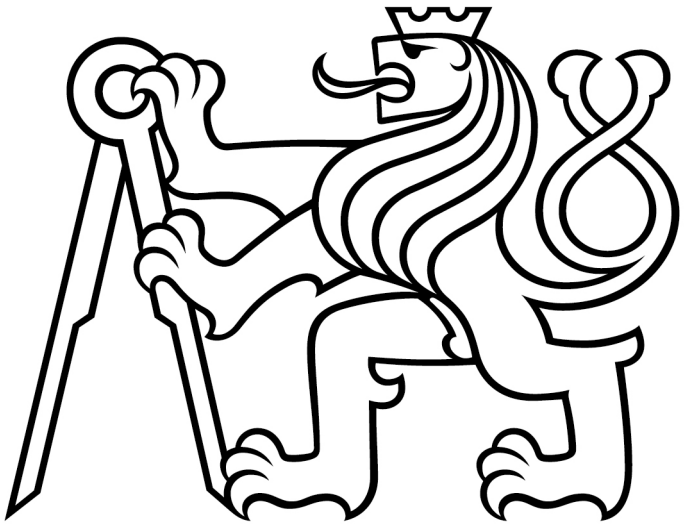
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR THESIS



Dmitrii Mikhailov

Graph-Based Simplex Algorithm for Discrete Energy Minimization

Thesis supervisor: **doc. RNDr. Daniel Průša, PhD**

August, 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Mikhailov Dmitrii** Personal ID number: **483580**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Graph-Based Simplex Algorithm for Discrete Energy Minimization

Bachelor's thesis title in Czech:

Grafový simplexový algoritmus pro minimalizaci diskrétní energie

Guidelines:

1. Get to know the problem of discrete energy minimization [2] and a graph-based simplex algorithm proposed to solve its general [3] and submodular instances [1] in the case of binary variables.
2. With the help of the supervisor, identify deficiencies in [1] (a confusing and incomplete description of the algorithm, implementation not working for larger instances, limited experiments).
3. Formulate your own description of the graph-based simplex algorithm for submodular instances, propose suitable data structures and implement the algorithm in your way.
4. Perform thorough testing of the implementation using artificial instances as well as instances from low-level computer vision [4]. Analyze the obtained results.

Bibliography / sources:

[1] E. Tiuzhina: Application of Simplex Algorithm for Submodular Discrete Energy Minimization with Binary Variables. Czech Technical University in Prague, Bachelor thesis, 2018.
[2] Y. Boykov, V. Kolmogorov: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. IEEE Trans. on Pattern Analysis and Machine Intelligence, 26(9):1124–1137, Sept. 2004.
[3] D. Průša: Graph-based Simplex Method for Pairwise Energy Minimization with Binary Variables. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), Boston, MA, USA, June 7-12, 2015.
[4] Max-flow problem instances in vision: https://vision.cs.uwaterloo.ca/data/maxflow, University of Waterloo, Canada.

Name and workplace of bachelor's thesis supervisor:

doc. 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: **28.12.2020** Deadline for bachelor thesis submission: **13.08.2021**

Assignment valid until: **30.09.2022**

doc. RNDr. Daniel Průša, Ph.D. Supervisor's signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his 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

Author statement for undergraduate thesis

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 13.08.2021 Dmitrii Mikhailov

Acknowledgements

I would like to thank to my supervisor Daniel Pruša, for giving me all necessary information and because he explained me all information, which i couldn't understand. Also i would like to thank to my family, because thay always believe in my possibilities and always give a lot of support. And also i would like to thank to my girlfriend for helping me to keep calm and always be by my side.

Abstract

Discrete energy minimization is a combinatorial problem where the task is to find an optimal assignment of labels to nodes of a given undirected graph. The goal is to find an assignment of minimal total cost. This problem is widely used in low-level computer vision In general, it is an NP-hard problem. It can be solved approximately by algorithms based on a linear programming relaxation. It is possible to apply the simplex algorithm to solve the binary submodular energy minimization. However, the standard implementation of the simplex algorithm is not efficient on large inputs, because the simplex table is too huge. Our goal is to investigate a special, graph-based version of the simplex algorithm which will handle all possible configurations arising during the simplex method iterations.
Keywords : Submodular binary instances, Discrete energy minimization.

Abstract

Minimalizace diskrétní energie je kombinatorický problém, ve kterém hledáme optimální přiřazení labelů pro uzly daného neorientovaného grafu. Cílem je najít přiřazení s minimální celkovou cenou. Tento problém je široce používán například v oblasti počítačového vidění. Obecně se jedná o NP-těžkou úlohu. V případě submodulárních vstupů lze řešit relaxaci problému lineáním programováním. Standartní implementace simplexového algoritmu však není užitečná pro velké vstupy, protože simplexová tabulka by byla příliš velká. Našim cílem bude navrhnout specializovanou simplexovou metodu, která namísto simplexové tabulky pracuje s grafem, a která je schopna efektivně zpracovat všechny možné konfigurace, které se mohou během iterací simplexové metody objevit.
Klíčová slova : Súbmodulární binární instance, Minimalizace diskrétní energie.

Contents

1. Introduction

1.1. The problem of energy minimization.

Discrete energy minimization is a combinatorial problem where the task is to find an optimal assignment of labels to nodes of a given undirected graph. Unary and binary costs of assigning labels are defined for the graph nodes and edges, respectively. The goal is to find an assignment of minimal total cost. This problem is widely used e.g. in low-level computer vision [1] [2]. In general, it is an NP-hard problem. It can be solved approximately by algorithms based on a linear programming relaxation [3].

A prominent role has the variant with two labels. It can be expressed as quadratic pseudoboolean optimization [4] [5]. Max-flow/min-cut algorithms can be applied to find a partial optimal solution. The LP relaxation of the binary energy minimization is known to be half integral, and it is integral in the case of so called submodular instances. These facts imply that it is possible to apply the simplex algorithm to solve the binary submodular energy minimization. However, the standard implementation of the simplex algorithm is not efficient on large inputs, because the simplex table is too huge.

In this work we study a more efficient method which encodes the simplex tableau by a graph whose size is proportional to the size of the input graph. We show that basic and nonbasic variables form a special structure of subtrees in the input graph. This structure allows to compute rows and columns of the simplex tableau efficiently.

Special versions of the simplex method with similar properties, known as network simplex, have already been described in the literature for other problems [6]. Graph-based simplex method was proposed for the half-integral LP relaxation of the binary energy minimization in [7]. Graph-based simplex method for LP relaxation of submodular instances was investigated in [8]. However, this work does not give a full analysis and description of the algorithm steps. Our goal is to deliver a complete implementation which handles all possible configurations arising during the simplex method iterations.

2. Problem formulation

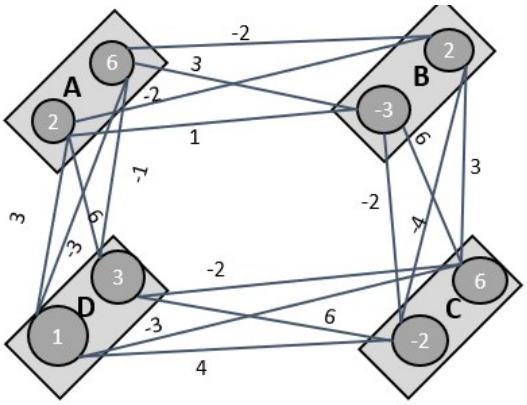
2.1. Binary energy minimization

Let $G = (V, E)$ be an undirected graph where elements of V are called objects and elements of $E \subseteq \binom{V}{2}$ are called object pairs. Let $C = \{0, 1\}$ be a set of labels. For each $u \in V$, let $p_u : C \to R$ be a unary cost function for assigning labels to object u, and for each $\{u, v\} \in E$, let $p_{uv} : C \times C \to R$ be a binary cost function for assigning labels to the objects of the object pair $\{u, v\}$. We adopt that $p_{uv}(k, \ell) = p_{vu}(\ell, k)$ for all $\{u, v\} \in E$ and $k, \ell \in C$. The task of energy minimization is to find

$$\min_{c \in C^V} \left[\sum_{u \in V} p_u(c_u) + \sum_{\{u,v\} \in E} p_{uv}(c_u, c_v) \right]$$

We shortly write $p_u(k)$ as $p_{u:k}$ and $p_{uv}(k, \ell)$ as $p_{uv:k\ell}$. An instance of energy minimization problem is defined as a tuple (V, E, p).

An instance of binary minimization problem can be represented by the scheme bellow. Objects are depicted as boxes. There are two nodes inside each objects representing assignment of label 0 and 1, respectively. Object pairs are represented by edges connecting nodes from the neighboring objects. Numbers in nodes and along edges are values of the unary and binary cost functions.



