Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Application of Simplex Algorithm for Submodular Discrete Energy Minimization with Binary Variables

Ekaterina Tiuzhina

Supervisor: RNDr. Daniel Průša, Ph.D.

Study Program: Open Informatics

Field of Study: Computer and Information Science

May 22, 2018

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Tiuzhina Ekaterina**	Personal ID number:	**434839**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Application of Simplex Algorithm for Submodular Discrete Energy Minimization with Binary Variables

Bachelor’s thesis title in Czech:

Aplikace simplexového algoritmu pro minimalizaci submodulární diskrétní energie s binární proměnnými

Guidelines:

1. Describe the problem of discrete energy minimization with binary variables and submodular pairwise costs [1]. Discuss its complexity and explain its importance for computer vision.
2. Formulate the problem as a linear program and derive a graph-based simplex method [3] solving it, similarly as it has been done in [2] for nonsubmodular instances.
3. Implement the algorithm in C++. Analyze its behaviour experimentally. Test it using random and computer vision instances.

Bibliography / sources:

[1] D. Průša: Graph-based Simplex Method for Pairwise Energy Minimization with Binary Variables. CVPR, 2015
[2] V. Kolmogorov, K. Rother: Minimizing Non-submodular Functions with Graph-cuts - a Review. PAMI, 2007
[3] G. B. Dantzig, M. N. Thapa: Linear programming 1 : Introduction. Springer, 1997

Name and workplace of bachelor’s thesis supervisor:

RNDr. Daniel Průša, Ph.D., Machine Learning, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **08.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

RNDr. Daniel Průša, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt   Student’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

I would like to sincerely thank my thesis supervisor RNDr. Daniel Průša, Ph.D. for allowing me to work on this topic, his immense knowledge, patience and help he had given me.
I would also like thank everyone who supported me during my studies, especially my wonderful family and my dearest boyfriend.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague on May 22, 2018 ....

Abstract

Discrete energy minimization is a well-known and widely used model which is substantive for many low-level computer vision problems. Since minimizing energy is often even an NP-hard problem, a great variety of approximation methods has been proposed. One of them is based on an LP relaxation of the problem. It has been shown, that it is possible to implement a graph-based simplex algorithm for energy minimization instance with binary variables. Moreover, the existing implementation of this algorithm is more effective for some data instances than alternative methods of approximation.

In this thesis we focus on implementing a graph-based simplex algorithm for submodular instances of energy minimization problems. We define a special LP relaxation for submodular instances, present a graph-based simplex algortihm for solving it and implement it in C++.

Keywords: discrete energy minimization, LP relaxation, simplex method, graph algorithms, submodular function.

Abstrakt

Minimalizace diskrétní energie je známý a široce používány model, který je podstatný pro velký počet problémů nizkourovňového počítačového vidění. Vzhledem k tomu, že minimalizace energie může být až NP-těžkým problémem, bylo navrženo mnoho způsobů aproximace. Jedním způsobem je LP relaxace problému. Bylo ukázáno, že je možné implementovat grafový simplexový algoritmus pro minimalizaci diskrétní energie s binarními proměnnými. Výhodou této metody je, že na určitých datech je efektivnější než alternativní metody aproximace.

V této bakalářské práci se zabýváme implementací grafového simplexového algoritmu pro submodulární instance problému minimalizace diskrétní energie. Nejdřív definujeme speciální LP relaxaci pro submodularní instance, poté navrhneme grafový simplexový algoritmus, který následně implementujeme v jazyce C++.

Klíčová slova: minimalizace diskrétní energie, LP relaxace, simplexová metoda, grafové algoritmy, submodularní funkce.

Contents

1 Introduction **1**

1.1 Background 1

1.2 Thesis structure 1

2 Preconditions **3**

2.1 Problem instance 3

2.2 Simplex method 3

3 Basis structure **7**

3.1 LP relaxation 7

3.2 LP variables 8

3.3 Graph structure 10

4 Algorithm **13**

4.1 Terminology 13

4.2 Initial graph state 15

4.3 Transition to simplex method 17

4.4 Root step 20

4.5 Edge step 22

4.6 General algorithm 24

5 Testing **27**

5.1 Generated random test data 27

5.2 Real test data 29

5.3 Analysis of results 29

6 Conclusion **31**

A Data format **34**

B Contents of CD **35**

C User manual **35**

1 Introduction

1.1 Background

Discrete energy minimization is a well-known and widely used model which is substantive for many low-level computer vision problems [5, 8, 9]. While minimizing energy is a hard (often even NP-hard) problem, a great variety of approximation methods has been proposed. One of them is the popular algorithm developed by Boykov and Kolmogorov which is based on graph cuts [5].

An alternative technique to solve the energy minimization is based on linear programming (LP) relaxation of the problem [10]. An approximate solution is obtained by applying a rounding scheme to a non-integral solution. For binary variables, it is known that an optimal solution of the LP relaxation is a vector with components from {0, 1/2, 1} (so-called half-integral vector) [2]. The value 1/2 represents an undecided variable. Each such solution can be extended into an optimal solution of the original problem preserving the decided variables [2]. The LP relaxation approach gives an optimal solution to the energy minimization problem in the case of so called *submodular* instances. This time, an optimal solution of the LP relaxation with components in {0, 1} can always be found [4].
A disadvantage of the LP relaxation approach might be that general LP solvers, which are based on the simplex or interior point methods, are prohibitively inefficient for large-scale instances. Moreover, there is little hope for a special solver since the LP relaxation of energy minimization problem is as hard as any general LP [7]. However, the situation is easier in the case of energy minimization with two labels as the LP relaxation reduces to max-flow [5]. It is also possible to implement an efficient, graph-based version of the simplex method [6], also known as simplex network for transportation, assignment and minimum cost-flow problems [3]. Moreover, it has been demostrated in [6] that on some instances the graph-based simplex algorithm runs faster then the max-flow based algorithm by Boykov and Kolmogorov.
In this thesis we focus on altering the graph-based simplex method implementation presented in [6], by the assumption that the input energy minimization problems are submodular. In the first place, we formulate an easier LP relaxation of the problem. Then we propose a graph-based implementation of the simplex method, which runs each iteration only on variables which are affected by it.

1.2 Thesis structure

In Section 2 we formulate the problem of pairwise energy minimization. We also describe basic principles of simplex method which are essential for implementing our algorithm. In Section 3 we formulate LP relaxation of the problem and describe the data structure which is going to be used in our algorithm implementation. The algorithm itself and its individual steps are presented in Section 4. In Section 5 we focus on testing our program on simple artificial datasets created by the author as well as some public test instances from [1], for example, surface fitting.

2 Preconditions

2.1 Problem instance

The task of pairwise energy minimization is to compute

$$\min_{\mathbf{k}\in K^V} \Bigg[\sum_{u\in V} \theta_u(k_u) + \sum_{\{u,v\}\in E} \theta_{uv}(k_u, k_v) \Bigg] \tag{1}$$

where V is a finite set of *objects*, E is a set of *object pairs* (thus, $G = (V, E)$ is an undirected graph), K is a finite set of *labels*, and the functions $\theta_u\colon K \to \mathbb{R}$ and $\theta_{uv}\colon K \times K \to \mathbb{R}$ are unary and pairwise *weights* [6]. We adopt that $\theta_{uv}(k, \ell) = \theta_{vu}(\ell, k)$. We shortly write $\theta_u(k)$ as $\theta_{u;k}$ and $\theta_{uv}(k, \ell)$ as $\theta_{uv;k\ell}$. The weights together form a vector $\boldsymbol{\theta} \in \mathbb{R}^I$ with

$$\begin{aligned} I = \{\, (u;k) \mid u \in V,\ k \in K \,\} \cup \\ \{\, (uv;k\ell) \mid \{u,v\} \in E;\ k,\ell \in K \,\}. \end{aligned} \tag{2}$$

We consider only energy minimization with two labels, hence K is fixed as $\{0, 1\}$ [6]. An instance of the problem is thus fully defined by a tuple $(V, E, \boldsymbol{\theta})$ [6].

Example 1. *Let* $V = \{a, b\}$*,* $E = \{\{a, b\}\}$*,* $\theta_{a;0} = 1$*,* $\theta_{a;1} = -1$*,* $\theta_{b;0} = 2$*,* $\theta_{b;1} = 0$*,* $\theta_{ab;00} = 0$*,* $\theta_{ab;01} = 1$*,* $\theta_{ab;10} = 2$ *and* $\theta_{ab;11} = 1$*. Then* $(V, E, \boldsymbol{\theta})$ *is an instance of energy minimization with two objects (a and b) and one object pair (*$\{a, b\}$*).*

We say that an instance $(V, E, \boldsymbol{\theta})$ is *submodular* if

$$\forall \{u, v\} \in E : \theta_{uv;01} + \theta_{uv;10} \geq \theta_{uv;00} + \theta_{uv;11}.$$

Weights of each energy minimization problem instance can be rewritten into an equivalent form where

$$\forall \{u, v\} \in E : \theta_{uv;00} = \theta_{uv;11} = 0.$$

Note that this implies $\theta_{uv;01} + \theta_{uv;10} \geq 0$ for submodular instances.

