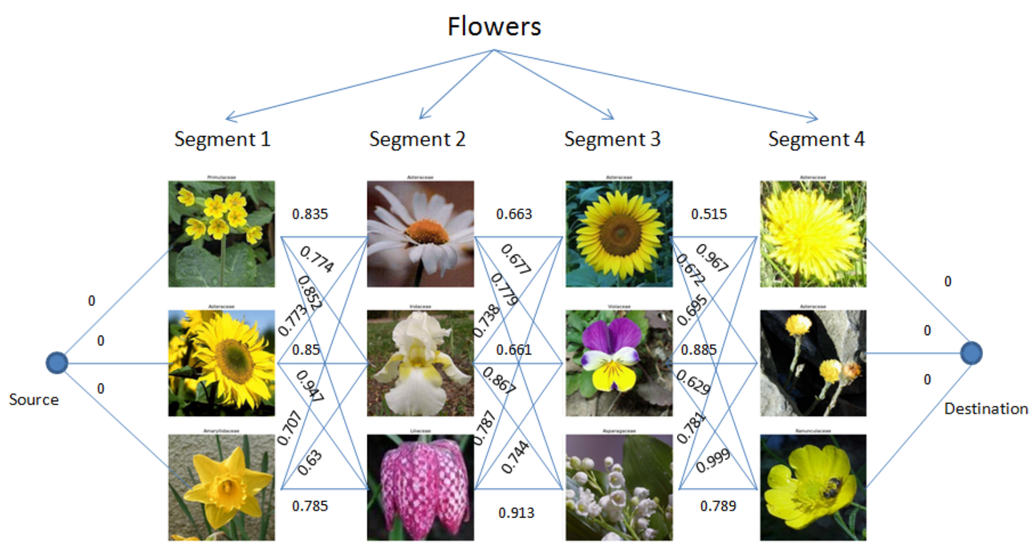
4.2.3 Working example

The Fig.4.7 represents an example of how the flowers graph looks like. This graph was created by randomly choosing the images in each segment. It is intended for illustration purpose only, i.e., it does not belong to the previously described classes of flowers graphs. As described in the previous chapters, the goal is to select one image per each column, so that the resulting image sequence is as coherent and as informative as possible. The SDP algorithm selects the edges and thus determines the path from which we can deduce which images are selected. The figure also shows the edge weights. On the right is a column vector representing the edge weights, sorted as described in the previous chapter (from left to right, see Section Illustration Graph Example) - from source node to segment 1, then the first image from segment 1 and all the edges that merge that image with the images in segment 2, then the second image from the segment 1 and so forth.

From the example of flowers graph in Figure 4.7, we obtain through SDP and SP the paths shown on Figure 4.8; the SP solution path is shown on the top subfigure, and the SDP solution path is shown on the bottom subfigure.

We can conclude that for the SP solution, the cost function function is equal to : $0.63 + 0.738 + 0.515 + 0(\frac{(0.63)^2}{2} + \frac{(0.738-0.63)^2}{2} + \frac{(0.515-0.738)^2}{2} + \frac{(0.515)^2}{2}) = 0.63 + 0.738 + 0.515 = 1.883$.

The solution for the SDP algorithm shown in the figure is obtained with λ equal to 5. Keeping that in mind, for the SDP solution, we can conclude that the cost function is equal to : $0.707 + 0.663 + 0.515 + 5(\frac{(0.707)^2}{2} + \frac{(0.663-0.707)^2}{2} + \frac{(0.515-0.663)^2}{2} + \frac{(0.515)^2}{2}) = 1.885 + 5(0.249 + 0.001 + 0.01 + 0.035) = 1.885 + 1.475 = 3.36$. With the SDP algorithm in this case we got a path where more flowers belong to the same flower family. In the SDP solution 3 flowers are from the same family of flowers and in the SP solution 2 flowers are from the same family of flowers.



```
[0],
[0],
[0],
[0.835],
 [0.774],
[0.852],
[0.773],
 [0.85 ],
 [0.947],
 [0.707],
[0.63 ],
[0.785],
[0.663],
[0.677],
[0.779],
[0.738],
 [0.661],
[0.867],
[0.787],
[0.744],
 [0.913],
[0.515],
 [0.967],
 [0.672],
[0.695],
 [0.885],
 [0.629],
 [0.781],
 [0.999 ],
 [0.789],
[0],
[0],
[0]
```

FIGURE 4.7: Example of flowers graph

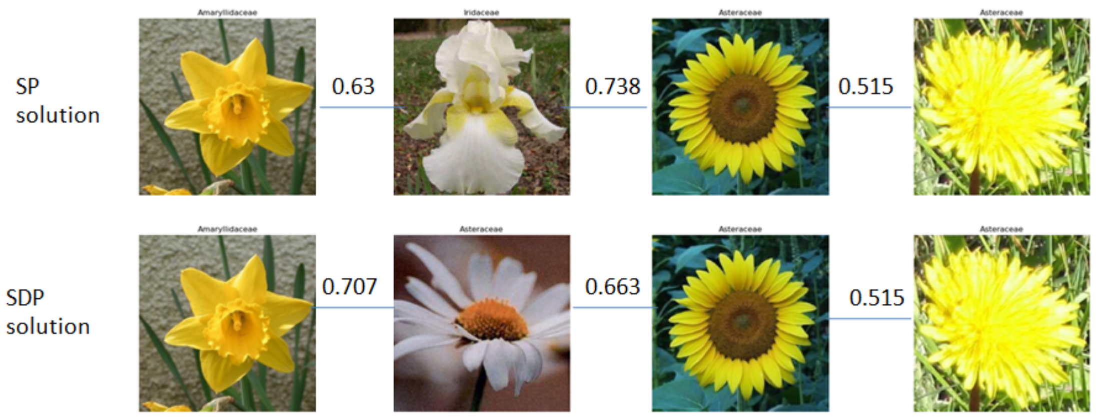


FIGURE 4.8: Solution paths for flowers graph

4.2.4 Results and Discussion

The SDP algorithm was executed on 40 different graphs. For each of 4 graph types we made 10 graphs and for each graph we varied λ values. We tested the algorithm for lambda equal to 0, 0.5, 1, 5, 10.

There are two tables with results for each example. From each experiment we obtain a path of 4 flower images, and with respect to families result is expressed as number of images in most popular[1] family over number of images in path. In the first table with results these values are calculated for each of the experiments. The

[1]The most common family of flower appears on images along the chosen path.

values are then summed up and divided by the number of experiments. The higher the value, the better.

In some cases, the SDP algorithm did not find a solution. Specifically, the SDP algorithm finds the vector z, but with our rounding procedures we do not always get a path in the graph. Problems occur if the numbers in the z vector are too similar. Therefore, for each example there is another table with the results where the obtained values are summed up as in the first table but not divided by the total number of experiments. They are divided by the number of experiments that gave a solution.

Next to each result in the table is written a number in parentheses that represents how many experiments have yielded solutions.