2.2. Submodular binary instances

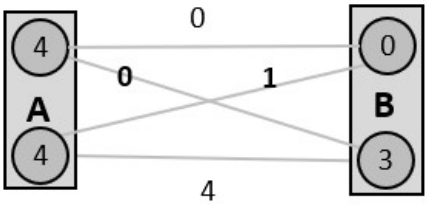
We say that an instance (V, E, p) is submodular if for each object pair $\{u, v\} \in E$:

$$p_{uv:01} + p_{uv:10} \geq p_{uv:00} + p_{uv:11}$$

For each (V, E, p), it is possible to change cost function so that we obtain an equivalent instance (V, E, p') fulfilling $p'_{uv:00} = 0$ and $p'_{uv:11} = 0$ for all $\{u, v\} \in E$.

A desired equivalent instance can be obtained by repeatedly applying so called reparametrization, which is demonstrated by the following schemes.

As a result of the reparametrization, we get an instance:



2.3. Linear programming relaxation

Let (V, E, p) be an instance of binary submodular energy minimization such that $p_{uv:00} = 0$ and $p_{uv:11} = 0$ for all $\{u, v\} \in E$.

We can formulate the following linear (LP) programming relaxation of the problem.

$$\min \sum_{u \in V} (p_{u:0} x_{u:0} + p_{u:1} x_{u:1}) + \sum_{\{u,v\} \in E} (p_{uv:01} x_{uv:01} + p_{uv:10} x_{uv:10})$$

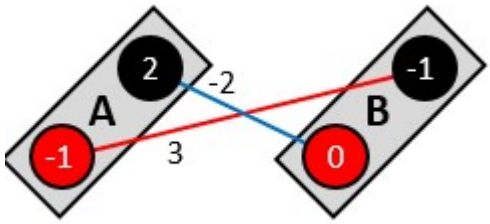
subject to:

$$\begin{aligned} x_{u:0} + x_{u:1} &= 1, & u &\in V \\ x_{u:0} + x_{uv:10} &= x_{v:0} + x_{uv:01}, & \{u, v\} &\in E \\ x &\geq 0 \end{aligned}$$

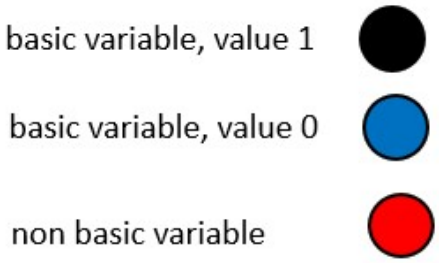
2.4. Solving the LP relaxation by simplex algorithm

We assume that the reader is familiar with the simplex algorithm [9].

We will show how it solves the following instance:



In the scheme above, nodes and edges are in one to one correspondence with LP relaxation variables. We use three colors (black, blue, red) to categorize the variables:



For the given instance, we assemble the simplex table:

	xA:0	xA:1	xB:0	xB:1	xAB:01	xAB:10	d
	-1	2	0	-1	3	-2	0
xA:1	1	1	0	0	0	0	1
xB:1	0	0	1	1	0	0	1
xAB:10	1	0	-1	0	-1	1	0

Before i start simplex algorithm, i need to change values of non-basic variables to zeros. It can be solved with subtraction 2 times first row, then add 1 times second row, and then add one time third row, after this operation we have this simplex table and can start the algorithm.

	xA:0	xA:1	xB:0	x B:1	xAB:01	xAB:10	d
	-1	0	-1	0	1	0	-1
xA:1	1	1	0	0	0	0	1
xB:1	0	0	1	1	0	0	1
xAB:10	1	0	-1	0	-1	1	0

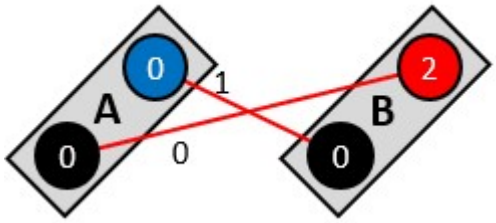
Firstly we need to choose the column with minimal value, in this case it could be the first column or the third. Let choose the third column. For each row find a criterion, for the first is 1/0 (infinity), for the second is (1/1) = 1, for the last one it's (0/-1). Pivot in this situation will be determined by the minimum non negative value. The pivot will be in the second row.
Let make a first iteration of simplex algorithm with addition second row to each row for changing value in the third column to 0. And also we need to add $x_{B:0}$ to a basic variables, and delete $x_{B:1}$ from basic variables. After first iteration we have

	xA:0	xA:1	xB:0	xB:1	xAB:01	xAB:10	d
	-1	0	0	1	1	0	0
xA:1	1	1	0	0	0	0	1
xB:0	0	0	1	1	0	0	1
xAB:10	1	0	0	1	-1	1	1

Ratios for the rows are the minimal negative value in the first column, so there we need to find pivot. Criterion for each row are (1,none,1), so we can choose first and third row, let choose the third one. And let us make the next iteration:

	xA:0	xA:1	xB:0	xB:1	xAB:01	xAB:10	d
	0	0	0	2	0	1	1
xA:1	0	1	0	-1	1	-1	0
xB:0	0	0	1	1	0	0	1
xA:0	1	0	0	1	-1	1	1

In this case all costs are positive, so we don't need next iterations, so simplex algorithm is finished. And this is our optimal decision with cost = (-d), so in this case with cost equal to -1. The result is represented as follows:

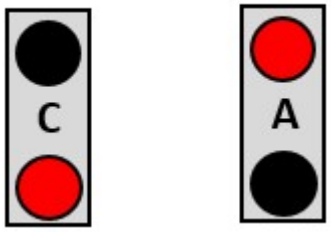


3. Preparations for the algorithm

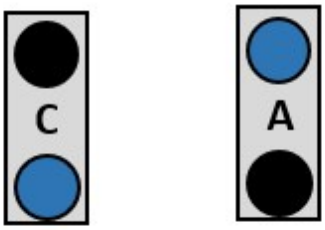
3.1. Introduction to the graph structure

In this chapter we derive equations for basic variables when they are expressed as a linear combination of non-basic variables. It means that we give a characterization of the simplex table rows.

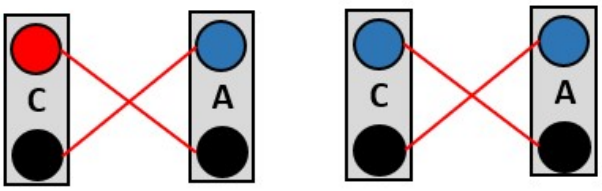
The theorems rely on the structure of basic and non-basic variables described in [8]. The structure is as follows. There are two types of objects.
Root objects contain one basic and one non-basic variable (of value 1, i.e., black).



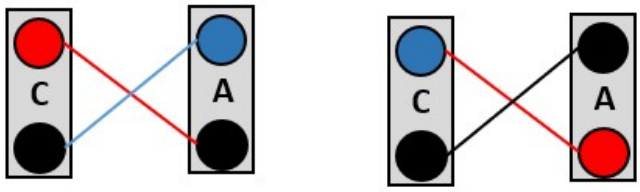
Ordinary objects contain two non-basic variables (of value 0 and 1, i.e., blue and black).



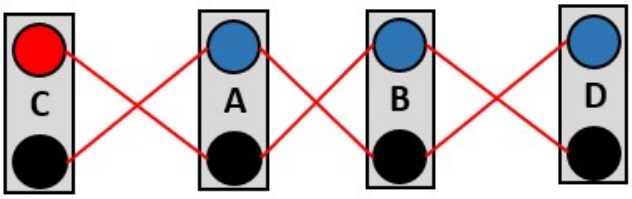
Next, there are two types of object pairs.
Tree object pairs contain two non-basic edge variables (red)



while non-tree object pairs contain one basic (blue or black) and one non-basic variable.



Subgraphs of objects connected by tree edges form trees with exactly one root object.



3.2. Equations for basic variables

3.2.1. Node variables

Let (N, E, p) be an instance of the discrete energy minimization problem where $G = (V, E)$ is a undirected graph and p is a cost vector.

Theorem 3.1. *Let $x_{u:\ell}$ be a basic variable for some object $u \in V$ and a label $\ell \in \{0, 1\}$. Let $u_0, \ldots, u_k$ $(k \geq 0)$ be a tree path from $u = u_k$ to a root u_0. Then it holds*

- *if $x_{u_0:\ell}$ is a non-basic variable, then $x_{u_k:\ell} + \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_0:\ell} = 0$,*
- *otherwise $x_{u_k:\ell} + \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\hat{\ell}} = 1$,*

where $\hat{\ell} = 1 - \ell$.

Proof: By induction on k.

Let $k = 0$. It means that $u = u_k = u_0$ is a root and $x_{u_0:\ell} = x_{u_k:\ell}$ is a basic variable, hence $x_{u:\ell} + x_{u:\hat{\ell}} = 1$, which is equivalent to the theorem statement.