2.2 Simplex method

Simplex method or *simplex algortithm* is a popular algorithm for solving linear programs [11]. This method requires a linear program to be in the following form:

$$\min \quad c^T x$$

subject to:

$$\begin{aligned} Ax &= b \\ x &\geq \mathbf{0} \end{aligned}$$

This form is sometimes called the *standard form* [11].
Let us show how simplex method works on the following example:

$$\min \quad -3x_2 - 2x_3 \tag{3}$$

subject to:

$$x_1 + x_2 + 3x_3 = 5 \tag{4a}$$
$$x_2 - 4x_3 + x_4 = 2 \tag{4b}$$
$$x_3 + x_5 = 4 \tag{4c}$$
$$2x_2 + x_3 + x_6 = 3 \tag{4d}$$
$$x \geq \mathbf{0} \tag{4e}$$

Note that this example is already in the standard form. Let us write the coefficients from objective function and constraints into simplex tableau.

$$\begin{array}{c|cccccc|c} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & d \\ \hline & 0 & -3 & -2 & 0 & 0 & 0 & 0 \\ \hline x_1 & 1 & 1 & 3 & 0 & 0 & 0 & 5 \\ x_4 & 0 & 1 & -4 & 1 & 0 & 0 & 2 \\ x_5 & 0 & 0 & 1 & 0 & 1 & 0 & 4 \\ x_6 & 0 & 2 & 1 & 0 & 0 & 1 & 3 \end{array} \tag{5}$$

The top row holds coefficients from the objective function vector, therefore, we refer to it as *objective function row* [11]. The rightmost entry in the objective function row corresponds to the value of objective function and holds value 0. We call the rest of the rows *LP constraint rows* or simply *constraint rows*.
We divide the set of n variables into m *basic* variables and $n - m$ *non-basic* variables, where m is a number of constraint rows [11]. The set $I \subseteq \{1, 2, ..., n\}$ is called *basis* if I consists of m elements and all columns from simplex tableau with indices from I are linearly independent, therefore, they form a regular $m \times m$ matrix [11]. We say the basis is *standard* if the basis columns form an identity matrix [11]. As simplex method works only with standard basis, we may have to perform a Gaussian elimination on a simplex tableau to transform the basis into standard one [11]. In our example the basis is $I = \{1, 4, 5, 6\}$ and it is already standard.

If there is at least one negative entry in the objective function row, then the objective function value can be lowered by making the corresponding variable basic [11]. We call this new basic variable *entering variable*. Along with choosing the new entry to basis, we choose one variable from basis which later becomes non-basic, this variable is called *leaving variable*. The simplex iteration consists of the following steps:

1. Check termination condition. If there are no negative entries in the objective function row, then the objective function value cannot be further lowered, meaning that we have found the minimal value and the algorithm is terminated [11].

2. Choose the smallest negative entry from the objective function row. The corresponding variable becomes the new entering variable [11].

$$\begin{array}{c|cccccc|c} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & d \\ \hline & 0 & \circled{-3} & -2 & 0 & 0 & 0 & 0 \\ \hline x_1 & 1 & 1 & 3 & 0 & 0 & 0 & 5 \\ x_4 & 0 & 1 & -4 & 1 & 0 & 0 & 2 \\ x_5 & 0 & 0 & 1 & 0 & 1 & 0 & 4 \\ x_6 & 0 & 2 & 1 & 0 & 0 & 1 & 3 \end{array} \tag{6}$$

3. Choose the leaving variable using the following *pivotal rule* for the column corresponding to entering variable. Firstly, for each row we compute the ratio of the rightmost entry to the related entry from the column which corresponds to the chosen entering variable. The smallest non-negative finite value is called *pivot* [11]. The variable from the leftmost column which refers to pivotal row is chosen as leaving variable.
   We see that x_5 cannot become leaving variable as the value in entering column corresponding to its row is zero, therefore, the ratio is not a finite number. From the remaining ratios 5/1, 2/1, 3/2 we choose the minimal non-negative value, in this case it is 3/2.

$$\begin{array}{c} \\ \\ x_1 \\ x_4 \\ x_5 \\ x_6 \end{array} \left[\begin{array}{cccccc|c} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & d \\ \hline 0 & -3 & -2 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 3 & 0 & 0 & 0 & 5 \\ 0 & 1 & -4 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 1 & 0 & 4 \\ 0 & \textcircled{2} & 1 & 0 & 0 & 1 & 3 \end{array} \right] \tag{7}$$

4. Perform the *Gaussian elimination* on the simplex tableau:

   - Modify the row corresponding to pivotal value so that the chosen pivot equals one.

$$\begin{array}{c} \\ \\ x_1 \\ x_4 \\ x_5 \\ x_6 \end{array} \left[\begin{array}{cccccc|c} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & d \\ \hline 0 & -3 & -2 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 3 & 0 & 0 & 0 & 5 \\ 0 & 1 & -4 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 1 & 0 & 4 \\ 0 & 1 & 0.5 & 0 & 0 & 0.5 & 1.5 \end{array} \right] \tag{8}$$

   - Add a multiple of pivotal row to every remaining constraint row so that values in constraint rows corresponding to entering variable column are zero.

$$\begin{array}{c} \\ \\ x_1 \\ x_4 \\ x_5 \\ x_6 \end{array} \left[\begin{array}{cccccc|c} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & d \\ \hline 0 & -3 & -2 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 2.5 & 0 & 0 & -0.5 & 3.5 \\ 0 & 0 & -4.5 & 1 & 0 & -0.5 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 & 4 \\ 0 & 1 & 0.5 & 0 & 0 & 0.5 & 1.5 \end{array} \right] \tag{9}$$

   - Add a multiple of pivotal row to objective function row in order to set the coefficient of the entering variable to zero.

$$\begin{array}{c} \\ \\ x_1 \\ x_4 \\ x_5 \\ x_6 \end{array} \left[\begin{array}{cccccc|c} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & d \\ \hline 0 & 0 & -0.5 & 0 & 0 & 1.5 & 4.5 \\ \hline 1 & 0 & 2.5 & 0 & 0 & -0.5 & 3.5 \\ 0 & 0 & -4.5 & 1 & 0 & -0.5 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 & 4 \\ 0 & 1 & 0.5 & 0 & 0 & 0.5 & 1.5 \end{array} \right] \tag{10}$$

5. Repeat the steps above until terminal condition is reached.

In the second simplex iteration we select the following entering variable and pivot:

$$\begin{array}{c|cccccc|c|} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & d \\ \hline & 0 & 0 & -0.5 & 0 & 0 & 1.5 & 4.5 \\ \hline x_1 & 1 & 0 & \textcircled{2.5} & 0 & 0 & -0.5 & 3.5 \\ x_4 & 0 & 0 & -4.5 & 1 & 0 & -0.5 & 0.5 \\ x_5 & 0 & 0 & 1 & 0 & 1 & 0 & 4 \\ x_2 & 0 & 1 & 0.5 & 0 & 0 & 0.5 & 1.5 \end{array} \qquad (11)$$

After performing the Gaussian elimination the simplex tableau holds the following entries:

$$\begin{array}{c|cccccc|c|} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & d \\ \hline & 0.2 & 0 & 0 & 0 & 0 & 1.4 & 5.2 \\ \hline x_3 & 0.4 & 0 & 1 & 0 & 0 & -0.2 & 1.4 \\ x_4 & 1.8 & 0 & 0 & 1 & 0 & -1.4 & 6.8 \\ x_5 & -0.4 & 0 & 0 & 0 & 1 & 0.2 & 2.6 \\ x_2 & -0.2 & 1 & 0 & 0 & 0 & 0.6 & 0.8 \end{array} \qquad (12)$$

As there is no negative entry in the first row, the minimal value of objective function is found, thus, simplex algorithm is terminated. The right top corner of simplex tableau holds the negative value of objective function [11].
In our example the solution is $d = -5.2$. The basis is $\{x_2,\, x_3,\, x_4,\, x_5\}$ and the values of basic variables are $x_2 = 0.8$, $x_1 = 1.4$, $x_4 = 6.8$ and $x_5 = 2.6$. Non-basic variables x_1 and x_6 are equal to zero.

3 Basis structure

3.1 LP relaxation

The pairwise energy minimization with two labels has the following linear programming (LP) relaxation [10]:

$$\min \sum_{u\in V} \sum_{k\in\{0,1\}} \theta_{u;k} x_{u;k} + \sum_{\{u,v\}\in E} \sum_{k,\ell\in\{0,1\}} \theta_{uv;k\ell} x_{uv;k\ell} \tag{13}$$

subject to:

$$\begin{aligned} x_{u;0} + x_{u;1} &= 1, && u \in V && \text{(14a)}\\ x_{uv;k0} + x_{uv;k1} &= x_{u;k}, && k \in \{0,1\}, \{u,v\} \in E && \text{(14b)}\\ x &\geq \mathbf{0}. && && \text{(14c)} \end{aligned}$$

$x_{u;k}$ and $x_{uv;k\ell}$ are LP variables. We adopt $x_{uv;k\ell} = x_{vu;\ell k}$. Note that the graph-based method in [6] was derived for this LP relaxation. It is known that for a submodular instance there is always an optimal solution with all $x_{u;k}$ and $x_{uv;k\ell}$ in {0, 1} [2]. In this case we can even propose a simpler LP relaxation variant, with fewer LP variables and constraints.

Weights of each energy minimization can be rewritten into an equivalent form where

$$\forall\{u,v\} \in E : \theta_{uv;00} = \theta_{uv;11} = 0$$

(see Section 4 for details on the so called *reparametrization*). We note that this implies $\theta_{uv;01} + \theta_{uv;10} \geq 0$ for submodular instances.