In each of following experiments, we run algorithm with parameter λ equal to $0, 0.5, 1, 5, 10$.

4.2.5 Fine grain correct solutions

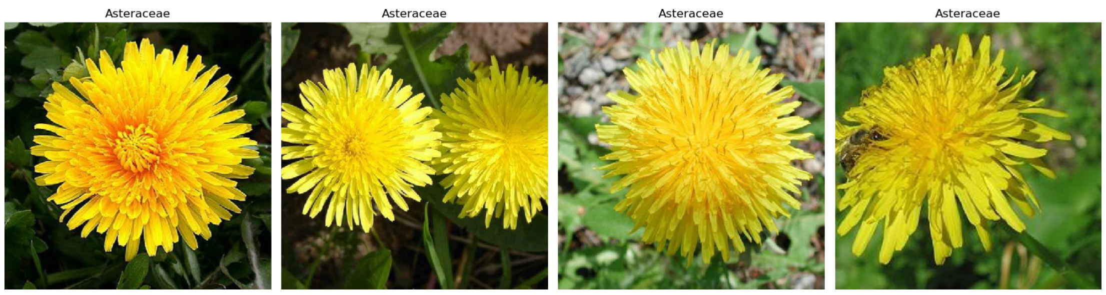


FIGURE 4.9: Fine grain correct solutions - Example of shortest path solution

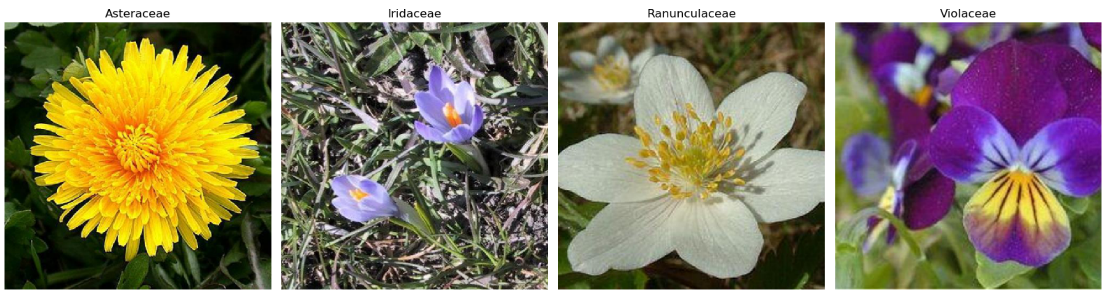


FIGURE 4.10: Fine grain correct solutions - Example of SDP solution with $\lambda = 5$

In this and also in each of the following examples, one of 10 experiments is selected and the shortest path and the path selected by the SDP algorithm for some lambda value are presented. In the solution of the shortest path, we see that the same flowers were selected along the path. The solution of the SDP algorithm shows flowers that make a path that we consider to be incorrect. However, it can be said that the chosen path is visually enjoyable because of the colors that appear gradually with the images. One thing to note is that the algorithm with lambda equals 5 found a path that is different in 3 segments compared to the shortest path solution. In the table with results, we see that the larger the lambda, the smaller the values for SDPS1 and SDPS2, which tells us that the solutions are different from SP (in the sense that in some segments are selected images that the SP algorithm did not select).

TABLE 4.1: Fine grain correct solutions results averaged with number of experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.650
0.5	0.625	0.650	-
1	0.475	0.600	-
5	0.025	0.275	-
10	0.050	0.200	-

TABLE 4.2: Fine grain correct solutions results averaged with number of feasible experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.650
0.5	0.694 (9)	0.650 (10)	-
1	0.678 (8)	0.667 (9)	-
5	0.250 (1)	0.392 (7)	-
10	0.500 (1)	0.400 (5)	-

4.2.6 Coarse grain correct solutions

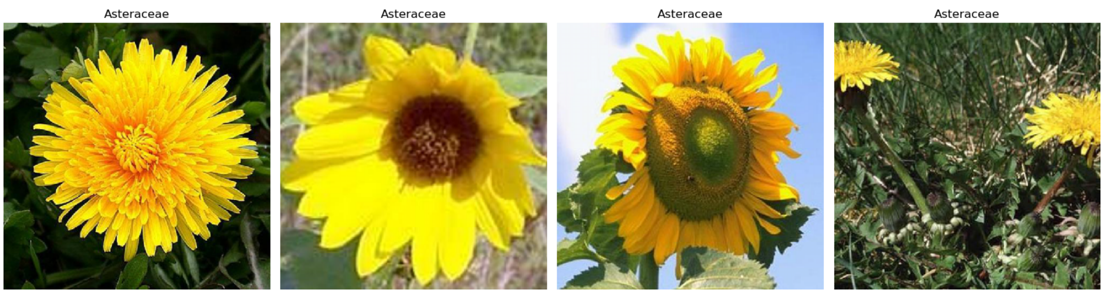


FIGURE 4.11: Coarse grain correct solutions - Example of shortest path solution

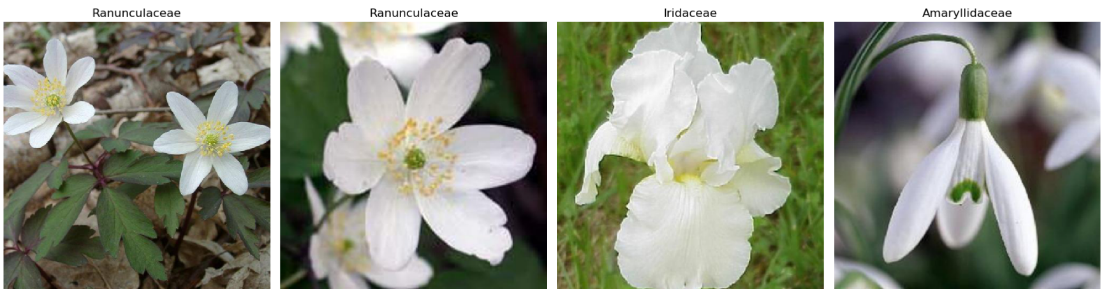


FIGURE 4.12: Coarse grain correct solutions - Example of SDP solution with $\lambda = 1$

As in the previous example, two solutions obtained from one experiment are shown. The solution obtained by the shortest path is also considered to be "correct" here. We can see that in the solution obtained by the SDP algorithm we do not have

TABLE 4.3: Coarse grain correct solutions results averaged with number of experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.550
0.5	0.275	0.500	-
1	0.275	0.350	-
5	0.150	0.350	-
10	0.175	0.300	-

TABLE 4.4: Coarse grain correct solutions results averaged with number of feasible experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.550
0.5	0.392 (7)	0.500 (10)	-
1	0.392 (7)	0.388 (9)	-
5	0.300 (5)	0.350 (10)	-
10	0.430 (4)	0.300 (10)	-

a path that is made by the same flower family. However, we can say that the path obtained by the SDP algorithm is harmonious. In this example, we see that the SDP algorithm with the lambda value equal to 1 has found a solution that is completely different from the solution obtained by shortest path.