Let $k = 1$. It holds that

$$x_{u_1:\ell} + x_{u_1 u_0:\hat{\ell}\ell} - x_{u_0:\ell} - x_{u_1 u_0:\ell\bar{\ell}} = 0\,. \tag{1}$$

If $x_{u_0:\ell}$ is a non-basic variable, the theorem statement is fulfilled. If $x_{u_0:\ell}$ is a basic variable, we substitute $x_{u_0:\ell} = 1 - x_{u_0:\hat{\ell}}$ to (1) and we obtain

$$x_{u_1:\ell} + x_{u_1u_0:\hat{\ell}\ell} - x_{u_1u_0:\ell\hat{\ell}} + x_{u_0:\hat{\ell}} = 1\,,$$

which shows again that the theorem statement is valid.

Let $k > 1$. It holds that

$$x_{u_k:\ell} + x_{u_ku_{k-1}:\hat{\ell}\ell} - x_{u_{k-1}:\ell} - x_{u_ku_{k-1}:\ell\hat{\ell}} = 0\,. \tag{2}$$

Case 1: Assume that $x_{u_0:\ell}$ is a non-basic variable. By the induction hypotheses, it holds that

$$x_{u_{k-1}:\ell} + \sum_{i=1}^{k-1}\left(x_{u_iu_{i-1}:\hat{\ell}\ell} - x_{u_iu_{i-1}:\ell\hat{\ell}}\right) - x_{u_0:\ell} = 0 \tag{3}$$

which implies

$$x_{u_{k-1}:\ell} = x_{u_0:\ell} - \sum_{i=1}^{k-1}\left(x_{u_iu_{i-1}:\hat{\ell}\ell} - x_{u_iu_{i-1}:\ell\hat{\ell}}\right)\,. \tag{4}$$

We substitute (4) to (2) and obtain:

$$x_{u_k:\ell} + x_{u_ku_{k-1}:\hat{\ell}\ell} - x_{u_ku_{k-1}:\ell\hat{\ell}} + \sum_{i=1}^{k-1}\left(x_{u_iu_{i-1}:\hat{\ell}\ell} - x_{u_iu_{i-1}:\ell\hat{\ell}}\right) - x_{u_0:\ell} = 0\,,$$

$$x_{u_k:\ell} + \sum_{i=1}^{k}\left(x_{u_iu_{i-1}:\hat{\ell}\ell} - x_{u_iu_{i-1}:\ell\hat{\ell}}\right) - x_{u_0:\ell} = 0\,,$$

which proves the theorem.

Case 2: Assume that $x_{u_0:\ell}$ is a basic variable. By the induction hypotheses, it holds that

$$x_{u_{k-1}:\ell} + \sum_{i=1}^{k-1}\left(x_{u_iu_{i-1}:\hat{\ell}\ell} - x_{u_iu_{i-1}:\ell\hat{\ell}}\right) + x_{u_0:\ell} = 1\,. \tag{5}$$

We can express $x_{u_{k-1}:\ell}$ from the above equation and substitute it to (2) to obtain that

$$x_{u_k:\ell} + \sum_{i=1}^{k}\left(x_{u_iu_{i-1}:\hat{\ell}\ell} - x_{u_iu_{i-1}:\ell\hat{\ell}}\right) + x_{u_0:\ell} = 1\,,$$

which again proves the theorem. □

3.2.2. Edges connecting nodes of different trees

Theorem 3.2. *Let $x_{u:\ell}$ and $x_{v:\hat{\ell}}$ be a basic variable for some object $u \in V$ and $v \in V$ and a label $\ell \in \{0,1\}$. Let $u_0, \ldots, u_k$ $(k \geq 0)$ be a tree path from $u = u_k$ to a root u_0 and $v_0, \ldots, v_k$ $(k \geq 0)$ be a tree path from $v = v_k 1$ to a root v_0. Then it holds*

- *if $x_{u_0:\ell}$ is a non-basic variable and $x_{v_0:\ell}$ and is a non-basic variable, then $x_{u_k v_{k1}:\ell\hat{\ell}} + \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_0:\ell} - \sum_{i=1}^{k1} \left(-x_{v_i v_{i-1}:\hat{\ell}\ell} + x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\ell} - x_{u_k v_{k1}:\hat{\ell}\ell} = 0$,*
- *if $x_{u_0:\ell}$ is a basic variable and $x_{v_0:\ell}$ and is a non-basic variable, then $x_{u_k v_{k1}:\ell\hat{\ell}} + \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\hat{\ell}} - \sum_{i=1}^{k1} \left(-x_{v_i v_{i-1}:\hat{\ell}\ell} + x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\ell} - x_{u_k v_{k1}:\hat{\ell}\ell} = 1$,*
- *if $x_{u_0:\ell}$ is a non-basic variable and $x_{v_0:\ell}$ and is a basic variable, then $x_{u_k v_{k1}:\ell\hat{\ell}} + \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_0:\ell} - \sum_{i=1}^{k1} \left(-x_{v_i v_{i-1}:\hat{\ell}\ell} + x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\ell} - x_{u_k v_{k1}:\hat{\ell}\ell} = 1$,*
- *if $x_{u_0:\ell}$ is a basic variable and $x_{v_0:\ell}$ and is a basic variable, then*

$$x_{u_k v_{k1}:\ell\hat{\ell}} + \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\hat{\ell}} - \sum_{i=1}^{k1} \left(-x_{v_i v_{i-1}:\hat{\ell}\ell} + x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\ell} - x_{u_k v_{k1}:\hat{\ell}\ell} = 1 \ ,$$

where $\hat{\ell} = 1 - \ell$.

Proof: First of all, we know the equation for the edge between nodes x_{u_k} and $x_{v_{k_1}}$:

$$- x_{u_k:\ell} - x_{u_k v_{k1}:\hat{\ell}\ell} + x_{v_{k1}:\ell} + x_{u_k v_{k1}:\ell\hat{\ell}} = 0 \, . \tag{6}$$

Case 1: Assume that $x_{u_0:\ell}$ is a non-basic variable and $x_{v_0:\ell}$ and is a non-basic variable, then we have 2 equations from theorem 5.1:

$$- x_{u_k:\ell} = \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_0:\ell} \, . \tag{7}$$

$$x_{v_{k1}:\ell} = - \sum_{i=1}^{k1} \left(x_{v_i v_{i-1}:\hat{\ell}\ell} - x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\ell} \, . \tag{8}$$

We substitute (7) and (6) and also substitute (8) and (6), and obtain:

$$\sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_0:\ell} - x_{u_k v_{k1}:\hat{\ell}\ell} - \sum_{i=1}^{k1} \left(x_{v_i v_{i-1}:\hat{\ell}\ell} - x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\ell} + x_{u_k v_{k1}:\ell\hat{\ell}} = 0 \, . \tag{9}$$

Case 2: Assume that $x_{u_0:\ell}$ is a basic variable and $x_{v_0:\ell}$ and is a non-basic variable, then we have 2 equations from theorem 5.1:

$$-x_{u_k:\ell} = \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\hat{\ell}} - 1\,. \tag{10}$$

$$x_{v_{k1}:\ell} = -\sum_{i=1}^{k1} \left(x_{v_i v_{i-1}:\hat{\ell}\ell} - x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\ell}\,. \tag{11}$$

We substitute (10) and (6) and also substitute 11 and (6), and obtain:

$$\sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\hat{\ell}} - x_{u_k v_{k1}:\hat{\ell}\ell} - \sum_{i=1}^{k1} \left(x_{v_i v_{i-1}:\hat{\ell}\ell} - x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\ell} + x_{u_k v_{k1}:\ell\hat{\ell}} = 1\,. \tag{12}$$

So it's equal to the theorem Case 3: assume,that $x_{u_0:\ell}$ is a non-basic variable and $x_{v_0:\ell}$ and is a basic variable, then we have 2 equations from theorem 5.1:

$$-x_{u_k:\ell} = \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_0:\ell}\,. \tag{13}$$

$$x_{v_{k1}:\ell} = -\sum_{i=1}^{k1} \left(x_{v_i v_{i-1}:\hat{\ell}\ell} - x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + 1 - x_{v_0:\hat{\ell}}\,. \tag{14}$$

We substitute (13) and (6) and also substitute (14) and (6), and obtain:

$$-\sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\ell} + x_{u_k v_{k1}:\hat{\ell}\ell} + \sum_{i=1}^{k1} \left(x_{v_i v_{i-1}:\hat{\ell}\ell} - x_{v_i v_{i-1}:\ell\hat{\ell}} \right) + x_{v_0:\hat{\ell}} - x_{u_k v_{k1}:\ell\hat{\ell}} = 1\,. \tag{15}$$

it's also equal to the theorem Case 4: assume,that $x_{u_0:\ell}$ is a basic variable and $x_{v_0:\ell}$ and is a basic variable, then we have 2 equations from theorem 5.1:

$$-x_{u_k:\ell} = \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\hat{\ell}} - 1\,. \tag{16}$$