After the reparametrization, we formulate the following LP relaxation.

$$\min \sum_{u\in V} (\theta_{u;0} x_{u;0} + \theta_{u;1} x_{u;1}) + \sum_{\{u,v\}\in E} (\theta_{uv;01} x_{uv;01} + \theta_{uv;10} x_{uv;10}) \tag{15}$$

subject to:

$$\begin{aligned} x_{u;0} + x_{u;1} &= 1, && u \in V && \text{(16a)}\\ x_{u;0} + x_{uv;10} &= x_{v;0} + x_{uv;01}, && \{u,v\} \in E && \text{(16b)}\\ x &\geq \mathbf{0}. && && \text{(16c)} \end{aligned}$$

To prove that this LP system is equivalent of the original LP relaxation, we show that there is always an optimal solution with variables $x_{uv;k\ell}$ in the range from 0 to 1. More precisely, there is always an optimal solution which fulfills

$$\forall\{u,v\} \in E : (\min\{x_{uv;10}, x_{uv;01}\} = 0) \ \wedge \ (\max\{x_{uv;10}, x_{uv;01}\} \leq 1)\,. \tag{17}$$

Proof: If $x_{uv;10}, x_{uv;01} \geq \Delta > 0$, decreasing $x_{uv;10}$ and $x_{uv;01}$ by Δ decreases the objective function by $(\theta_{uv;01} + \theta_{uv;10})\Delta$ which is a non-negative as $\theta_{uv;01} + \theta_{uv;10} \geq 0$.

The space complexity of simplex tableau for data set with m objects and n object pairs is $\mathrm{O}((m+n)^2)$. Though we can use tableau-based simplex method for small data instances, it would not work for large data because it would not be memory efficient. This is the reason why we propose to implement a graph-based simplex method. We shall expand the definition of the graph $G = (V, E)$ mentioned in section 2.1.

3.2 LP variables

We define a new graph $G' = (V', E')$ for solving LP relaxation of a submodular instance by expanding a previously defined graph $G = (V, E)$ (1). We say V' is a finite set of *LP variable nodes* and E' is a set of *LP variable edges*. The numbers of LP variable nodes matches the number of objects in graph G and the number of LP variable edges matches the number of object pairs in G. The new graph, however, has expanded structure compared to previous graph. Each LP variable node consists of two parts: $x_{u;0}$ and $x_{u;1}$ where $u \in V'$. We define $x_{u;0}$ and $x_{u;1}$ as *nodes*. For convenience we shall refer to these as upper and lower nodes.

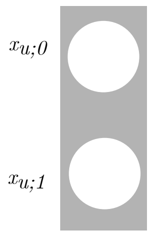

Figure 1: LP variable node structure.

For each object from graph G we define its correspondent LP variable node:

$$\forall v \in V : \exists v' \in V' : v' = \{(v, 0), (v, 1)\}. \tag{18}$$

An LP variable edge is a structure that is used to represent a connection between two different LP variable nodes. It is represented by four *edges*: $x_{uv;00}$, $x_{uv;01}$, $x_{uv;10}$, $x_{uv;11}$. We shall refer to the parts of an LP variable edge as upper-upper, upper-lower, lower-upper or lower-lower edge as shown on the image below. Upper-upper and lower-lower edge shall be further called *horizontal* edges and the remaining ones shall be called *diagonal*.

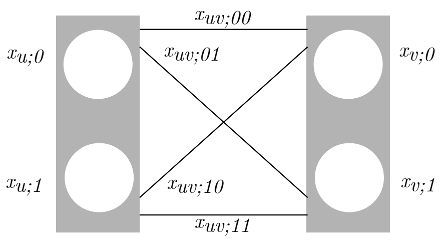

Figure 2: LP variable edge structure.

For each object pair from G we define its correspondent LP variable node:

$$\forall e \in E : \exists e' \in E' : e' = \{(e, 00), (e, 01), (e, 10), (e, 11)\}. \tag{19}$$

Both node and edge hold two numeric values: *variable* and *coefficient.* The coefficient is a value we get from data set. The variable is going to be used in our simplex algorithm. We define three types of variables:

- *non-basic variable*

- *zero basic variable* (basic variable with zero value)
- *unitary basic variable* (basic variable with unit value)

We use the following colors for graphic representation of different types of variables:

🔴 *non-basic variable*

🔵 *zero basic variable*

⚫ *unitary basic variable*

As we mentioned before, during the simplex algorithm, our variables can only have values {0, 1} because we work with only submodular problem instances. For every pair of nodes that belong to the same object the following condition from LP relaxation must be valid (16a):

$$x_{u;0} + x_{u;1} = 1 \tag{20}$$

Thus we can assume that in a submodular problem instance every LP variable node has exactly one unitary basic node. The remaining node can be either zero basic or non-basic.

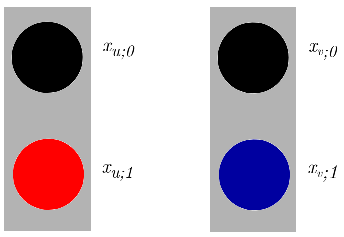

Figure 3: Possible types of LP variable nodes in graph.

As seen in the LP relaxation , horizontal edges are not present in any of the linear program constraints. Thus we assign zero coefficients for every horizontal edge in the graph before starting simplex algorithm. We shall later work with only diagonal parts of every LP edge. For simplicity, we shall refer to every LP variable edge as it has only two parts: upper-lower and lower-upper diagonal edges.
For every LP variable edge that connects two distinct LP variable nodes u and v the following condition must be valid (16b):

$$x_{u;0} + x_{uv;10} = x_{v;0} + x_{uv;01} \tag{21}$$

As mentioned above, every LP variable node necessarily consists of unitary basic node. The remaining node holds zero value. That implies that there are two possible solutions for the equation above:

- $x_{u;0} = x_{v;0}$

In that case we get the following equation:

$$x_{uv;10} = x_{uv;01} \tag{22}$$

As our task is to minimize value of the objective function we derive that

$$x_{uv;10} = x_{uv;01} = 0 \tag{23}$$

In addition to this, at least one of the edges $x_{uv;10}$ and $x_{uv;01}$ must be non-basic [6].

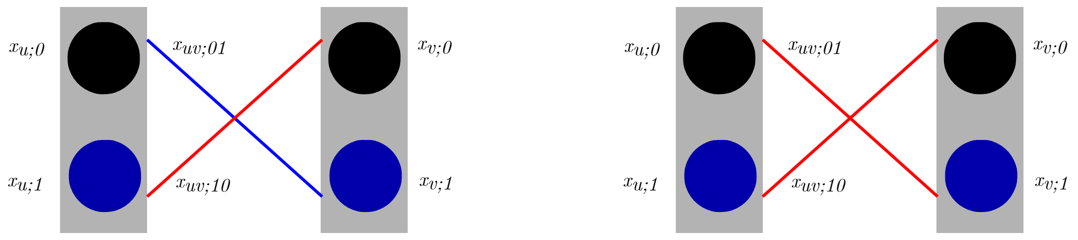

Figure 4: Possible types of diagonal edges.

- $x_{u;0} \neq x_{v;0}$

Let us assume that $x_{u;0} = 1$ and $x_{v;0} = 0$. Then we get the following equation for edges:

$$x_{uv;10} = x_{uv;01} + 1 \tag{24}$$

Because of this we derive $x_{uv;10} = 1$ and $x_{uv;01} = 0$. This means that every LP edge should necessarily have at least one non-basic diagonal edge. The other diagonal edge can either be non-basic or basic with any possible value.

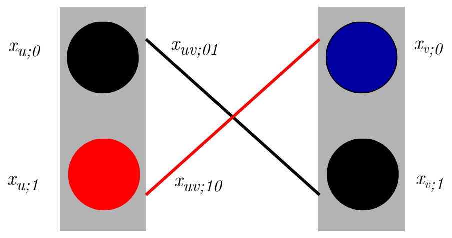

Figure 5: Possible types of diagonal edges.

3.3 Graph structure

Now that we have derived the structure of LP nodes and LP edges in our graph we should derive how can LP nodes be connected. As mentioned in Section 3.2, two distinct LP nodes are connected via an LP edge.
The unitary basic diagonal edge should only be used to connect two basic unitary nodes. On the contrary, zero basic diagonal edge should only connect different types of nodes. Non-basic edge should connect either different types of nodes parts or alternatively two non-basic nodes. Thus, there are three possible types of connections between LP nodes in graph:

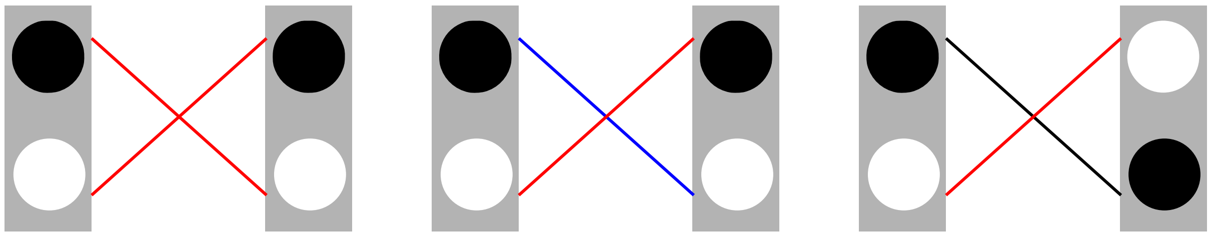

Figure 6: Types of connections between LP nodes in graph. White color represents non-basic or zero basic variable.

An LP node with non-basic node is called *root object* or just *root*. Otherwise, we call an LP node *complex node*. We define an LP edge with two non-basic diagonal parts *tree edge* and we refer to any other type of LP edge as *non-tree edge*. We define a *tree* as a structure that consists of a single root, a set of nodes and a set of tree edges (we note that both sets can be empty).