4.2.7 Multiple correct solutions

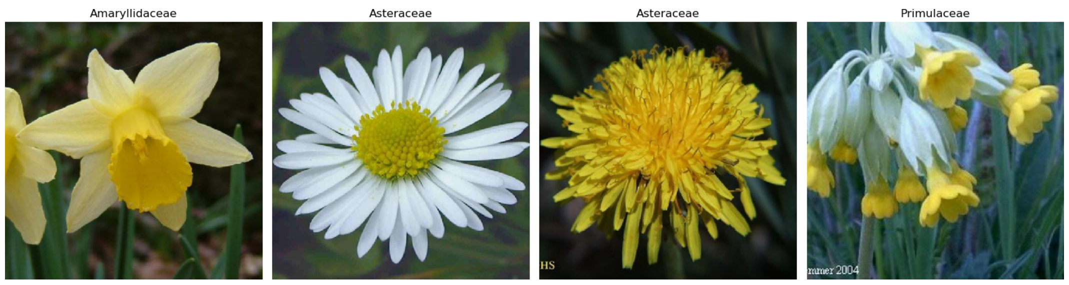


FIGURE 4.13: Multiple correct solutions (fl - fl - fl -fl) - Example 1 of shortest path solution

Unlike the first two examples, this example shows two solutions from two experiments. Since we have two cases of "correct" solutions in this experiment, we introduce the labels fl-fl-fl-fl (flower - flower - flower - flower) and fl-fa-fa-fl (flower - family - family - flower) to clearly indicate which path and which tables they refer to which case. In the first example, the SDP solution is considered to be "correct" with a lambda value equal to 1. In this example fl - fl - fl -fl (flower - flower - flower - flower) means path of 4 same flowers.

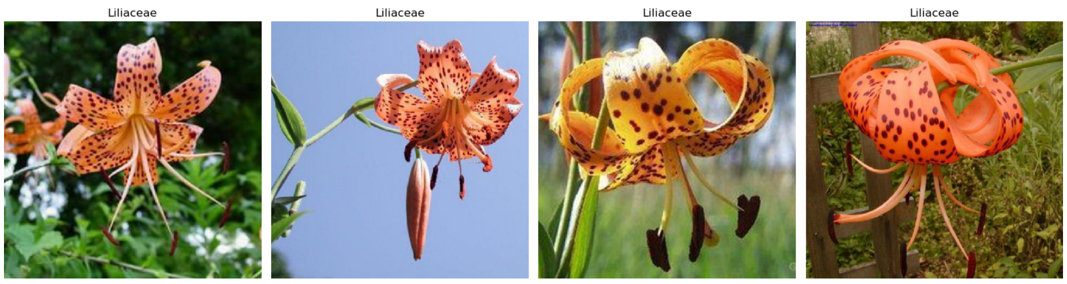


FIGURE 4.14: Multiple correct solutions (fl - fl - fl -fl) - Example 1 of SDP solution with $\lambda = 1$

TABLE 4.5: Multiple correct solutions results (fl - fl - fl -fl) averaged with number of experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.650
0.5	0.425	0.650	-
1	0.125	0.775	-
5	0.075	0.500	-
10	0.100	0.500	-

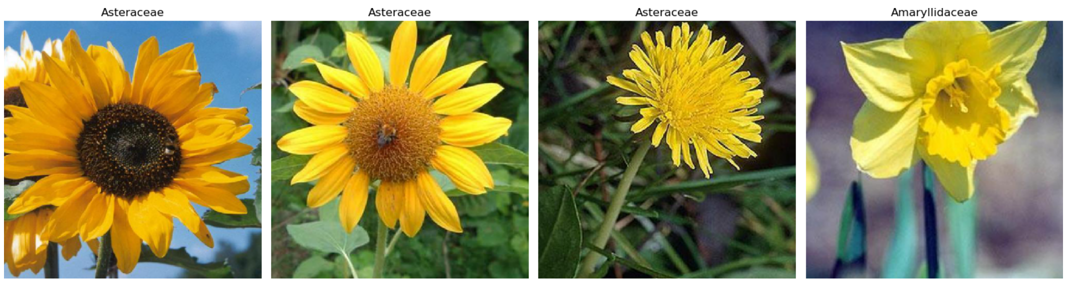


FIGURE 4.15: Multiple correct solutions (fl - fa - fa -fl) - Example 2 of shortest path solution

In the second example we have example with fl - fa - fa - fl (flower - family - family - flower). In this case we have solutions where the first, the second and the third flower are from the same family, but only second and third flower are the

TABLE 4.6: Multiple correct solutions results (fl - fl - fl -fl) averaged with number of feasible experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.625
0.5	0.850 (5)	0.650 (10)	-
1	0.625 (2)	0.860 (9)	-
5	0.375 (2)	0.500 (10)	-
10	0.500 (2)	0.500 (10)	-

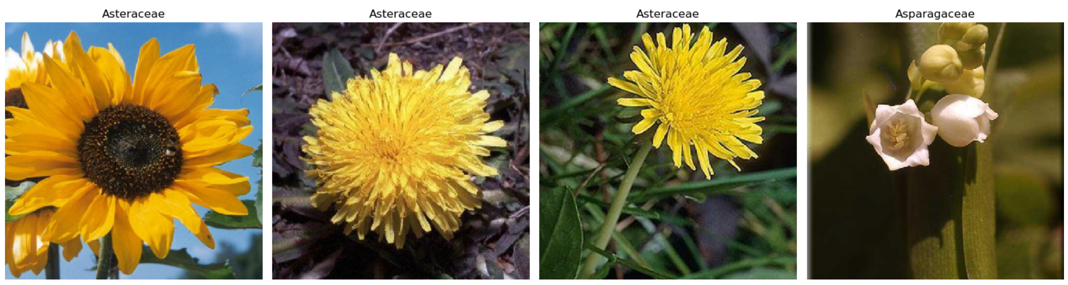


FIGURE 4.16: Multiple correct solutions (fl - fa - fa -fl) - Example 2 of SDP solution with $\lambda = 5$

TABLE 4.7: Multiple correct solutions results (fl - fa - fa -fl) results averaged with number of experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.625
0.5	0.45	0.625	-
1	0.139	0.750	-
5	0.125	0.500	-
10	0.100	0.500	-

same. Both the SDP solution and the shortest path solution are the same accuracy because they differ in the fourth segment where no path contains flowers from the same flower family. Also the solutions of the SDP algorithm and the shortest path solution differ in the second segment. Both solutions contain flowers belonging to the same flower family. The shortest path solution contains a flower that is the same as the flower in the first segment, while the SDP algorithm solution contains a flower that is the same as the flower in the third segment.