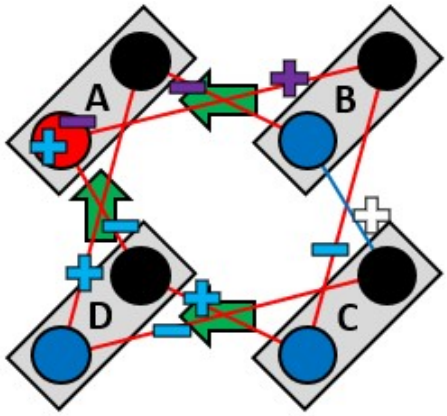
$$x_{v_{k1}:\ell} = -\sum_{i=1}^{k1} \left(x_{v_i v_{i-1}:\hat{\ell}\ell} - x_{v_i v_{i-1}:\ell\hat{\ell}} \right) - x_{v_0:\hat{\ell}} + 1\,. \tag{17}$$

We substitute (16) and (6) and also substitute (17) and (6), and obtain:

$$\sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\hat{\ell}} - x_{u_k v_{k1}:\hat{\ell}\ell} - \sum_{i=1}^{k1} \left(x_{v_i v_{i-1}:\hat{\ell}\ell} - x_{v_i v_{i-1}:\ell\hat{\ell}} \right) - x_{v_0:\hat{\ell}} + x_{u_k v_{k1}:\ell\hat{\ell}} = 0\,. \tag{18}$$

□

3.2.3. Edges connecting nodes of the same tree



Theorem 3.3. *Let $x_{u:\ell}$ be a basic variable for some object $u \in V$ and a label $\ell \in \{0,1\}$. Let $u_0, \ldots, u_k$ $(k \geq 0)$ be a tree path from $u = u_k$ to a root u_0 and $u_0, \ldots, u_{k1}$ $(k1 < 0)$ be a tree path from $v = v_k 1$ to a root v_0 Then it holds*

- $x_{u_k u_{k1}:\ell\hat{\ell}} + \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - \sum_{i=k1+1}^{0} \left(-x_{u_i u_{i-1}:\hat{\ell}\ell} + x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_k u_{k1}:\hat{\ell}\ell} = 0$

where $\hat{\ell} = 1 - \ell$.

Proof: First of all, we know the equation for the edge between nodes x_{u_k} and $x_{u_{k_1}}$:

$$- x_{u_k:\ell} - x_{u_k u_{k1}:\hat{\ell}\ell} + x_{u_{k1}:\ell} + x_{u_k u_{k1}:\ell\hat{\ell}} = 0\,. \tag{19}$$

Also from the theorem 5.1 we know next equations for $x_{u_k:\ell}$ and $x_{u_k:\ell}$:

$$- x_{u_k:\ell} = \sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_0:\ell} \tag{20}$$

$$x_{u_{k_1}:\ell} = - \sum_{i=k1+1}^{0} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_0:\ell} \tag{21}$$

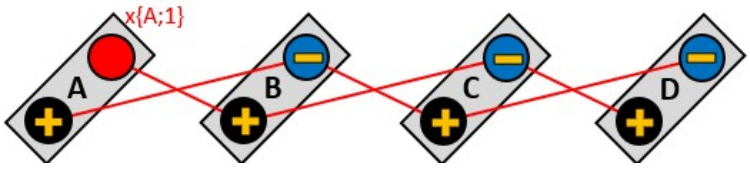
Let substitute (20),(21) with (19):

$$\sum_{i=1}^{k} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) - x_{u_k u_{k1}:\hat{\ell}\ell} - \sum_{i=k1+1}^{0} \left(x_{u_i u_{i-1}:\hat{\ell}\ell} - x_{u_i u_{i-1}:\ell\hat{\ell}} \right) + x_{u_k u_{k1}:\ell\hat{\ell}} = 0\,. \tag{22}$$

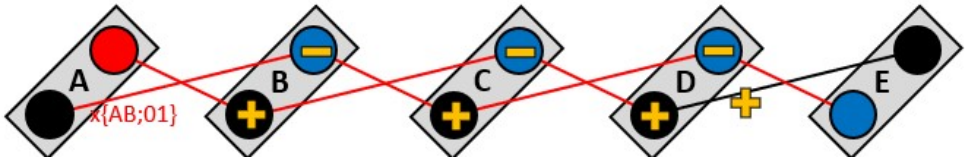
In other case if a $x_{u_0:\ell}$ is a basic variable, those values will also disappear and will be the same equation. Edge always has value 0 because basic edge with value 1 can only be between two basic variables, but because it carried out between two roots of the same tree it can't exist. □

3.2.4. Coefficients in simplex table columns

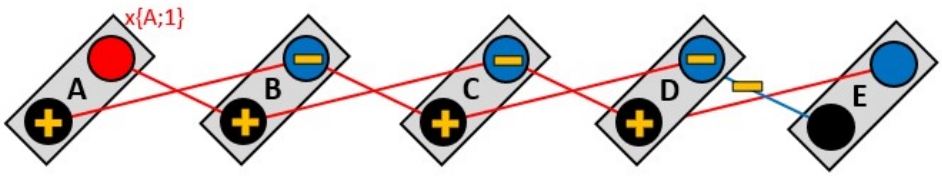
Let us assume that u_1 is a root, and we know,that $x_{u_1:0} = 1$.



And assume, that we want to find the coefficients of all basic variables for $x_{u_1:1}$
$x_{u_n:0} : x_{u_n:0} + \sum_{k\in 1:n}(X_{u_k u_{k+1}:10}) - \sum_{k\in 1:n}(X_{u_k u_{k+1}:01}) + x_{u_1:1} = 1$ ("+")
$x_{u_n:1} : x_{u_n:1} - \sum_{k\in 1:n}(X_{u_k u_{k+1}:10}) + \sum_{k\in 1:n}(X_{u_k u_{k+1}:01}) - x_{u_1:1} = 0$ ("−")
So we find the signs for unary variables, let now find the signs for binary variables



$x_{u_n u_{n+1}:01} : ... + x_{u_n u_{n+1}:01} + \sum_{k\in 1:n-1}(X_{u_k u_{k+1}:01}) - \sum_{k\in 1:n-1}(X_{u_k u_{k+1}:01}) + x_{u_1:1} = 1$ ("+")



$x_{u_n u_{n+1}:10} : ... + x_{u_n u_{n+1}:10} - \sum_{k\in 1:n-1}(X_{u_k u_{k+1}:01}) + \sum_{k\in 1:n-1}(X_{u_k u_{k+1}:01}) - x_{u_1:1} = 0$ ("−")

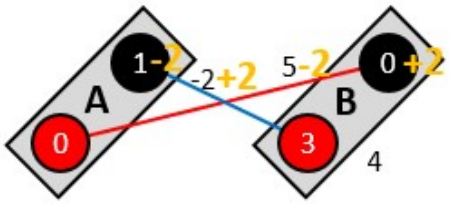
4. Description of the algorithm

4.1. Definitions of the parts of the simplex algorithm

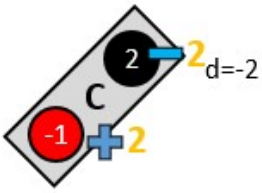
"Blue variable" means variable which is basic and has the value 0 in simplex table.
"Black variable" means variable which is basic and has the value 1 in simplex table.
"Red variable" means non-basic variable and has the value 0 in simplex table.
"Green edges" means directions from children to the root of all trees in this graph
"Blue edges" means entering and leaving variables.

4.2. Changing costs of all basic variables to 0

Before the first iteration of the simplex algorithm, we should change all costs of variables to 0, using theorems, that we proved before.
Firstly we should change costs of all basic variable, and we do this with next operation.



And after this we should change costs of all nodes with next operation



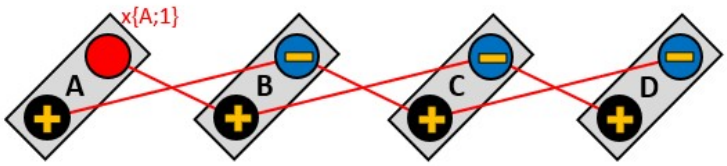
And then we can go to the iterations.

4.3. Searching for a leaving variable

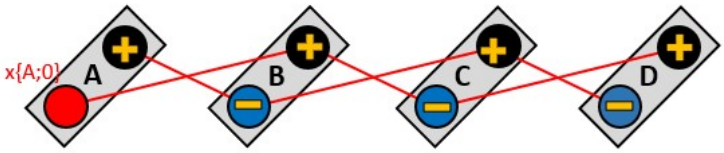
We find element with entering variable and leaving variable. First of all we need to choose variable, that has the minimum cost. There are two different situations, which we can get after this step:
1) If the minimum-cost variable is a node it would be our entering variable.
2) If the minimum-cost variable is an edge it would be our entering variable.