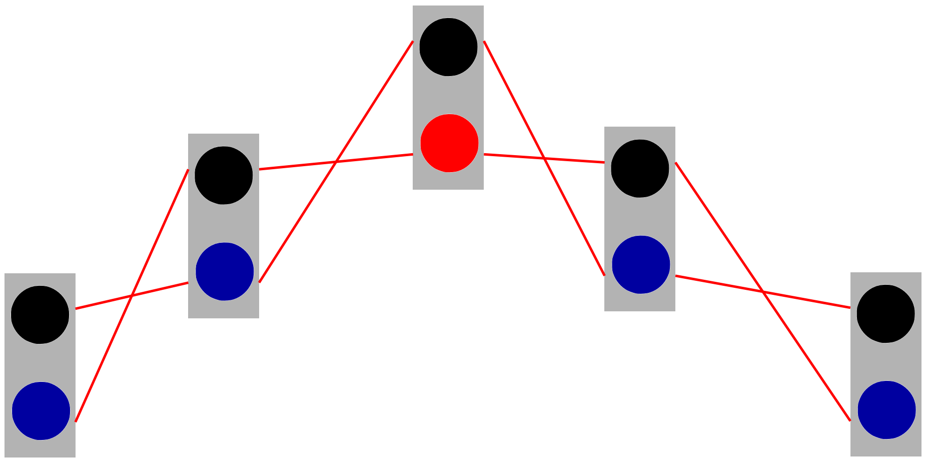

Figure 7: Example of a tree.

The goal of minimizing discrete energy by using the simplex algorithm can be interpreted as a task of splitting the LP relaxation graph into a set of disjunctive trees, where all non-basic variable have non-negative coefficients. The trees can be connected with each other by non-tree edges. Thus, our goal is to find a set of trees and a set of non-tree edges that represent the minimal value of discrete energy.

4 Algorithm

4.1 Terminology

Throughout the description of the algorithm we use the following terms:

- The term *node* is used to refer to parts of LP nodes.
- We define *weighted element* as node or edge that belongs to LP relaxation graph.
- In this Section we refer to every LP node as *complex node*.
- The term *root* is used to describe a complex node with a non-basic part.
- We say that two complex nodes have *equal orientation* or are *equally oriented* if either both have unitary basic upper node or both have unitary basic lower node. Otherwise, the complex nodes are *differently oriented*.

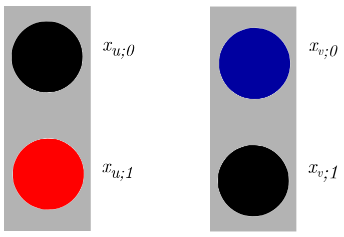

Figure 8: Example of two differently oriented complex nodes.

- Unlike tree edges, non-tree edges do not necessarily connect nodes that belong to the same tree or subtree. We refer to a non-tree edge that connects nodes that do not belong to the same subtree as *outer non-tree edge*.

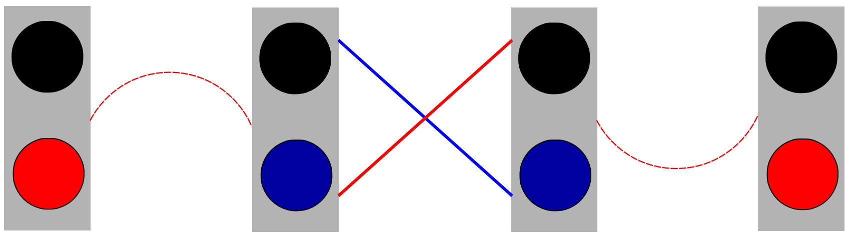

Figure 9: Example of an outer non-tree edge. Red dashed lines represent tree branches.

We call the opposite an *inner non-tree edge*. We note that this term applies to edges that connects nodes from the same tree or subtree.

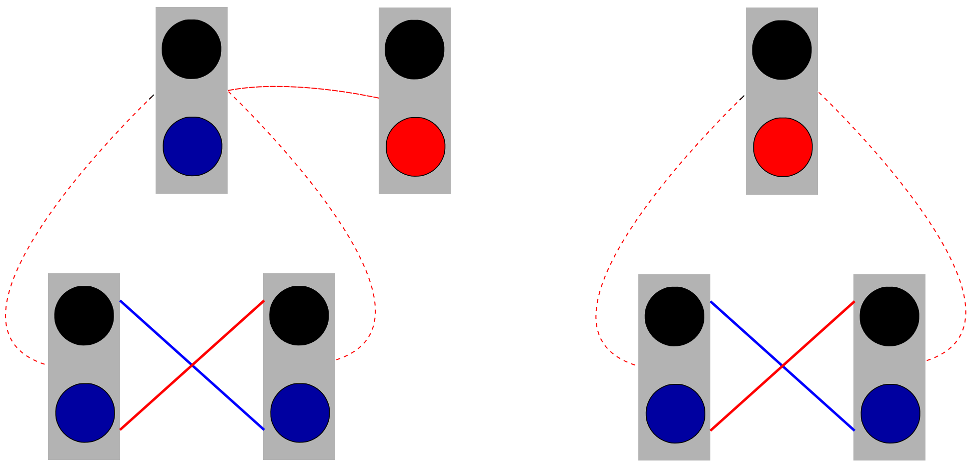
Figure 10: Example of an inner non-tree edge between nodes from the same tree or subtree. Red dashed lines represent tree branches.

Note that unitary basic inner non-tree edges cannot possibly exist because every two nodes in a tree are equally oriented.

- For every inner non-tree edge we define a term *inner tree* which stands for the tree to which nodes connected by given edge belong. Similarly, we define a term *outer tree* that is used for trees connected by outer non-tree edges. We say that every outer edge has two inner trees.
- Though the LP relaxation graph is undirected by definition, we add some artificial direction parameters for edges to simplify processing the graph:
  1. We say, that a tree edge between complex nodes x and y *leads from* x *to* a complex node y if y is located farther from the root than x (i.e. y has greater depth than x).

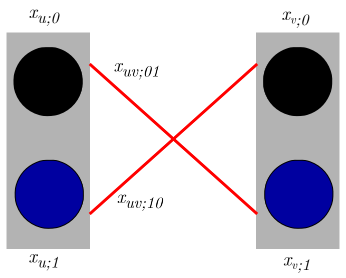

Figure 11: Tree edge direction: edges $x_{uv;01}$ and $x_{uv;01}$ lead to the complex node x_v.

  2. we say, that a zero basic non-tree edge between complex nodes x and y *leads to* y *from* x if the upper diagonal edge between x and y is zero basic.

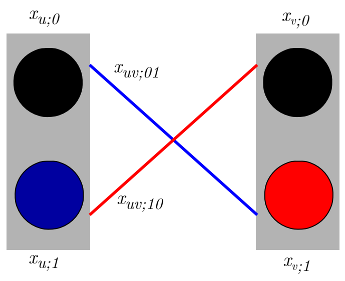

Figure 12: Non-tree edge direction: edges $x_{uv;01}$ and $x_{uv;01}$ lead to the complex node x_v.

3. We do not define any artificial direction parameters for unitary basic edges.

- A *tree recolor* is the following tree processing:

  1. For every complex node we swap the variable types of complex node nodes.
  2. Every outer zero basic non-tree edge becomes unitary basic.
  3. Every outer unitary basic non-tree edge becomes zero basic.

  We do not have to change inner non-tree edges during recolor, as not changing their variables does not violate the graph structure.

- During every simplex iteration at least one weighted element coefficient is altered. We call changing coefficients of every weighted element affected during simplex iteration a *coefficient update*.

- We call a sequence of complex nodes and tree edges that leads from complex node to its tree root a *root branch*. Observe that the smallest root branch is a tree root itself.

- As we may often process root branches that lead from nodes connected by outer non-tree edges, we create definitions for them. We call the branch leading from starting node *left root branch* and we refer to the second branch as *right root branch*.

- We refer to absolute value of leaving variable coefficient as *balancing value*.

- We shall use the following symbols for graphic representation of entering and leaving variables:

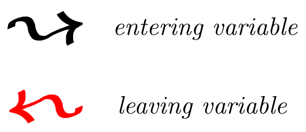

Figure 13: Symbols used for graphic representation of entering / leaving variables.

4.2 Initial graph state

Before running the simplex algorithm we need to transform the given graph into *initial state* defined as a set of V single node trees and E non-tree edges, where V and E are numbers of nodes and edges respectively. In other words, we want every node to be a root of a tree and we want every tree have exactly one node which is a tree root. Consequently, every complex edge has to be non-tree as every edge connects two different trees with each other. So, in initial state there can be only one type of connection between complex nodes:

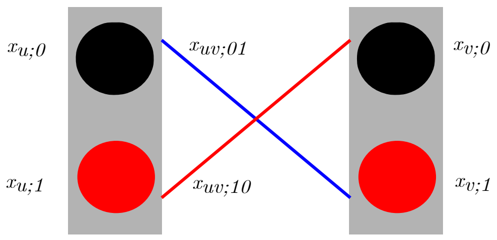

Figure 14: Connections between objects in initial graph state.

We also want to perform *reparametrization* on the graph, meaning that we want to set coefficients of all basic variables to zero. Firstly, we need to set coefficients of all horizontal edges to zero.