TABLE 4.8: Multiple correct solutions results (fl - fa - fa -fl) results averaged with number of feasible experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.625
0.5	0.900 (5)	0.625 (10)	-
1	0.625 (2)	0.833 (9)	-
5	0.625 (2)	0.500 (10)	-
10	0.500 (2)	0.500 (10)	-

TABLE 4.9: No correct solutions results averaged with number of experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.400
0.5	0.325	0.475	-
1	0.150	0.425	-
5	0.150	0.312	-
10	0.08	0.125	-

4.2.8 No correct solutions

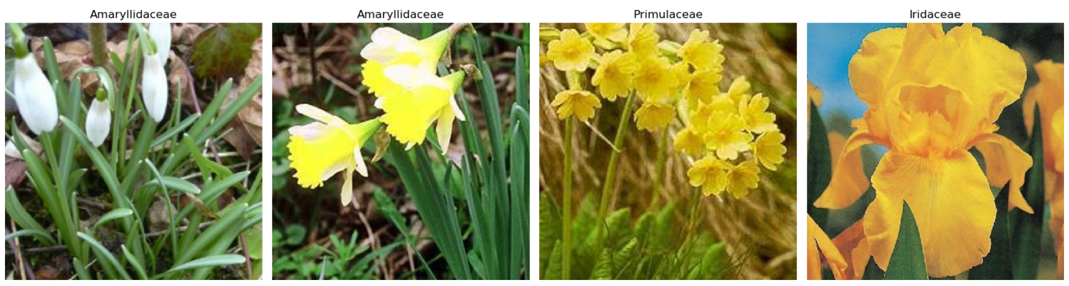


FIGURE 4.17: No correct solutions - Example of shortest path solution

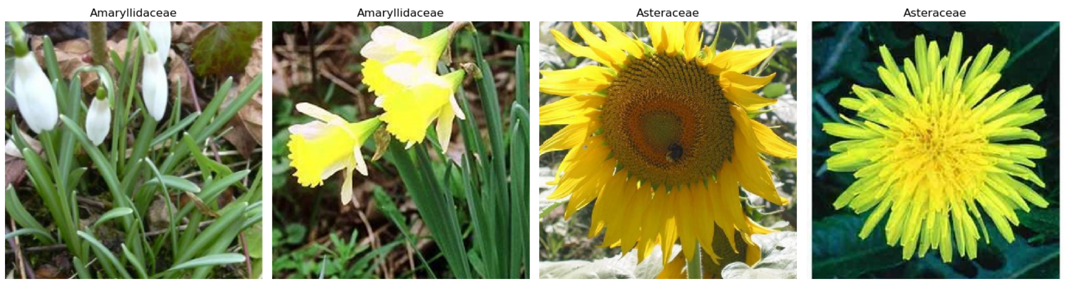


FIGURE 4.18: No correct solutions - Example of SDP solution with $\lambda = 0.5$

This example differs from the previous examples by calculating the results in the tables. As two families of flowers can be found here on the path, the results are divided separately for the first part of the path (so values are divided by 2 and not by 4 as in the previous examples because we look at the path of length two) and after that the second part of the path is considered (since we have four segments in total, the length of second part of path is also length two). Subsequently, the results were summarized and averaged and thus the tables presented in this subsection were created. In the number of experiments that gave the solution, if an odd number is divided by two, it is rounded up.

To summarize, it can be concluded from all the experiments that a higher value for the smoothing parameter will cause the path selected by the SDP algorithm to be different from the path obtained through the SP algorithm. Rounding Procedure 2 proved to be better than Rounding Procedure 1 because in several experiments

TABLE 4.10: No correct solutions results averaged with number of feasible experiments

λ	SDPS1	SDPS2	SPS
0	-	-	0.400
0.5	0.406 (8)	0.475 (10)	-
1	0.500 (3)	0.425 (10)	-
5	0.750 (2)	0.406 (8)	-
10	0.375 (2)	0.218 (8)	-

it gave solutions (where Rounding Procedure 1 did not give a solution, Rounding Procedure 2 was mainly able to find a solution). The more different the value in the tables from the value obtained for the SP are more likely it is that the solutions obtained with the SDP algorithm and SP algorithm will differ.

4.3 Social media storytelling

This section presents experiments involving data and images from the Edinburgh Festival 2016. The data we received and used for our experiments consist of the story name, segment title, the images arranged by segments and adjacency matrix which we use to make a graph. In this section we show 4 examples of stories. Each story has 4 segments and each segment has 4 images. For each example, the solution obtained by the SDP algorithm (indicated in the figures with 'SDP') is shown and the solution obtained by the shortest path algorithm (indicated by 'SP' in the figures) is also shown. The name of the story is shown for each story, as well as the titles by segment.

In [21] it was explained how to select images that are illustrating candidates in segments for a particular story. This was not part of our work. In the following paragraph, we explain how the data on which we applied our algorithm was made. For an image to represent a segment, it must be a relevant image. Retrieval information techniques such as text retrieval methods have been used. Based on text processing of the documents, images that are relevant to represent a particular segment were found. The relevance in these stories is calculated via the BM25 baseline. BM25 is a ranking function that evaluates the relevance of a document [34]. BM25 baseline is a new method from [21], consisting of few steps. The first step is text processing (stop words and stemming filter) for social media posts and text describing segments. The second step is using the BM25 method to rank the publications containing the images that are relevant to describe a given segment. The output of this method are images that are relevant to describe the segment (images from top ranked publications) [21].

The relevance of the images is important, but the transitions from image to image in adjacent segments are also important. In [21] the authors define the transition distance between two sequential images based on certain image features. Features that were used in [21] are luminance, color correlogram, number of edges, color moment, pHash, entropy, concepts, CNN dense, environment, scene category, scene attributes. The adjacency matrix (edge weights in our graph) is made of transition

Story	Segment 1	Segment 2	Segment 3	Segment 4
Joyful Moments at EdFest2016	Standing Ovations and Applauses	Selfies	People Drinking and Eating	Evenings at EdFest2016
Scottish Elements	Bagpipes	Food and drink	Outfits	Military parade
Edinburgh Festival attractions	Music shows	Theater and Comedy	Circus	Street Performances
Street Performances	The Edinburgh Festival is home to one of the most unique celebrations of arts	Street circus is a popular attraction at Edinburgh Festival with several artists such as unicycle jugglers	Street circus is full of colorful artists	Bagpipes

distances. In order to make transition distances, distance features are used. For transition features in our case, the penultimate layer of a ResNet50 is used, the first color moment, color histogram and concepts extracted from the ResNet50. A detailed explanation can be found in [21].

In the examples from the flowers dataset, it turned out that the SDP algorithm would almost certainly find at least one different image on the path than the SP algorithm if the smoothing parameter was equal to 5 or 10. Therefore, we decided to put the smoothing parameter to be 8 in these experiments. So, the smoothing parameter in all examples is equal to 8.