4.3.1. Entering variable is a node

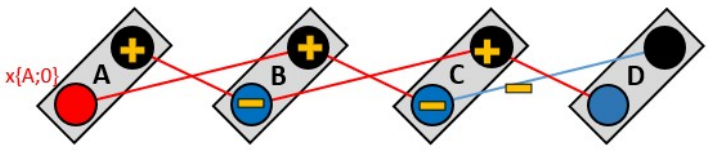
After theorems, which we had prove in third part of the project, we can get next rules how to choose a pivot. If our entering variable has a label 1. So in this case when our entering variable is $X_{A:1}$, all varibles in this tree, which has the opposite label as $X_{B:1}$, $X_{C:1}$ has a positive coefficient of minimal value in their equations. So they can be chosen as pivot. So it can be a leaving variable. But they don't get privelege if we will get other pivots, but with other value. So it would be Blue variables.



If our entering variable has a label 0. So in this case when our minimal-cost variable is $X_{A:0}$, all variables in this tree, which has the opposite label as $X_{B:0}$, $X_{C:0}$ has a positive coefficient of minimal value in their equations. So they can be chosen as leaving variable. But they don't get privelege if we will get other pivots, but with blue color. So it would be Blue variables.



Next we will consider other situations how can we choose an edge as a leaving variables. So in this case if we have $X_{A:0}$ as a entering variable, we should choose an edge which goes from a opposite label. So in this case it would be any line , such as $X_{AB:10}, X_{BC:10}, X_{CD:10}$, So it would be any line with this direction which goes from any end of this tree.



2) Implementation for searching leaving variable, when entering variable is a node;
For implementation of searching leaving variable, i am using breadth-first search algorithm. Starting from a root node, where is based our entering variable, and for each node we decide which of two variables are possible pivots. In previous cases it would be any node black variable, which lies in nodes of the tree. When this node have at least in one direction an edge, which is not in this tree (don't have colors of 01 and 10 both red). So in this

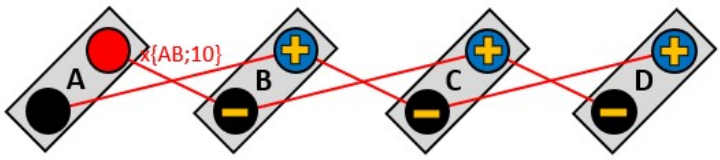
case if one of the line which has the same label, that our entering variable (for example our minimal variable has label 0, so edge should have label 01) has color different from red("Blue" or "Black"). If we get blue pivot, we terminate the search, because we found a pivot for next step and this is our pivot for the next step. If in the end of the searching we have only black pivots, we take one of them.

Algorithm 1 Searching for a leaving variable

```
1: queue ← ()
2: queue ← startNode
3: while queue ≠ empty do
4:     node ← queue.pop
5:     for neigbour in node.neigbours do
6:         if neigbour is a child of node then
7:             pivotBlueNode ← node.node0 or node.node1 and color
8:             pivotBlackNode ← node.node0ornode.node1andcolor
9:             queue ← neigbour
10:        else
11:            pivotBlueLine ← line.line01orline.line10andcolor   ▷ The line that not in
   the tree, but goes out from node of this tree
12:            pivotBlackLine ← line.line01orline.line10andcolor
13:        end if
14:    end for
15: end while
```

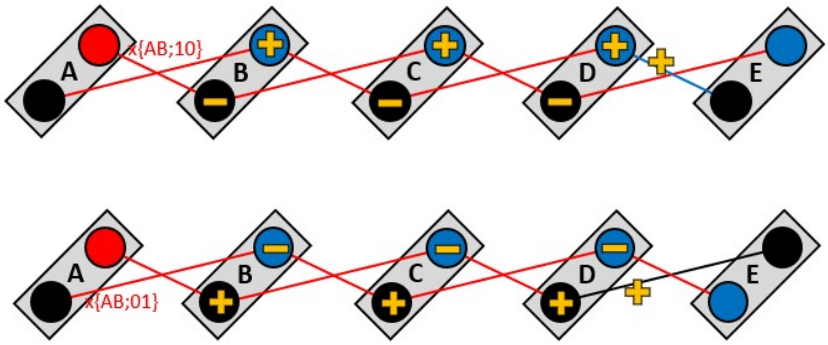
4.3.2. Entering variable is an edge

. Similary as in the search of a pivot node, we should choose any variables in the nodes of the same tree, which has the same label. For example if $X_{AB:10}$ is a entering variable, we need to search all variables of nodes which has label 1. For example $X_{B:1}, X_{C:1}, X_{D:1}$, they can be chosen as leaving variable. If we had $X_{AB:01}$ as a entering variable, we choose one variable of all variables of nodes in this tree which has label 0, such as $X_{B:0}, X_{C:0}, X_{D:0}$.



When searching for an edge to be taken as the leaving variable, we should search any line, which is not in the tree, but goes from the endings of this tree. And those should have

the same label as our minimal line. For exampe if we have $X_{AB:01}$ as an entering variable, and we search for it leaving edge variable, we should choose line which has the same label $X_{DE:01}$, but if we have $X_{AB:10}$ as a line, we should choose as a pivot line $X_{DE:10}$, so it would be our leaving variable.



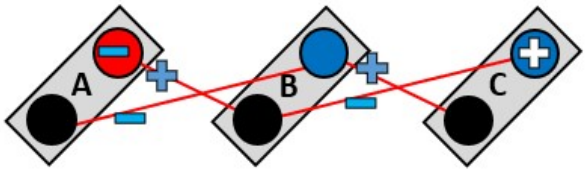
For the search we use the same algotithm as in previous case, but this time we start the search from a node, in which our line leads. And from this node we perform the search, if we find a blue pivot we terminate searching, otherwise we continue.

4.4. Distract raw in simplex table

Distract the raw in simplex table of leaving variable, from costs of all variables , which are presented in equations of leaving variable. We should consider two different examples:
1) If leaving variable is a node variable.
2) If leaving variable is a edge variable.

4.4.1. Leaving variable is a node

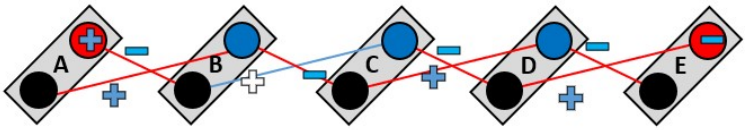
Because after theorems for node variable. We know their equation;



And we need to distract this raw from the costs, of variable, which are presented in this equation. And we need to distract this raw as many times, as a cost of entering variable. So for example we need to distract from $X_{AB:10}$ cost of entering variable. And add to the $X_{AB:01}$ cost of entering variable. And make the same thing for others variables of this equations

4.4.2. Leaving variable is an edge

Because after theorems for edge variable we know their equation. So equation of $X_{BC:01}$ will be:



And we make the same operations as in previous example. From each variable, that has a positive coefficient in the equation, we need to distract cost of entering variable. For example from $X_{BC:01}$,$X_{AB:01}$,$X_{A:1}$, and distract cost of entering variable from costs of all variables, which have negative coefficient in equation of leaving variable. Such as $X_{CD:10}$,$X_{DE:10}$,$X_{E:1}$.

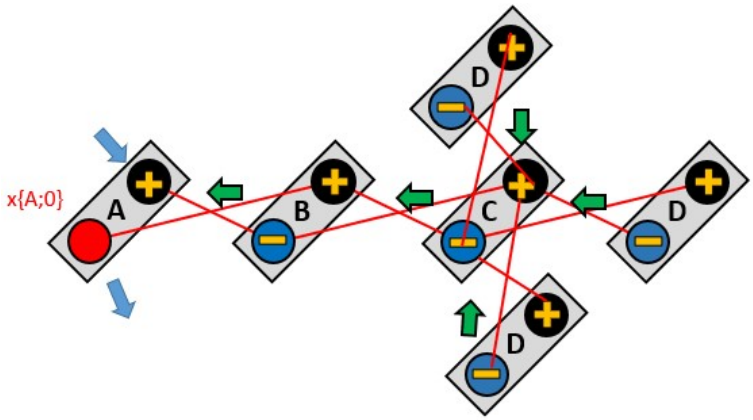
4.5. Changes of direction and changes of colors

Next step of our simplex algorithm would be distraction of our pivot from each other line. For that we use changing of directories and of colors. We should consider 4 different situations:
1) Entering variable and leaving variable are both nodes
2) Entering variable is a node, but leaving variable is a edge
3) Entering variable is a edge, but leaving variable is a node
4) Entering variable and leaving variable are both edges