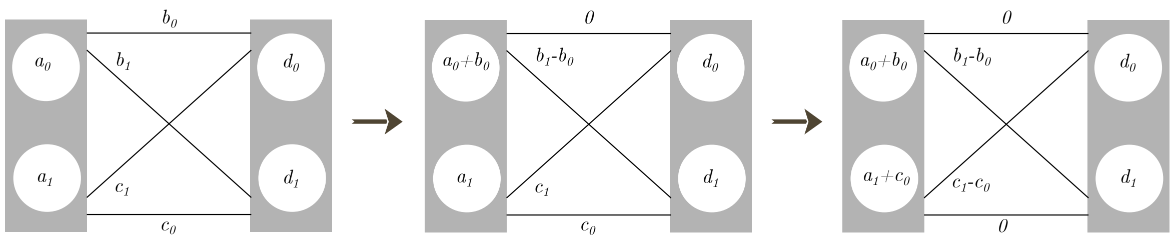

Figure 15: Reparametrization, step 1: setting horizontal edge coefficient to zero

After performing this operation, we set coefficient of every upper-lower edge to zero.

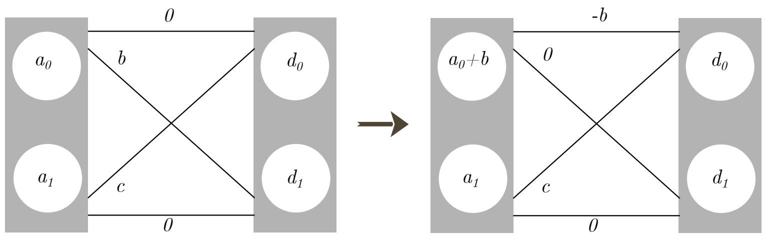

Figure 16: Reparametrization, step 2: setting diagonal edge coefficient to zero.

Then again we need to set upper horizontal edge coefficient to zero as it might have been possibly changed during previous step.

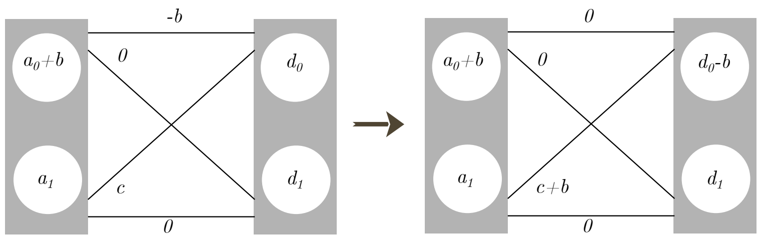

Figure 17: Reparametrization, step 3: setting horizontal edge coefficient to zero.

Finally, we set complex node variables to non-basic and basic with zero value and we set coefficient of every complex node basic part to zero.

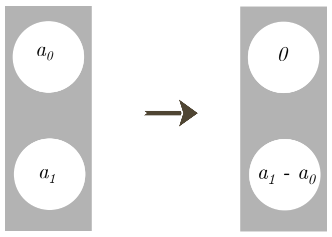

Figure 18: Reparametrization, step 4: altering complex node coefficients.

As a result of these transformations our graph becomes a set of V root nodes and E non-tree edges. Moreover, no non-tree edge contains unitary basic variables because we want to avoid processing unitary basic edges during first step.

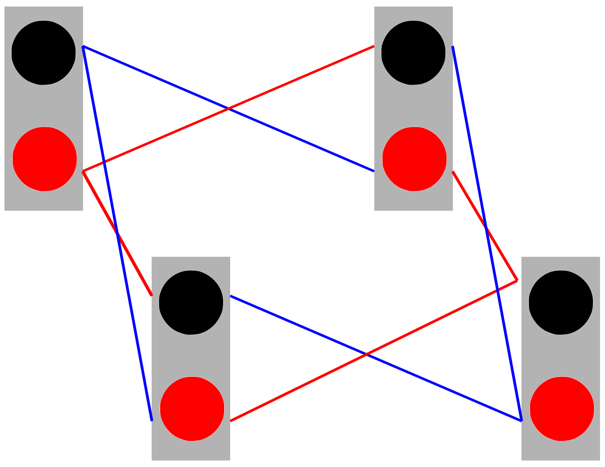

Figure 19: Example of graph structure in initial state.

4.3 Transition to simplex method

As we mentioned before, we need to transform basis into standard in order to perform simplex algorithm [11]. We should therefore rewrite constraints from LP relaxation into m equations where m is a number of basic variables. Moreover, every basic variable is present in exactly one equation and every equation contains precisely one basic variable. Thus, we get m columns in simplex tableau that represent a standard basis. We refer

to these m equations as *standard constraints*.
We say that basic variable x is *dependent* on a non-basic variable y if a standard constraint for x contains y multiplied by non-zero coefficient [6]. We expand the definition of basic variable dependency by saying that x is *positively dependent* or *has positive dependency* on y if coefficient of y is positive, and we say x is *negatively dependent* on y if y is multiplied by a negative number.
We observe that we do not need to modify LP constraints (16a) for unitary basic nodes, because they contain exactly one basic variable. However, the approach to constructing standard constraints is not as straightforward for other types of variables. Let us describe the way to represent every type of variable by standard constraints:

- **basic nodes**
  We shall describe an approach for upper nodes, as the standard constraints for lower nodes can be derived from these using the original LP constraints (16a). From the definition of tree we can assume that every node belongs to exactly one tree. Therefore, there is a branch leading from the node to the tree root. As this branch consists of tree edges, which are non-basic, we can derive the standard constraint for every basic node in it.
  Let us start with the following example of the shortest possible two-node root branch:

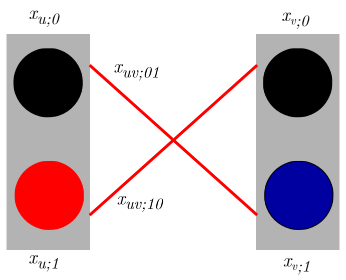

Figure 20: Two-node root branch.

To derive an equation for $x_{v;0}$ we modify LP constraint for nodes and diagonal edges (16b) by substituting $x_{u;0}$ for $1 - x_{u;1}$:

$$x_{v;0} + x_{uv;01} - x_{uv;10} + x_{u;1} = 1 \tag{25}$$

The constraint for $x_{v;1}$ is derived from the previous one by substituting $x_{v;0}$ for $1 - x_{v;1}$:

$$x_{v;1} - x_{uv;01} + x_{uv;10} - x_{u;1} = 0 \tag{26}$$

Let us derive the constraint for a node in a three-node branch:

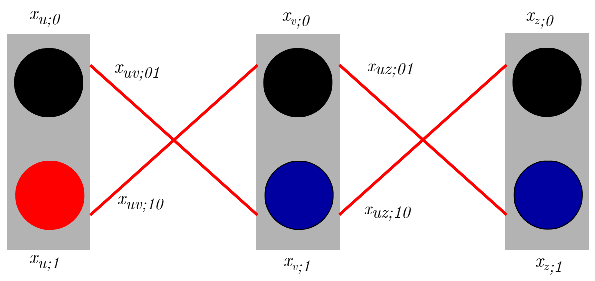

Figure 21: Three-node root branch.

We derive the standard constraint from this LP constraint equation:

$$x_{v;0} + x_{vz;10} = x_{z;0} + x_{vz;01} \tag{27}$$

We rewrite the value of $x_{v;0}$ variable from the previously found standard constraint (25). Thus, we get the following equation for $x_{z;0}$

$$x_{u;1} + x_{uv;01} - x_{uv;10} + x_{z;0} + x_{vz;01} - x_{vz;10} = 1 \tag{28}$$

Afterwards, we create a constraint for $x_{z;0}$:

$$-x_{u;1} - x_{uv;01} + x_{uv;10} + x_{z;1} - x_{vz;01} + x_{vz;10} = 0 \tag{29}$$

By using mathematical induction we can derive a general standard constraint for every $x_{v;0}$ and $x_{v;1}$ assuming that $x_{v;0}$ is a non-basic root node:

$$x_{v;0} + \sum x_{uv;01} - \sum x_{uv;10} + x_{u;1} = 1 \tag{30}$$

$$x_{v;1} - \sum x_{uv;01} + \sum x_{uv;10} - x_{u;1} = 0 \tag{31}$$

Therefore, we note that every node is dependent only on its tree root and tree edges from the root branch where it belongs.

- **non-tree edges**
  Let us consider a zero basic non-tree edge and its left and right branches. If both branches have a common complex node, then we can assume from (30), (31) and (16b) that the edge is dependent on both branches parts prior to their common part as well as a non-basic edge it is paired with. In this case, the standard constraint for every zero basic non-tree edge $x_{st;01}$ is in the following form:

$$x_{st;01} - x_{st;10} - \sum x_{uv;01} + \sum x_{uv;10} + \sum x_{wz;01} - \sum x_{wz;10} = 0, \tag{32}$$

  where $\{x_{uv;01}, x_{uv;10}\}$ belong to the right branch and $\{x_{wz;01}, x_{wz;10}\}$ are part of the left branch.
  Should there be no common complex node between left and right branches, then the zero basic edge is dependent on both branches and their roots and its diagonal non-basic edge.
  Thus, a standard constraint for the zero basic edge $x_{st;01}$ can be written as:

$$x_{st;01} - x_{st;10} - x_{a;1} - \sum x_{uv;01} + \sum x_{uv;10} + x_{b;1} + \sum x_{wz;01} - \sum x_{wz;10} = 0, \tag{33}$$

  where $\{x_{a;1}, x_{uv;01}, x_{uv;10}\}$ belong to the right branch and $\{x_{b;1}, x_{wz;01}, x_{wz;10}\}$ are part of the left branch.
  We have described the rule to derive standard constraints for zero basic edge. We do not provide a description of the rule for unitary basic edge as we opt not to work with them in our implementation. The reason for it is that unitary basic edges are quite challenging to process unlike other variables, and, as for every unitary basic edge the pivotal value does not equal zero, then we can either find a variable with pivotal value zero, or choose a variable with pivotal value one which is less challenging to process.