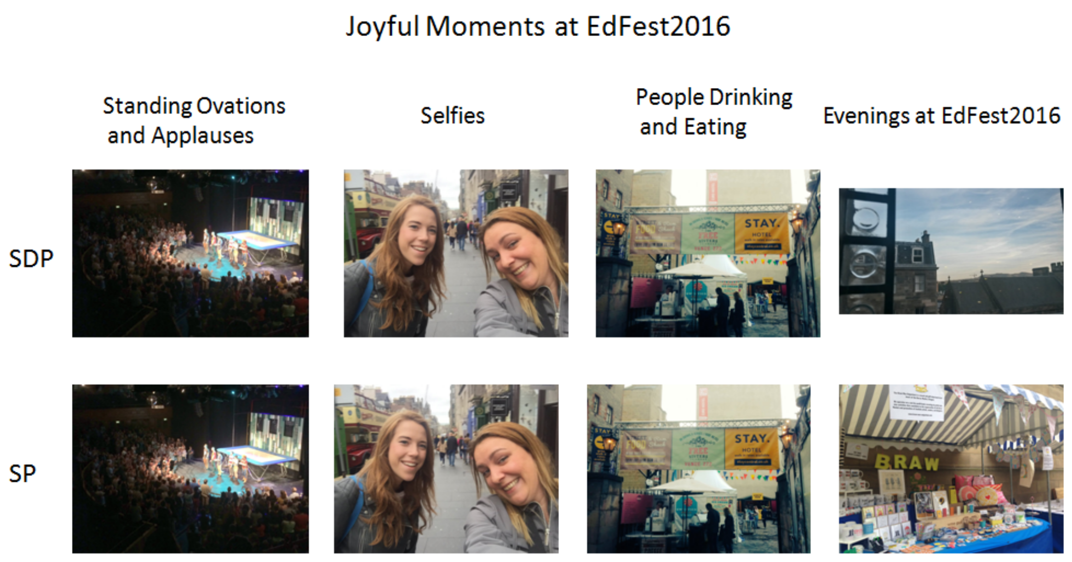

Joyful Moments at EdFest2016
Standing Ovations and Applauses
Selfies
People Drinking and Eating
Evenings at EdFest2016
SDP
SP

FIGURE 4.19: Story 1

In the example of the Story 1, we see that the solution of the SDP algorithm and the solution of the SP algorithm differ in the fourth segment, where the SDP selected an image that more closely matches the name of the segment. We can see that the image representing segment 4 in solution of the SDP algorithm is more consistent with the description of the segment than the image representing segment 4 from the solution of the SP algorithm. From the solution of the SP algorithm in the image representing the third segment cannot be concluded that it has something to do with the evenings.

Similar results as for Story 1 were obtained with Story 2. The SDP algorithm solution and the SP algorithm solution again differ in only one segment, like in Story 1, in this case the last segment is concerned. Unlike Story 1, in the example of Story 2, we see that the images representing the fourth segment obtained by the SDP algorithm and the SP algorithm both correspond to the segment description. However, we can conclude that the image obtained by the SDP algorithm is more visually consistent with colors - if we look at the transition from third segment to fourth segment.

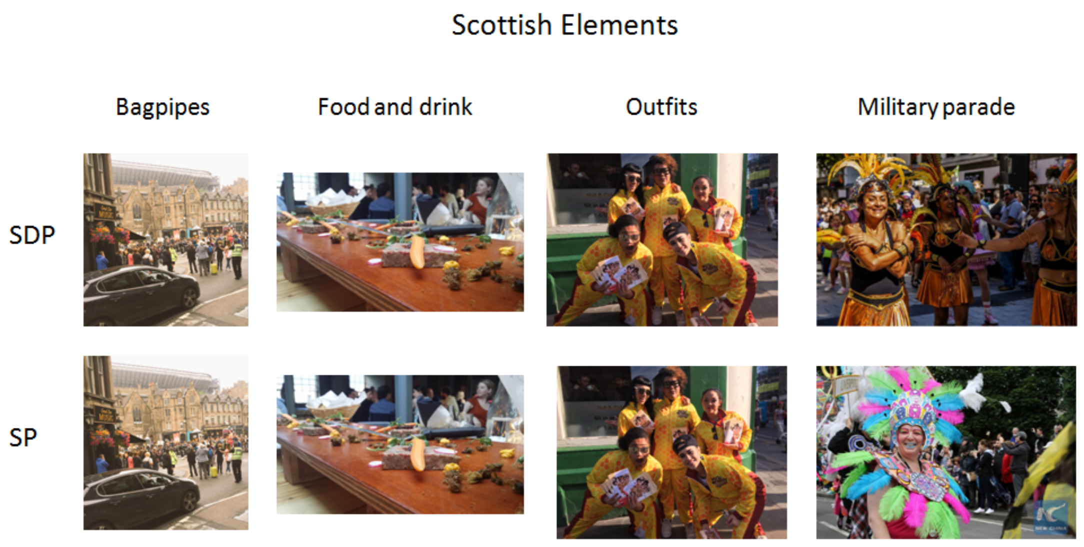


FIGURE 4.20: Story 2

In the example of Story 3, the solution of the SDP algorithm and the solution of the SP algorithm differ in two segments. When solving the SDP algorithm, it can be noticed that both the image illustrating the segment 3 and the image illustrating the segment 4 correspond more to the description of the segment than the images obtained by the SP algorithm.

In the example of Story 4, we have the biggest difference between the SDP algorithm solution and the SP algorithm solution. The paths obtained with the two mentioned algorithms differ in 3 segments. On the solution of the SDP algorithm, we see that in the second segment, theater is presented, while on the solution of the SP algorithm, a group of people is shown. Also, in segment 3, on solution of SDP algorithm can be seen in the distance as if a circus tent was installed, while solution of algorithm again has a selected image of a group of people. In the pictures illustrating the fourth segment, both the SDP algorithm solution and the SP algorithm solution, can be seen street performance.

In the examples of the stories presented in this chapter, we can say that the solution of the SDP algorithm turned out better than the solution of the SP algorithm.