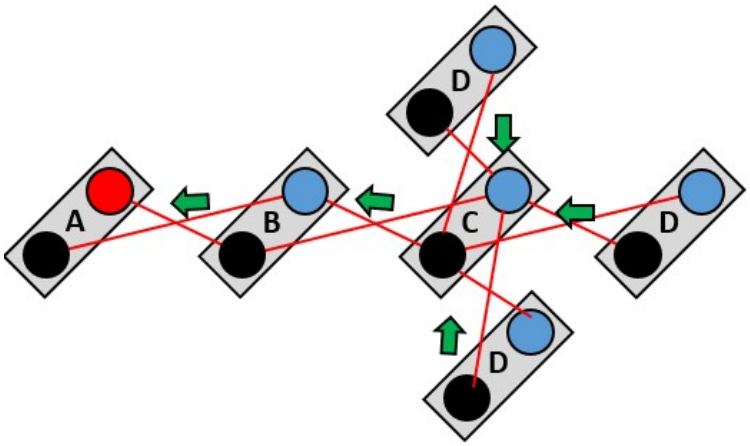
4.5.1. Entering variable and leaving variable are both nodes

In this example we also have two different situations:
1) Both entering and leaving variables stay in the same node
2) Leaving variables and entering variables stay in different nodes

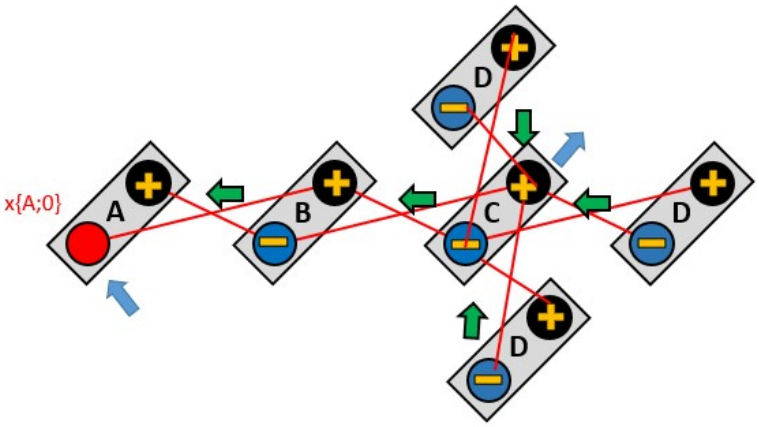
1) Both entering and leaving variables are located in the same node



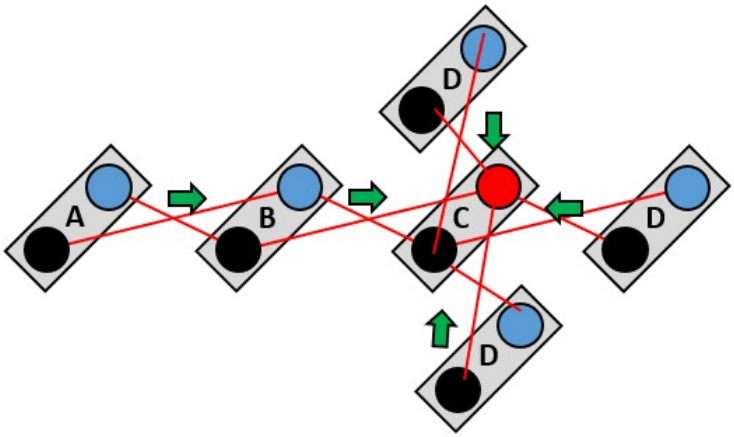
In this situation we don't need to change directions, because node, which contains both this variables will still be a root of the tree, so we only need to change colors between themselves in each node of the tree. So it would be the result:



But if entering and leaving variables lean in different nodes:



We should change directions of all edges, that adjacent from our root to the node, which obtains leaving variable, and this node would be the next root, and because pivot is black we also need to change colors of other variable in this node and then change colors of all node in this tree



Algorithm 2 Changing colors of nodes in the tree

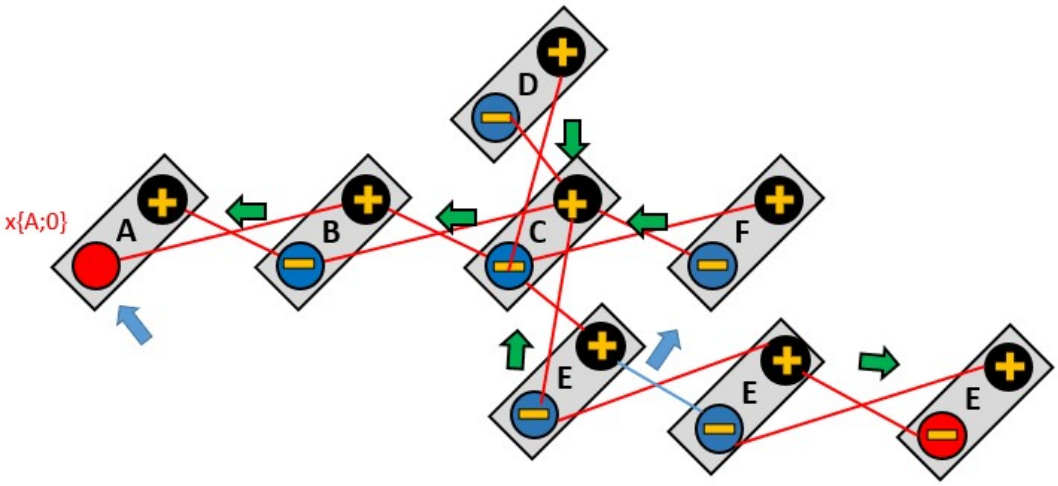
```
1: nwev                                                    ▷ (node with entering variable)
2: nwlv                                                    ▷ (node with leaving variable)
3: queue ← nwev
4: while queue ≠ empty do
5:     node ← queue.pop
6:     node.node0.color ← node.node1.color
7:     node.node1.color ← node.node0.color
8:     for neigbour in node.neigbours do
9:         if neigbour is a child then
10:            queue ← neigbour
11:        else
12:            Change colours of lines that go from ending variables
13:        end if
14:    end for
15: end while
```

4.5.2. Entering variable is a node and leaving variable is an edge

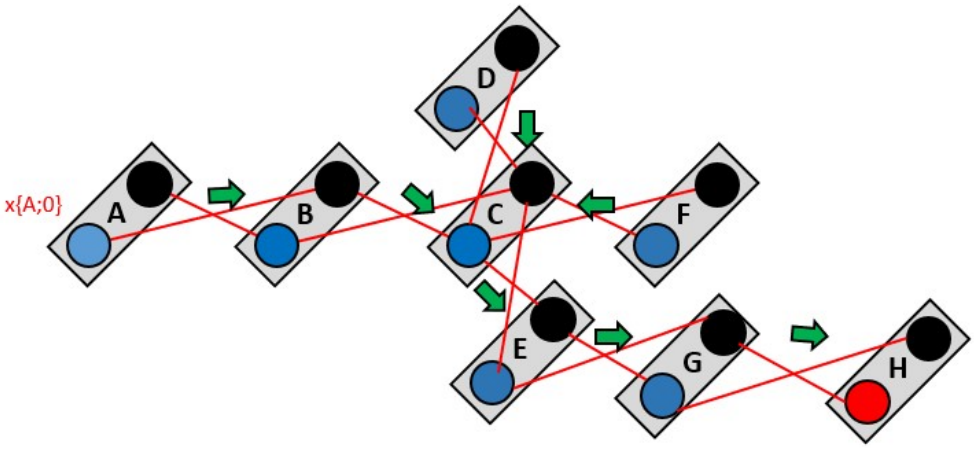
In this section we also have two different situations :
1) Leaving variable is a blue edge.
2) Leaving variable is a black edge.
1) Entering variable is a blue edge

Algorithm 3 Changing direction

```
1: nwev                                          ▷ (node with entering variable)
2: nwlv                                          ▷ (node with leaving variable)
3: node ← nwlv
4: while node.parent ≠ empty do
5:     parent ← node.parent
6:     line ← line between node and parent
7:     line.ChangeDirection
8:     node ← node.parent
9: end while
```



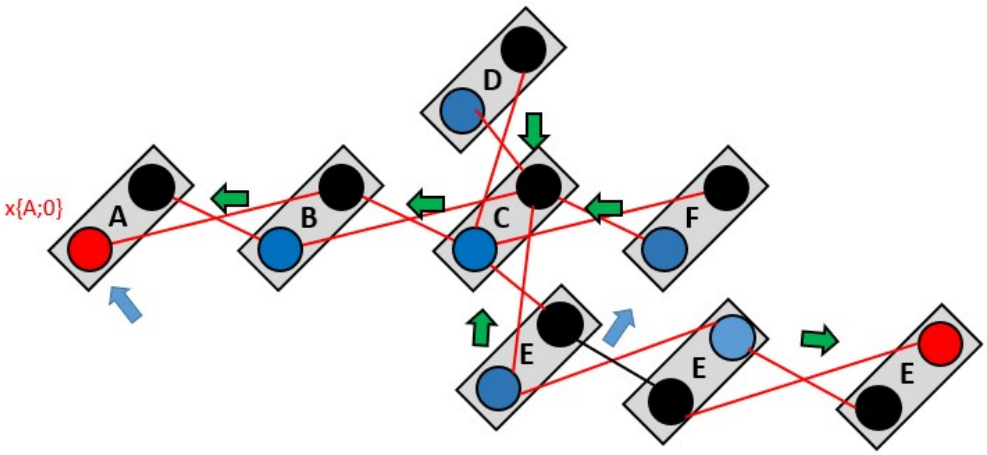
In this case we don't need to change any colors of our tree which has leaving variables. Because the new tree, have the same colors of their nodes as our tree, because this edge is blue. In this case we need to change directions of all edges that go from root of our old tree to the node, from which leaving variable is starting. And then we change our entering variable to blue color, and also change leaving variable to red color. And edge which had this leaving variable will have direction from ending of old tree to the new tree.