4.4 Root step

We define *root step* as an iteration of simplex algorithm where the entering variable is a non-basic node. It follows that this node has to be a part of a root.

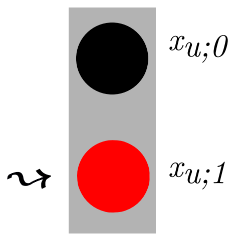

Figure 22: Root leaving variable.

We should now choose a suitable leaving variable which will become basic during simplex iteration. It is required that the leaving variable should have positive dependency on entering variable because of simplex tableau iteration rules.
We can derive two types of leaving variables from previous equations:

- *outer basic edge*
  As we mentioned before, any inner non-tree edge is not dependent on the root of its inner tree. Basic outer non-tree edges, however, depend on roots from both their outer trees, hence they can become a leaving variable. Both unitary and zero basic can be chosen to become a leaving variable. We, however, opt to avoid selecting unitary ones in our algorithm as processing them can result in challenging graph processing during simplex iteration.
  The search for outer zero basic edge is performed by the breadth-first search (BFS) algorithm. We need, however, to run a tree traversal beforehand during which we put a mark on the tree root and all its complex nodes. After that we perform a BFS tree traversal in order to find a zero basic edge that connects a marked complex node with unmarked one. If such an edge is found, then it becomes non-basic and the entering node becomes a zero basic variable.

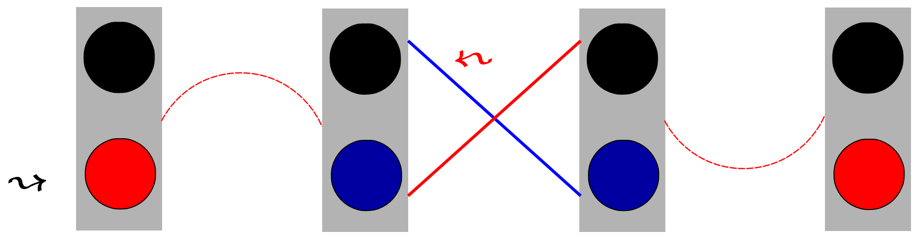

Figure 23: Entering and leaving variables.

Then we perform a coefficient update on the tree branches leading from both nodes connected by the found edge. For the left branch, we add balancing value to every upper diagonal edge coefficient and subtract it from every lower diagonal edge coefficient, after that, we add balancing value to non-basic root node of the tree. The right branch coefficient update is performed contrariwise.

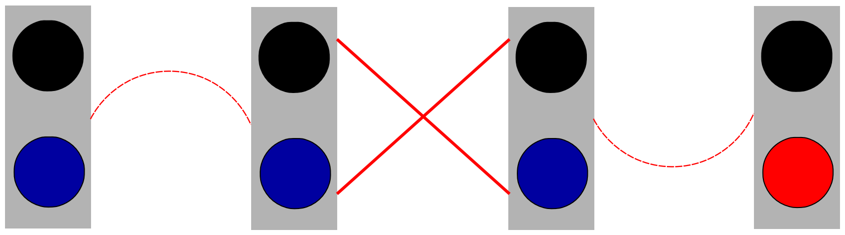
Figure 24: Result of simplex iteration.

- *unitary basic node that belongs to the tree with current root*
  Every complex node node is dependent on its tree root, namely, on its basic node. Particularly, every unitary basic node has a positive dependency on its root non-basic node. It follows therefore that every unitary basic node from any complex node belonging to the tree with current root can become a leaving variable. For simplicity we always choose a basic part of the current root to leave the basis.

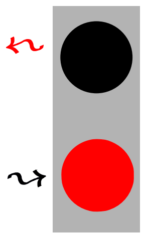
Figure 25: Entering and leaving variables.

As the tree root consists of one basic and one non-basic node, the equation for leaving node in simplex tableau consists of only its root parts. Therefore, we do not need to perform a coefficient update to any weighted element but the root itself. The rule is to add the balancing value to both root node coefficients.
We observe, however, that an invalid tree structure emerges as a result of simplex iteration. Therefore we need to perform a tree recolor in order to create a valid graph.

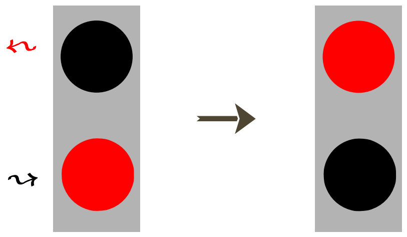
Figure 26: Result of simplex iteration.

Note that if a suitable outer basic edge exists, then it will be chosen as it has greater pivotal value in simplex tableau. That implies the need to search for an edge that can become a leaving variable. If we find it, we perform basis update as well as variables and coefficients change. If we do not, we should take some of the unitary basic nodes that belong to the current tree. We can observe that the easiest solution is to take unitary basic part of current root and mark it as leaving variable. Thus, we opt to always do so in our implementation.

The algorithm for root simplex iteration is the following:

1. Traverse the tree and mark all nodes that belong to it.
2. Perform a search for a zero basic outer non-tree edge that is positively dependent on the root.
3. If the edge is found, make it leave the basis, perform coefficient update for left and right edge branches.
4. If the edge is not found, then take the basic part of current root as a leaving variable, update coefficients only for the root and perform a tree recolor.
5. Make the non-basic root node enter the basis.

4.5 Edge step

We define *edge step* as an iteration of simplex algorithm where non-basic edge is an entering variable. For submodular problem instances entering edge has to be non-tree, thus we have two possible outcomes:

- *lower diagonal edge*
  As the entering edge is necessarily a tree edge, then it necessarily leads to a tree complex node. We observe, that a zero basic node in this complex node is a suitable entering variable as it has positive dependency on entering variable and its pivotal value is zero. Besides, we cannot possibly choose another entering variable with lesser pivotal value, as non-negative pivotal values in LP relaxation graph are in $\{0, 1\}$. Thus, for simplicity, we choose a zero basic node from a tree complex node to which chosen entering edge leads to be the leaving variable.

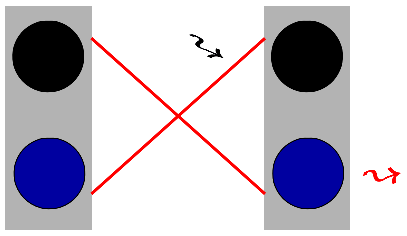

Figure 27: Entering and leaving variables.

  After choosing a suitable leaving variable, we perform a coefficient update on a left root branch of the entering edge: we add the balancing value to every upper diagonal edge coefficient and subtract it from every lower diagonal edge coefficient. As for the tree root, we increase its non-basic node coefficient by the balancing value. Then we set entering edge to be zero basic and the leaving node to be non-basic. We observe, that after simplex iteration there is a new tree in the graph.

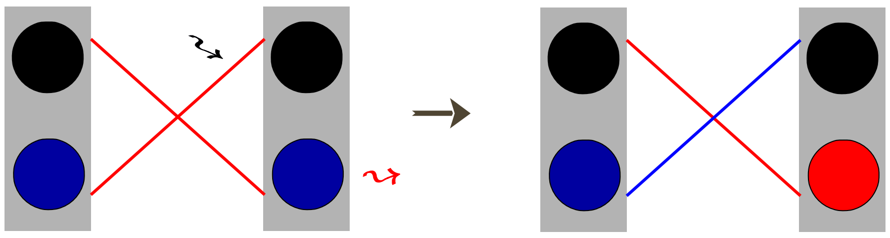
Figure 28: Result of simplex iteration.

We observe, that this particular simplex iteration can be interpreted as splitting the tree in a graph into two new trees which are connected by a zero basic outer tree edge.

- *upper diagonal edge*
  As opposed to lower diagonal edge, upper diagonal edge has negative dependency on zero basic node and positive dependency on unitary basic node. We cannot, however, choose the unitary basic node as leaving variable immediately, because its pivotal value is not equal to one, thus, it is not guaranteed, that there is no lesser pivotal value in the graph. Therefore, we have to search for suitable leaving variables with unitary pivotal value. If our search is successful, then the found variable becomes leaving, if it does not return any suitable variables, then we set the right adjacent unitary basic node to be the entering variable.
  Let us inspect all possible edges and complex nodes to choose as leaving variables. We observe, that a leaving variable should either belong to the same tree as the entering variable, or, alternatively, be an non-tree edge. As mentioned in previous paragraph, we cannot consider nodes belonging to the subtree to become leaving variables. Moreover, nodes from the root branch are not dependent on the chosen entering variable. That leaves non-tree edges as the only possibility to become leaving variable with unitary pivotal value. We can assume, that a suitable non-tree edge should be adjacent with a complex node from the subtree, otherwise, it will not be dependent on the entering edge. In addition to this, any inner non-tree for the subtree has no dependency on entering variable. Thus, we assume, that the leaving variable necessarily belongs to the set of outer non-tree edges for entering edge's subtree. The search algorithm for the leaving edge with unitary pivotal value is similar to the one we use during the root iteration.
  If we manage to find a suitable leaving edge, then we perform a coefficient update on its left and right branches.