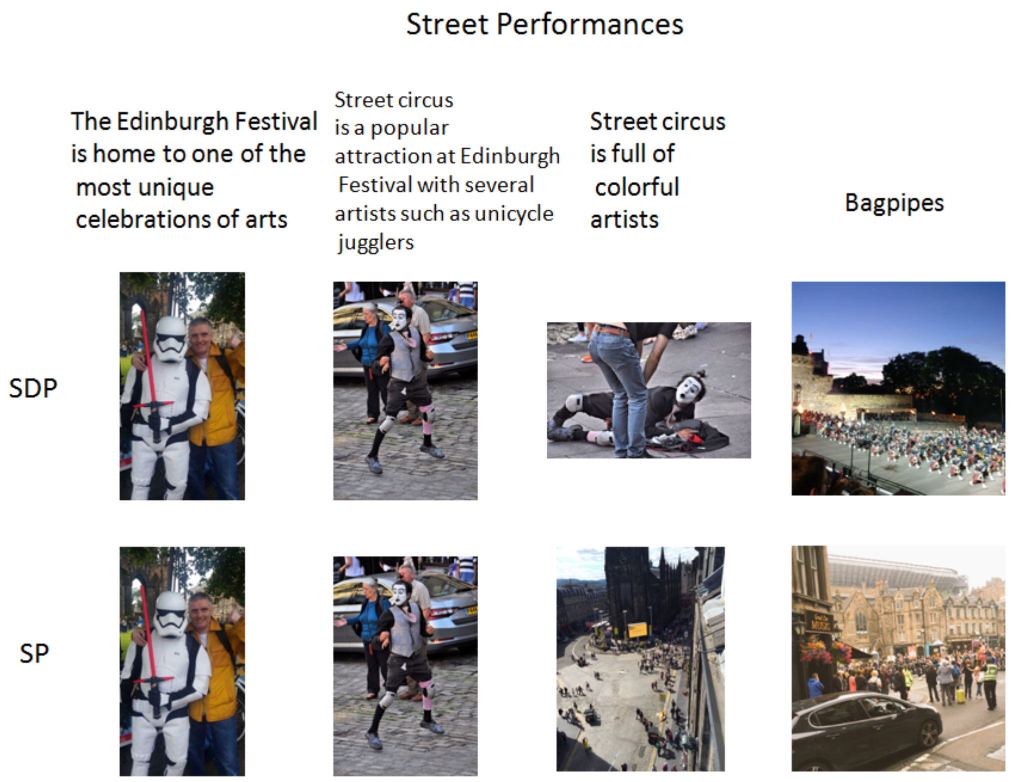

Street Performances
The Edinburgh Festival is home to one of the most unique celebrations of arts
Street circus is a popular attraction at Edinburgh Festival with several artists such as unicycle jugglers
Street circus is full of colorful artists
Bagpipes
SDP
SP

FIGURE 4.21: Story 3

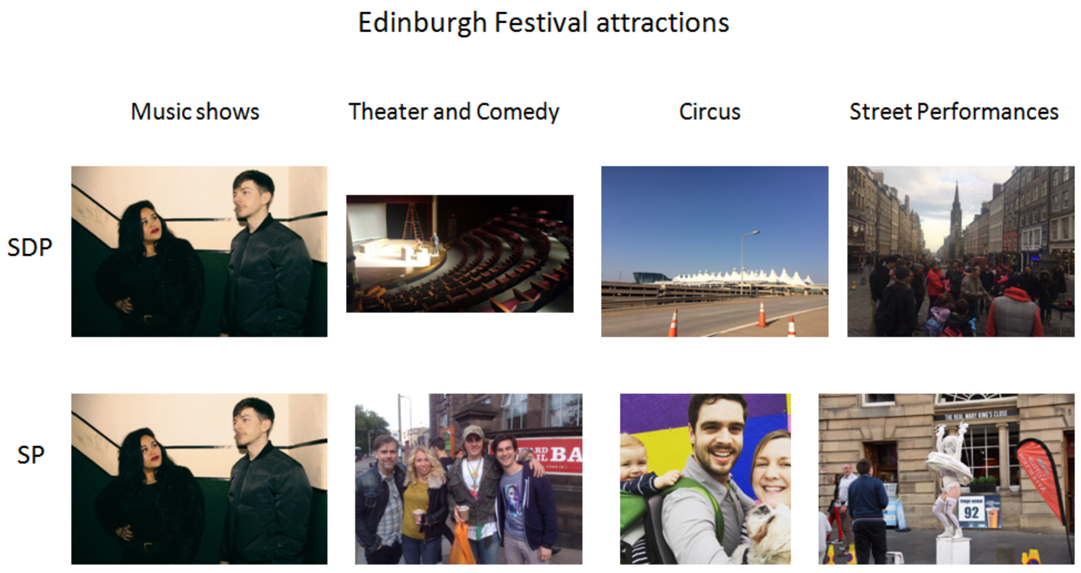


FIGURE 4.22: Story 4

Although the complexity of the SDP algorithm is always greater than the complexity of the SP algorithm (for the SP algorithm, λ is equal to zero, and therefore term 2 will always be zero), in some cases it is still worth trying to find a solution through the SDP algorithm .

Chapter 5

Conclusion and Future Work

Mathematical models are prevalent in many fields today. A common approach is to model a real life problem as an optimization problem. The main challenge of such an approach is to define the problem in the right way, using appropriate constraints.

Our algorithm has shown promising results in the experiments performed, both on real and synthetic data. However there are some downsides of the proposed algorithm.

The implementation of the SDP algorithm itself is much more demanding than the implementation of the shortest path algorithm. In addition, the source and destination nodes we added may create a problem in the sense that due to the definition of the cost function, a path that is not as balanced as possible can be selected.

For example, say we have two paths in a graph (graph with source and destination artificial nodes added) where one path is of edge weights $0-0-0.7-0-0$ and another one is of edge weights $0-0.5-0.1-0.5-0$. So, path $0-0.5-0.1-0.5-0$ is considered to be as balanced as possible. If we set λ to be equal to 2, the SDP algorithm, due to definition of cost function, will give us for path
$0-0-0.7-0-0$:

$$0+0+0.7+0+0+(0.7^2+0.7^2)=1.48$$

For the path $0-0.5-0.1-0.5-0$ we will get :

$$0+0.5+0.1+0.5+0+(0.5^2+0.4^2+0.4^2+0.5^2)=1.92$$

In this case $1.48<1.92$, so the SDP algorithm will choose the path $0-0-0.7-0-0$ which is not as balanced as possible.

If we consider our cost function but without adding source and destination nodes, for the path $0-0-0.7-0-0$ we will get cost function :

$$0+0.7+0+(0.7^2+0.7^2)=1.48$$

and for the path $0-0.5-0.1-0.5-0$ we will get value of cost function:

$$0.5+0.1+0.5+(0.4^2+0.4^2)=1.42$$

In this case $1.42<1.48$, the SDP algorithm will chose the path $0-0.5-0.1-0.5-0$ which is the path as balanced as possible.

Keeping that in mind, it would be interesting to reformulate the algorithm on the problem such that in the cost function, the adders related to the source and destination node are thrown out. This would avoid the problem that may arise in our case.

In future work, it would also be interesting to try another rounding procedure. There are many rounding procedures and it would certainly be possible to obtain a different solution and thus a different path would be found in the graph.

Apart from the application in creating visual stories, this approach could be used for example in travel agencies. In travel agency offers, segments would represent parts of the trips - segments in that case could represent locations or segments could represent daily travel plans.

The approach could also be applied to real estate agencies, where segments could represent different premises in apartments.

In the following figure we can see an example of a path in a graph where segments are kitchen[15], bathroom[26], living room[23], bedroom[19]. On the other example we can see that segments are cities - places which are included in tour of travel agency offer. Segments are Lisbon [16], Cascais [6], Sintra [28] and Cabo da Roca [18].