x{A;0}
A
B
C
D
E
F
G
H

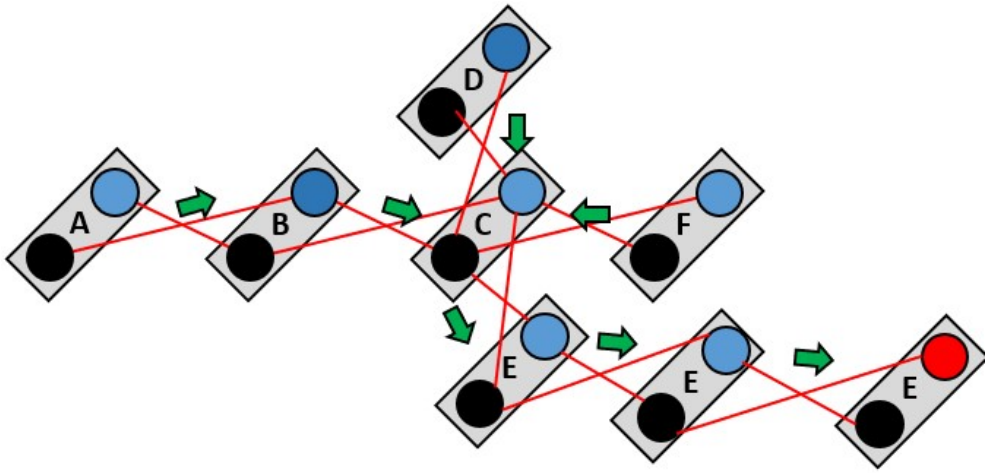
Algorithm 4 Changing direction for entering var node and for leaving var edge

```
1: nwev                                              ▷ (node with entering variable)
2: lwlv                                              ▷ (edge with leaving variable)
3: nodeOldTree ← line.nodefrom
4: nodeNewTree ← line.nodeto
5: node ← nodeoldtree
6: while node.parent ≠ empty do
7:     parent ← node.parent
8:     line ← line between node and parent
9:     line.ChangeDirection
10:    node ← node.parent
11: end while
12: nwev.ev.color ← "blue"
13: lwlv.lf.color ← "red"
14: line.ChangeDirectionfromto(nodeOldTree,nodeToTree)
```

2) Entering variable is a black edge



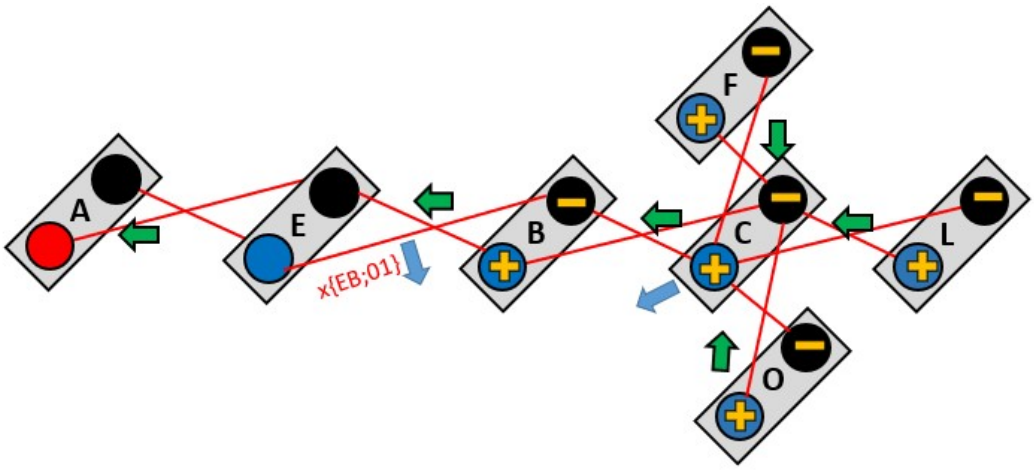
In this case we has the same algorithm as in previous case, because we need to change directions of the edges to the root of old tree. But because edge is black, and it means that colors of old tree are different from colors of the new tree. That means that before algorithm 4, we need to make algorithm 2,which will change colors of all nodes between their variables. An after that we will have the result



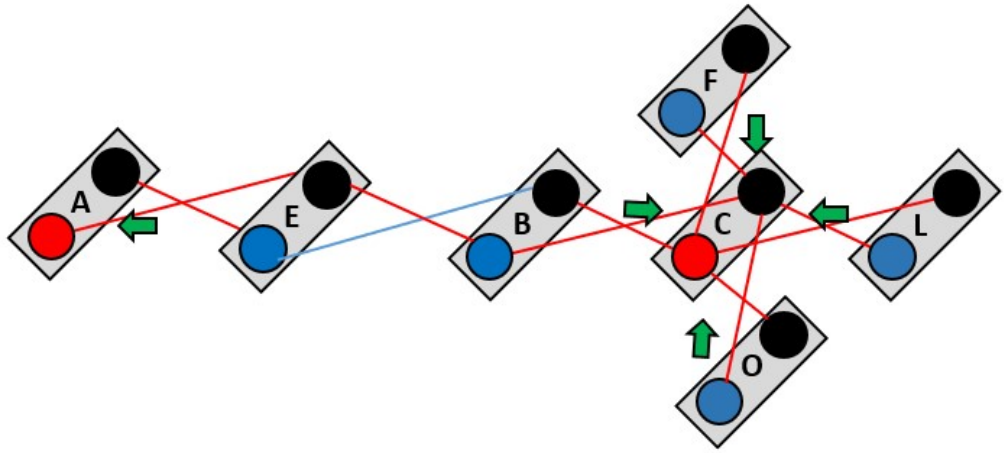
4.5.3. Entering variable is an edge and leaving variable is a node

In this example we also have two possible situations:
1) Leaving variable is a blue node.
2) Leaving variable is a black node.

1) Leaving variable is a blue node.



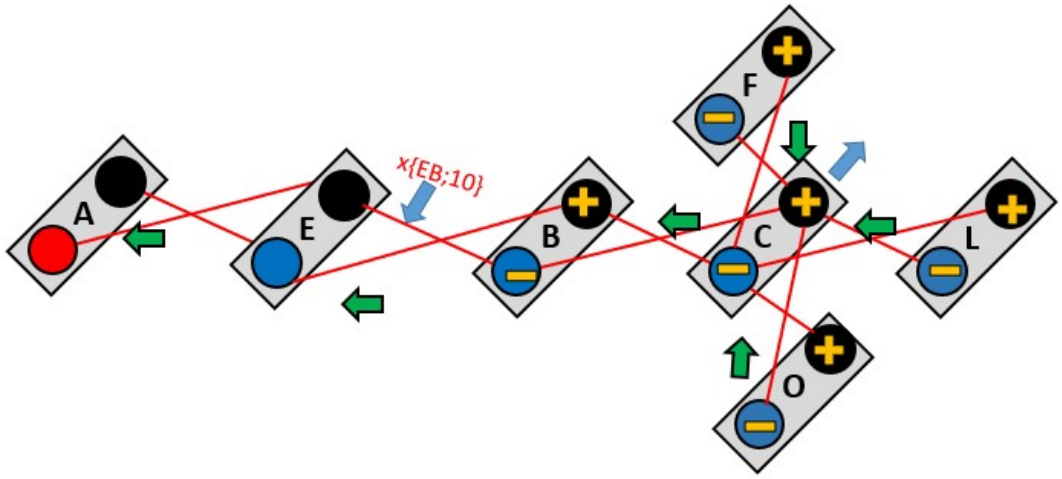
In this case we need to separate tree, by changing colors of entering variable, now this edge, where is located entering variable will be an edge between two trees. And node, which has leaving variable will be root of the new tree. And after this we should change direction of all edges that go from the root of new tree to the edge, which has entering variables.