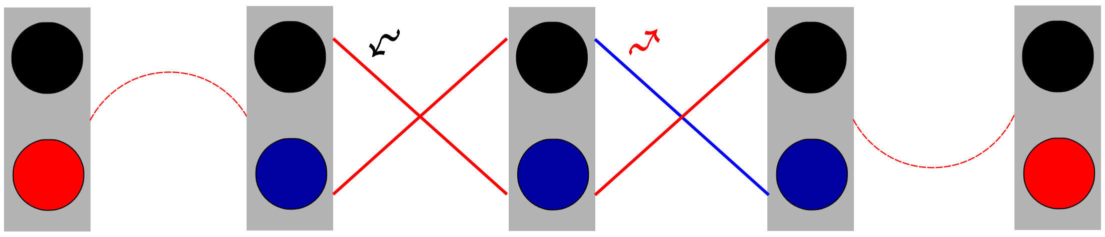
Figure 29: Entering and leaving variables.

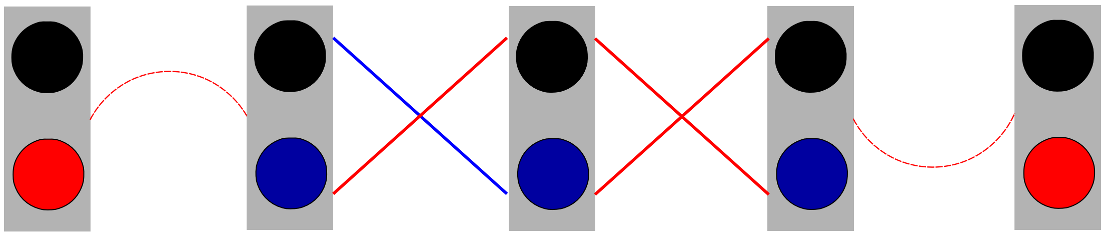
Figure 30: Result of simplex iterations.

If the suitable edge does not exist, then we choose the right adjacent unitary node to become a leaving variable and perform a coefficient update on its root branch.

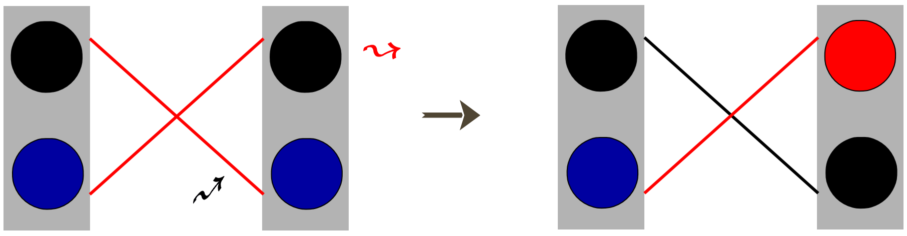
Figure 31: Result of simplex iteration.

The algorithm for edge simplex iteration is the following:

1. Determine the type of edge: either upper diagonal or lower diagonal.
2. If the entering edge is lower diagonal, then the zero basic node the edge leads to becomes the leaving variable.
3. If the entering edge is upper diagonal, then we perform a search for an outer non-tree edge which is positively dependent on entering edge. If we find such an edge, it becomes the leaving variable, otherwise, the unitary basic node the edge leads to becomes the leaving variable.

4.6 General algorithm

To find the minimal value of discrete energy we perform simplex iterations until we reach a state where the value of objective function cannot decrease, i.e. in which all non-basic variables have non-negative coefficient [11]. The general graph algorithm consists of 5 steps: *entering variable selection*, *leaving variable selection*, *iteration*, *basis update* and *termination check* [6].

- **Entering variable selection** is performed via *priority queue* which is filled with non-basic variables with negative coefficients. At the beginning of every simplex iteration we choose the first element from the queue.
- Both **leaving variable selection** and **iteration** depend on chosen entering variable. We determine the type of entering variable and perform either the root or edge simplex step which combines selection of suitable leaving variable and simplex iteration.

- **Basis update** is renewing the priority queue after iteration is performed. We want the queue to represent all non-basic variables with negative coefficients in current graph.
- **Termination check** determines if the graph state representing the minimal value of objective function is already found. This state is represented by a graph where every non-basic variable has a non-negative coefficient, thus, the priority queue is empty. In this case, we stop the algorithm.

After finishing simplex iterations, we determine the minimal value of objective function. We consider only unitary basic variables and their coefficients, as both zero basic and non-basic ones have zero value in solution, so minimal value is basically a sum of coefficients of all unitary basic values.
To begin with, we set initial minimal value to 0. As every complex node consists of one unitary basic node, we add all coefficients of unitary basic nodes to the minimal value. As for the edges, we consider edges connecting two unitary basic nodes to be also unitary basic, other edges are either zero basic or non-basic, thus, they are not part of the solution. Therefore, we add all coefficients of unitary basic edges to the minimal value. Note that we include horizontal edges as unitary basic variables in the solution as long as they connect two unitary basic nodes, hence we add their coefficients to the minimal value.

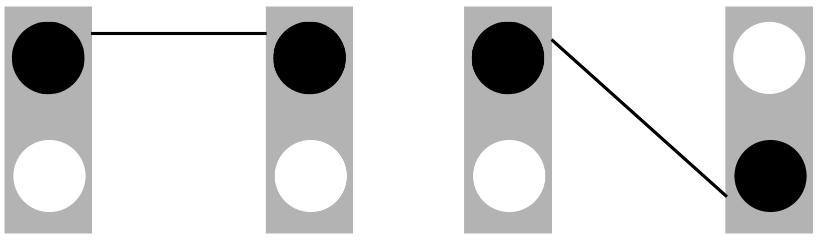

Figure 32: Example of edges between unitary basic nodes. Such edges are treated as unitary basic variables, therefore, are a part of the solution.

5 Testing

We have implemented the graph-based simplex algorithm in C++. The source code was written and compiled in Microsoft Visual Studio Community 2017. The tests were run on a notebook with OS Windows 8.1 64-bit, processor Intel Core i3-5010U 2.10GHz and 4GB RAM memory.

5.1 Generated random test data

The initial testing was performed on artificially generated datasets. Each dataset represents a $m \times m$ square grid. The datasets are text files in specific format (an example can be found in appendix A). Each text representation of a grid contains initial weights for every node and edge. The node weights form a normal distribution $N(0, a^2)$. Weights for edges form a normal distribution $|N(0, b^2)|$. To ensure that our instance is submodular we check if it satisfies the conditions presented in Section 2.1.
We randomly generated 3 groups of 100 datasets with number of complex nodes in range from 4 to 5000. These 3 groups are represented by the following parameters:

- $a \gg b$
  This test case represents a graph, where absolute values of edges coefficients are significantly lower that absolute value of nodes coefficients. Therefore, we take $b = 0$ so that all edges have zero coefficients. We note that in this case optimal solution is represented by unitary basic nodes as every edge, even unitary basic, has coefficient zero, and, thus, does not change optimal solution in any way.
  The running time is represented by the following graph:

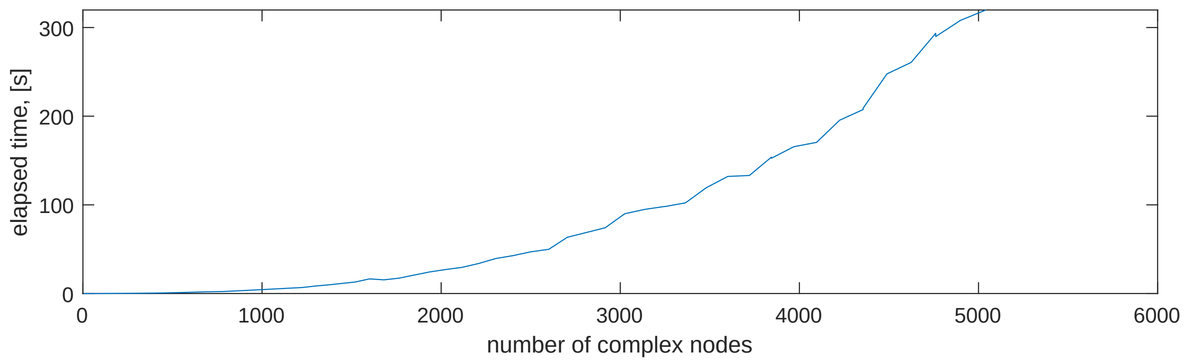

Figure 33: Running time for test data with $a \gg b$.

- $a = b$
  The running time is represented by the following graph:

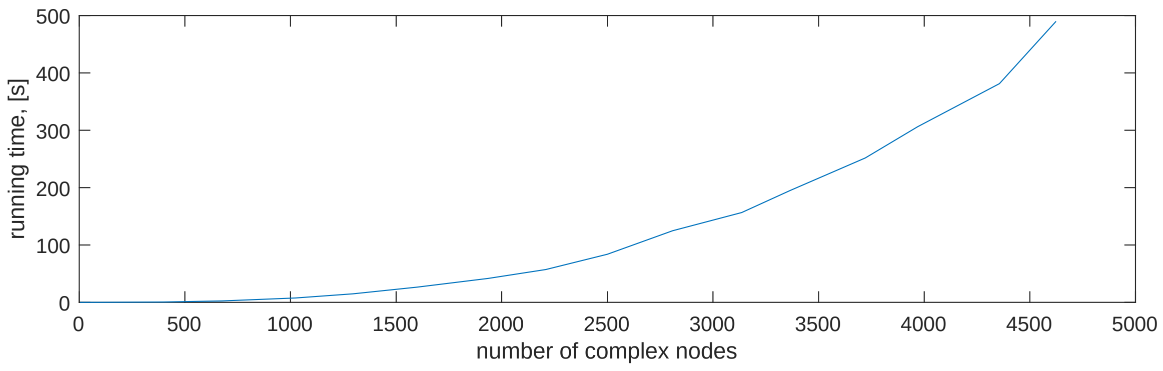

Figure 34: Running time for test data with $a = b$.