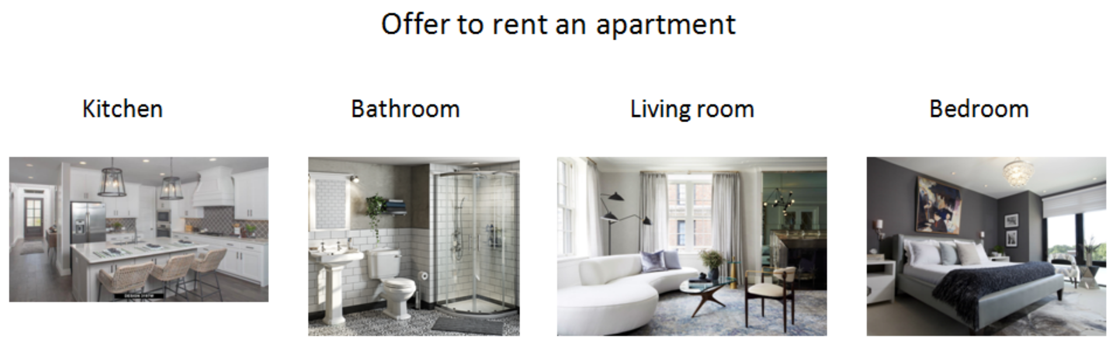


FIGURE 5.1: Future work example 1

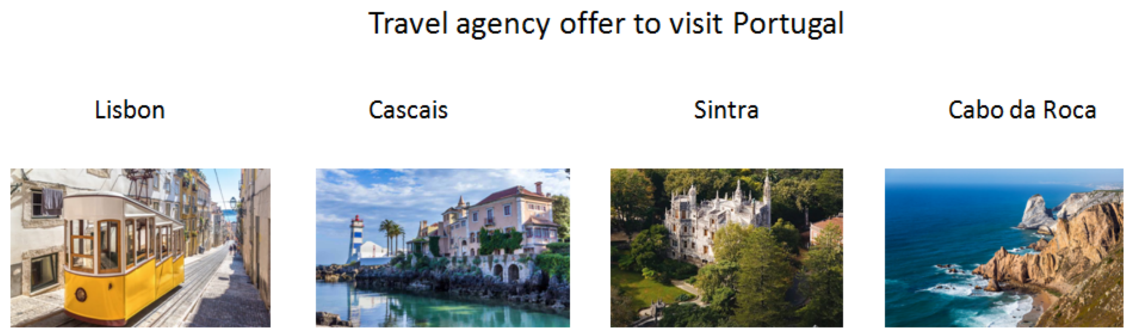


FIGURE 5.2: Future work example 2

These are of course not the only options where this approach could be applied. Any collection of images that can be divided into segments and where we can define some kind of relationship between the images can be good material for this kind of algorithm.

5.1 Flexibility of framework

- There are many ways to modify the objective function. For example error functions can be arbitrarily selected. One possibility is to replace the error function $(w_e - w_f)^2$ with $|w_e - w_f|^2$. So, our cost function in (3.1) :

$$\sum_{e \in E} w_e x_e + \lambda \sum_{\substack{e,f \in E \\ e,f \text{ segment-adjacent}}} \frac{(w_e - w_f)^2}{2} x_e x_f \ ,$$

  will look as follows:

$$\sum_{e \in E} w_e x_e + \lambda \sum_{\substack{e,f \in E \\ e,f \text{ segment-adjacent}}} \frac{|w_e - w_f|}{2} x_e x_f \ .$$

- Another interesting thing that could improve the algorithm is to introduce different λ for adjacent segments. In the case of 4 segments, there would be 3 lambdas: λ_{12} - λ between first and second segment, λ_{23} - λ between second and third segment and λ_{34} - λ between third and fourth segment. With this we would make the path smoother in some parts than in others - depending on what we want to achieve.

To sum up, although the complexity of the SDP algorithm is greater than that of the SP algorithm, it is also more flexible than that of the SP algorithm. Apart from its application in creating visual storylines, the SDP algorithm could find application in various fields. In addition to the examples mentioned above, we leave the reader to consider further applications.

Bibliography

[1] The CVXPY authors. *CVXPY*. 2019. URL: https://www.cvxpy.org/.

[2] Nicolas Bourbaki. *Algebra*. Reading, Massachusetts: Addison-Wesley, 1974.

[3] J. Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[4] NetworkX Developers. *NetworkX*. 2019. URL: https://networkx.github.io/documentation/stable/.

[5] NumPy developers. *NumPy*. 2019. URL: https://numpy.org/.

[6] easyJet. *cascais*. 2015. URL: https://www.easyjet.com/en/holidays/shared/images/guides/portugal/lisbon-coast/cascais.jpg.

[7] University of Oxford Visual Geometry Group Department of Engineering Science. *Flowers dataset*. http://www.robots.ox.ac.uk/~vgg/data/flowers/17/index.html. 2009.

[8] Stephen J.Wright Florian A.Potra. "Interior-points methods". In: *Journal of Computational and Applied Mathematics* 124 (2000), pp. 281–302.

[9] Python Software Foundation. *Python*. 2019. URL: https://www.python.org/.

[10] Alex Clark Fredrik Lundh and Contributors Revision. *PIL*. 2019. URL: https://pillow.readthedocs.io/en/stable/.

[11] Robert M. Freund. *Introduction to Semidefinite Programming (SDP)*. 2009. URL: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-251j-introduction-to-mathematical-programming-fall-2009/readings/MIT6_251JF09_SDP.pdf.

[12] A. M. Geoffrion. "Duality in Nonlinear Programming: A Simplified Applications-Oriented Development". In: *SIAM Review* (1971), 1–37.

[13] Vassos Hadzilacos. *Linear program formulations of the shortest path problem*. 2017.

[14] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[15] Perry Homes. *Kitchen*. 2019. URL: https://www.perryhomes.com/content/photogallery/gallery/b717b237-26a0-469b-98a5-034d6676faf3.jpg.

[16] demeures internationales. *lisbon*. 2019. URL: https://www.demeures-internationales.fr/file/si738169/lisboa-lisbonne-fi14454475.jpg.

[17] Eric Firing Michael Droettboom John Hunter Darren Dale and the Matplotlib development team. *matplotlib*. 2019. URL: https://matplotlib.org/.

[18] Living Lisboa. *cabo da roca*. 2017. URL: http://www.living-lisboa.com/wp-content/uploads/2017/03/daroca2.jpg.

[19] Boca do Lobo. *bedroom*. 2019. URL: https://www.bocadolobo.com/en/inspiration-and-ideas/wp-content/uploads/2018/03/Discover-the-Ultimate-Master-Bedroom-Styles-and-Inspirations-6_1.jpg.

[20] G. Mackiw. “A Note on the Equality of the Column and Row Rank of a Matrix”. In: *Mathematics Magazine* (1995), p. 68.

[21] Gonçalo Barreto Ferreira Marcelino. “A computational approach to the art of visual storytelling”. Thesis advisor: João Miguel da Costa Magalhães. MA thesis. Universidade Nova de Lisboa, 2018.

[22] Joachim Dahl Martin S. Andersen and Lieven Vandenberghe. *CVXOPT*. 2019. URL: `https://cvxopt.org/`.

[23] Hearst Magazine Media. *Living room*. 2019. URL: `https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/rl-655-park-ave-formal-living-room-009-1539282355.jpg`.

[24] Jerome Le Ny. “Rounding Techniques for Semidefinite Relaxations”. In: *Conference Proceedings* (2005).

[25] Ryan O’Donnell. *Lecture 14: Semidefinite Programming and Max-Cut*. 2008. URL: `https://www.cs.cmu.edu/~anupamg/adv-approx/lecture14.pdf`.

[26] Victoria Plum. *bathroom*. 2019. URL: `https://images.victoriaplum.com/pages/5ab12ced-182c-4a4b-be90-ec8faec843ed.jpg?auto=format`.

[27] S. Poljak, F. Rendl, and H. Wolkowicz. “A recipe for semidefinite relaxation for $(0, 1)$–quadratic programming”. In: *Journal of Global Optimization* 7.1 (1995), pp. 51–73.

[28] Turismo de Portugal. *sintra*. 2013. URL: `https://www.visitportugal.com/sites/default/files/styles/encontre_detalhe_poi_destaque/public/mediateca/N4.MPS1582d.jpg?itok=x7WnckRY`.

[29] A.J.M. Spencer. *Continuum Mechanics*. Mineola, New York: Dover Publications, 1980.

[30] StackExchange. *Positive semidefinite cone*. 2016. URL: `https://i.stack.imgur.com/8MiLU.png`.

[31] Leonardo Taccari. “Integer programming formulations for the elementary shortest path problem”. In: *European Journal of Operational Research* 252 (2016), pp. 122–130.

[32] scikit-image development team. *skimage*. 2019. URL: `https://scikit-image.org/`.

[33] L. Vandenberghe and S. Boyd. “Semidefinite Programming”. In: *SIAM Review* 38.1 (1996), pp. 49–95. DOI: `10.1137/1038003`.

[34] Wikipedia. *BM25*. `https://en.wikipedia.org/wiki/Okapi_BM25`. 2019.

[35] Wikipedia. *Social media*. `https://en.wikipedia.org/wiki/Social_media`. 2019.

Kristina Licenberger was born in Vrbas on 10^{th} of September 1994. She finished elementary school "Petefi Brigada" and music school "Isidor Bajić" in Kula. After that, she finished high school "Žarko Zrenjanin" in Vrbas. She recieved her Bachelors degree in Applied Mathematics in 2017 at Faculty of Sciences, University of Novi Sad, and she continued her Master studies in the field of Data Science at the same faculty. After passing all the exams, she spent two months in Lisbon doing her master's degree research in collaboration with the NOVA School of Science and Technology, NOVA University of Lisbon.

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:
RBR
Identifikacioni broj:
IBR
Tip dokumentacije: monografska dokumentacija
TD
Tip zapisa: tekstualni štampani materijal
TZ
Vrsta rada: master rad
VR
Autor: Kristina Licenberger
AU
Mentor: dr Dragana Bajović
MN
Naslov rada: Kreiranje vizuelnih priča korišćenjem semidefinitnog programiranja
NR
Jezik publikacije: engleski
JP
Jezik izvoda: e
JI
Zemlja publikovanja: Republika Srbija
ZP
Uže geografsko područje: Vojvodina
UGP
Godina: 2019.
GO
Izdavač: autorski reprint
IZ
Mesto i adresa: Novi Sad, Trg Dositeja Obradovića 4
MA
Fizički opis rada: 5 poglavlja, 48 strana, 35 lit. citata, 36 grafika
FO
Naučna oblast: matematika
NO
Naučna disciplina: primenjena matematika
ND
Ključne reči: Semidefinitno programiranje, Semidefinitna relaksacija, Konveksna optimizacija, Najkraći put, Teorija grafova, Vizuelni sižei
UDK
Čuva se: u biblioteci Departmana za matematiku i informatiku, Prirodno-matematičkog fakulteta, u Novom Sadu
CU
Važna napomena:
VN
Izvod: Motivacija je pronađena u radu od G.Marcelino koji koristi najkraći put u grafu kako bi pronašao najskladniji niz slika koji će ilustrovati određen događaj. U

ovom radu vršimo pregled teorije i praktičnu implementaciju algoritma za pronalazak puta sa balansiranim težinama grana u grafu. Definišemo pronalazak puta u grafu optimizacionim problemom a zatim na njemu primenimo semidefinitnu relaksaciju. Nakon semidefinitne relaksacije primenjujemo procedure zaokruživanja kako bi dobili rešenje - putanju. U radu su prikazani eksperimenti izvršeni na sintetičkim podacima, podacima koji sadrže slike cveća i podacima koji sadrže slike sa Edinburškog festivala 2016.godine.

IZ
Datum prihvatanja teme od strane NN veca: 29.08.2019.
DP
Datum odbrane:
DO
Članovi komisije:
KO
Predsednik: dr Dušan Jakovetić, docent
Mentor: dr Dragana Bajović, docent
Član: dr Miloš Stojaković, redovni profesor

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
KEY WORDS DOCUMENTATION

Accession number:
ANO
Identification number:
INO
Document type: monograph type
DT
Type of record: printed text
TR
Contents code: master thesis
CC
Author: Kristina Licenberger
AU
Mentor: PhD Dragana Bajović
MN
Title: Semidefinite Programming Approach to Visual Storyline Creation
XI
Language of text: English
LT
Language of abstract: e
LA
Country of publication: Republic of Serbia
CP
Locality of publication: Vojvodina
LP
Publication year: 2019.
PY
Publisher: author's reprint
PU
Publ. place: Novi Sad, Trg Dositeja Obradovića 4
PP
Physical description: 5 chapters, 48 pages, 35 references, 36 figures
PD
Scientific field: mathematics
SF
Scientific discipline: applied mathematics
SD
Key words: Semidefinite programming, Semidefinite relaxation, Convex optimization, Shortest path, Graph theory, Visual summaries
UC
Holding data: Department of Mathematics and Informatics's Library, Faculty of Sciences, Novi Sad
HD
Note:
N
Abstract: The motivation for this problem comes from the recent work by G. Marcelino et al. that uses shortest path to find the most coherent sequence of images that illustrate a given event. In this work we present the theoretical background

and practical implementation of algorithm for finding paths with balanced costs in weighted graph. We define finding a path in a graph by an optimization problem and then apply semidefinite relaxation to it. After semidefinite relaxation, we apply rounding procedures to get the solution - the path. The work contains experiments performed on synthetic data, data containing images of flowers and data containing images from the Edinburgh Festival 2016.

AB
Accepted by the Scientific Board on: 29.08.2019.
ASB
Defended:
DE