  The number of simplex iterations depending of number of complex nodes is represented by the following graph:

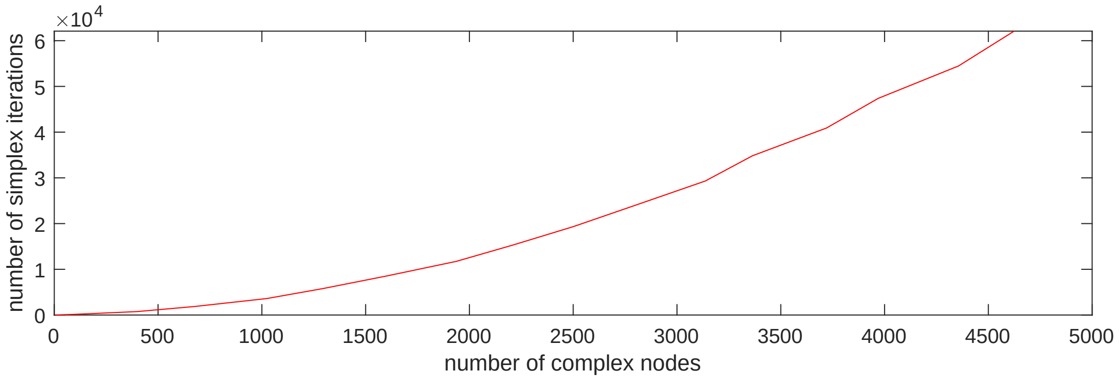

Figure 35: Number of simplex iterations for test data with $a = b$.

- $b = 2.5 \times a$
  Because absolute values of edges coefficients are greater than these of nodes, the graph representation of minimal energy might be a set of larger trees, thus we expect our algorithm to be slower. The running time is represented by the following graph:

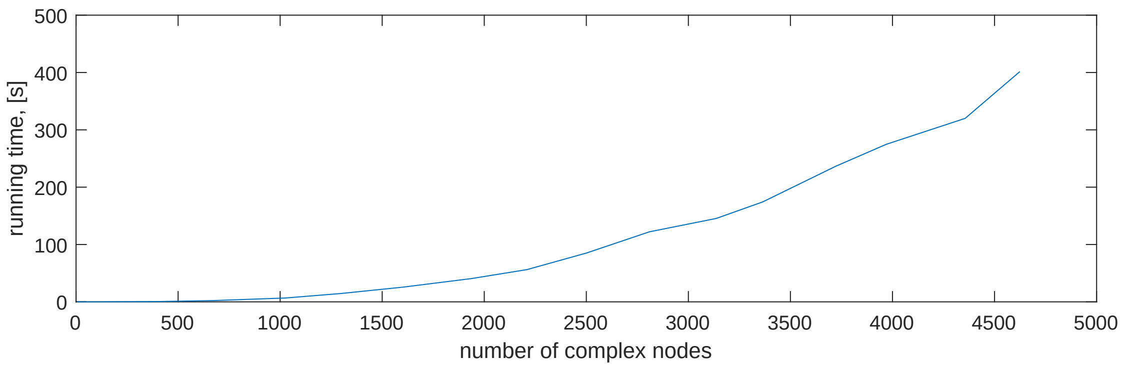

Figure 36: Running time for test data with $b = 2.5 \times a$.

The number of simplex iterations depending of number of complex nodes is represented by the following graph:

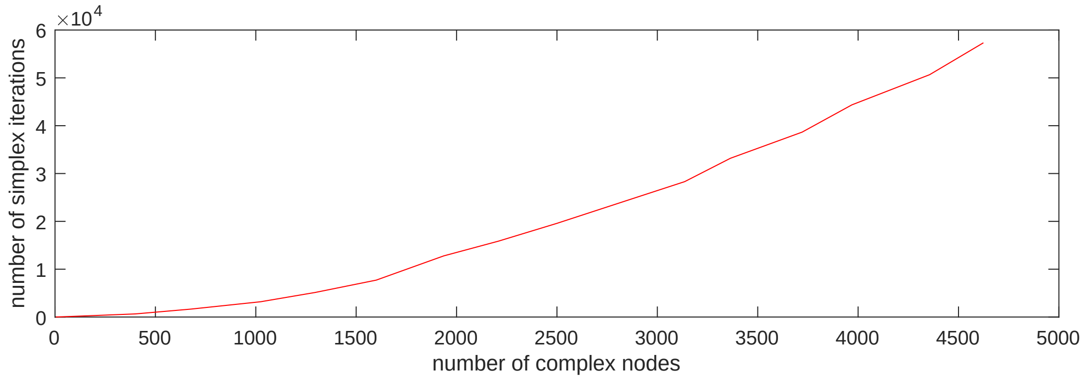

Figure 37: Number of simplex iterations for test data with $b = 2.5 \times a$.

5.2 Real test data

As an example of real life submodular data we have chosen the submodular surface-fitting instance *LB07-bunny* from [1].
As the instance presents a relatively large grid (the graph contains around 800000 complex nodes) our program did not manage to return an optimal solution during a sensible amount of time (it took 2.5 hours to finish 10 iterations). The progressive result for minimal energy that we got after 10 iterations was 1.54055×10^9.
The optimal value for this dataset returned by the program presented in [6] is 961163.

5.3 Analysis of results

For every test data instances we have compared results returned by our implementation with the results returned by the program presented in [6]. During testing we have encountered an issue, that our implementation does not always return the same optimal result as the reference implementation. Our program consistently returned correct minimal value only for test data with $a \gg b$. For the other datasets, the results were consistently correct only for smaller instances (up to 5×5 grid, i.e. 25 complex nodes). Another issue that we have encountered is running time of our program. As it can be seen from the figures in Section 5.1, the running time for instances with more than 2500 complex nodes exceeds 120 seconds, thus, it is not effective on large-scale data.

6 Conclusion

In this thesis we have proposed a graph-based simplex algorithm for solving submodular instances of the binary discrete energy minimization problem. As proved in [6], the general graph-based simplex algorithm is efficient for some instances. The idea to implement a graph-based algorithm for submodular data instances has some potential as it works with reduced number of variables and therefore can be beneficial for solving submodular instances.
However, we cannot say we present a successful implementation as we have not achieved proper results for all of the datasets. We believe, that the core of this problem is an ineffective representation of LP relaxation graph.
Overall, we believe that the idea to implement a graph-based simplex method for solving submodular problem instances has great potential due to smaller amount of variables and constraints in LP relaxation. We can guess that altering the data structure representing the LP relaxation graph can lead to reducing the number of traversed complex nodes in the graph. Other possible improvements include a usage of a different pivotal rule.

References

[1] Max-flow problem instances in vision. `http://vision.csd.uwo.ca/maxflow-data`, accessed on 15.02.2018.

[2] Endre Boros and Peter L. Hammer. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002.

[3] G.B. Dantzig and M.N. Thapa. *Linear Programming 1: Introduction.* Springer, 1997.

[4] Boris Flach and Michail I. Schlesinger. A class of solvable consistent labeling problems. In Francesc J. Ferri, José Manuel Iñesta Quereda, Adnan Amin, and Pavel Pudil, editors, *Advances in Pattern Recognition, Joint IAPR International Workshops SSPR 2000 and SPR 2000, Alicante, Spain, August 30 - September 1, 2000, Proceedings*, volume 1876 of *Lecture Notes in Computer Science*, pages 462–471. Springer, 2000.

[5] Vladimir Kolmogorov and Carsten Rother. Minimizing nonsubmodular functions with graph cuts-a review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(7):1274–1279, 2007.

[6] Daniel Průša. Graph-based simplex method for pairwise energy minimization with binary variables. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 475–483. IEEE Computer Society, 2015.

[7] Daniel Průša and Tomás Werner. LP relaxation of the potts labeling problem is as hard as any linear program. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 39(7):1469–1475, 2017.

[8] Carsten Rother, Vladimir Kolmogorov, Victor S. Lempitsky, and Martin Szummer. Optimizing binary MRFs via extended roof duality. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[9] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[10] Tomáš Werner. A linear programming approach to max-sum problem: A review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, July 2007.

[11] Tomáš Werner. Optimalizace, 2018. `https://cw.fel.cvut.cz/old/_media/courses/a4b33opt/opt.pdf`, in Czech, accessed on 15.02.2018.

A Data format

An example of text representation of a 2×2 grid:

```
nodes=4
edges=4
labels = 2
type = double

n 0 -1.31744 1.21077
n 1 -16.8425 4.14585
n 2 -1.92828 1.47341
n 3 -7.4515 2.68736
e 0 1 1.17457 1.05547 0.0102154 -0.546841
e 0 2 -1.04944 -0.625276 1.48596 0.660682
e 1 3 -0.829081 -0.888707 -0.539781 -2.55912
e 2 3 -0.482589 1.01922 -0.628956 0.339587
```

B Contents of CD

```
|---exe
|   simplex_graph.exe // executable file for implemented program
|---src
|       graph_definitions.h
|       graph_generator.cpp
|       graph_generator.h
|       graph_init.cpp
|       graph_init.h
|       graph_utils.cpp
|       graph_utils.h
|       main.cpp
|
|---test_data
|   // text data instances are located here
|---text
|   BP.pdf // bachelor thesis text
```

C User manual

The executable file can be found on enclosed CD in folder **exe**. In order to test the program on the particular instance, it is needed to type the path to text file to command line. The result shown by the program is the optimal value, number of upper and lower unitary basic nodes in the solution and elapsed time.