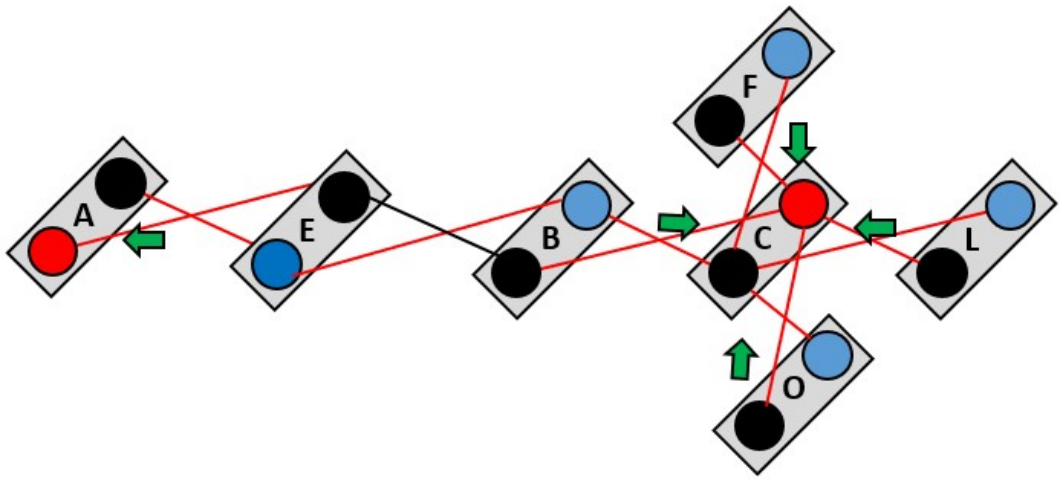
Algorithm 5 Changing direction for entering var line and for leaving var node

```
1: lwev                                              ▷ (line with entering variable)
2: nwlv                                              ▷ (node with leaving variable)
3: nodeOldTree ← lwev.nodefrom
4: nodeNewTree ← lwev.nodeto
5: node ← nwlv
6: while nodet ≠ nodeOldTree do
7:     parent ← node.parent
8:     line ← line between node and parent
9:     line.ChangeDirection
10:    node ← node.parent
11: end while
12: lwev.ev.color ← nwlv.if.color
13: nwlv.lf.color ← "red"
14: nwlv.parent ← nwlv
```

2) leaving variable is a black node:



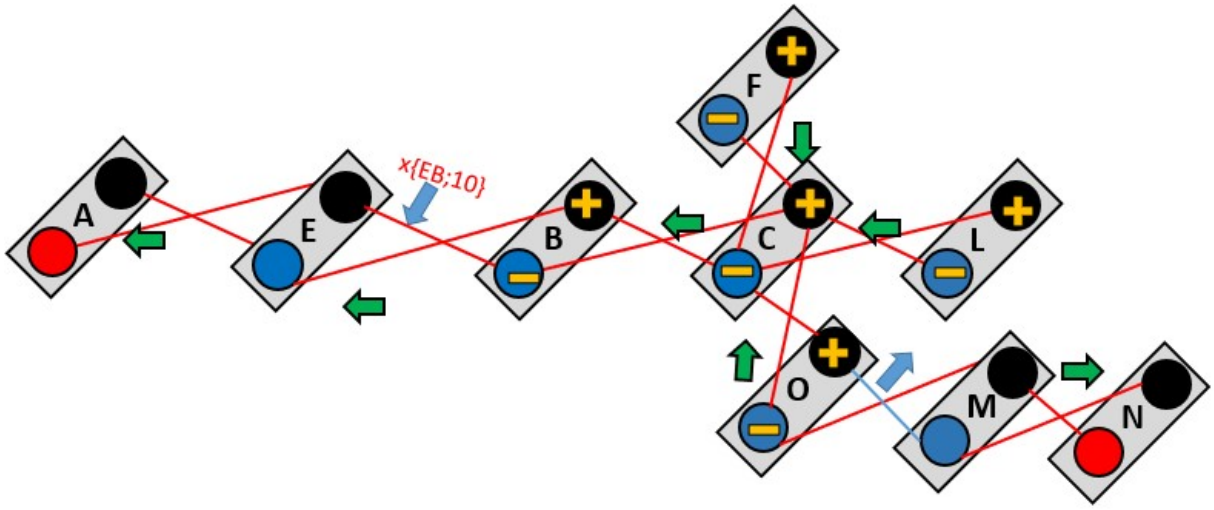
For this situation we need to do the same algorithm as in the previous case, when leaving variable was blue pivot. But only thing that different in these solving. Before last algorithm, that will separate this tree, we need to swap colors of variables in each node of the tree, starting with node, where is starting edge, which has entering variable. We change this nodes, with algorithm 2. And result would be



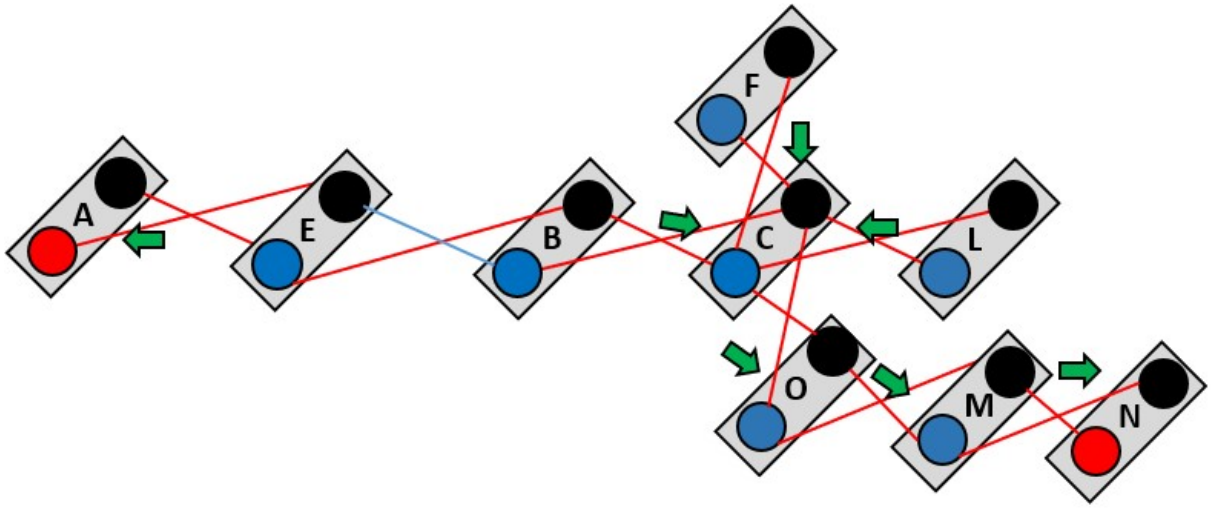
4.5.4. Entering variable is an edge and leaving variable is an edge

In this example we also have two different situations:
1) Leaving variable is a blue edge.
2) Leaving variable is a black edge.

1) Leaving varriable is a blue edge



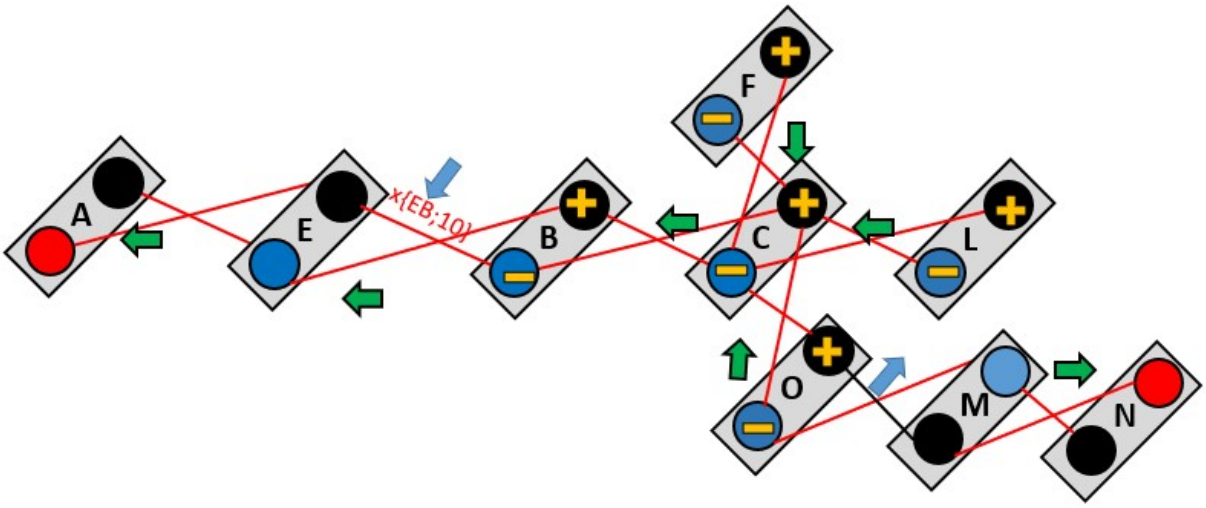
Like in the previous example we need to separate our tree by entering variable, and join remaining tree to other tree by changing color of leaving variables. So we need to change directions of all edges, which lie on the road from leaving edge and entering edge. And after we change colors of entering and leaving variables, this step will be finished. We don't need to swap colors in nodes, because this leaving edge is blue, that means that tree to which we want to join our remaining tree has the same colors of nodes as our tree.



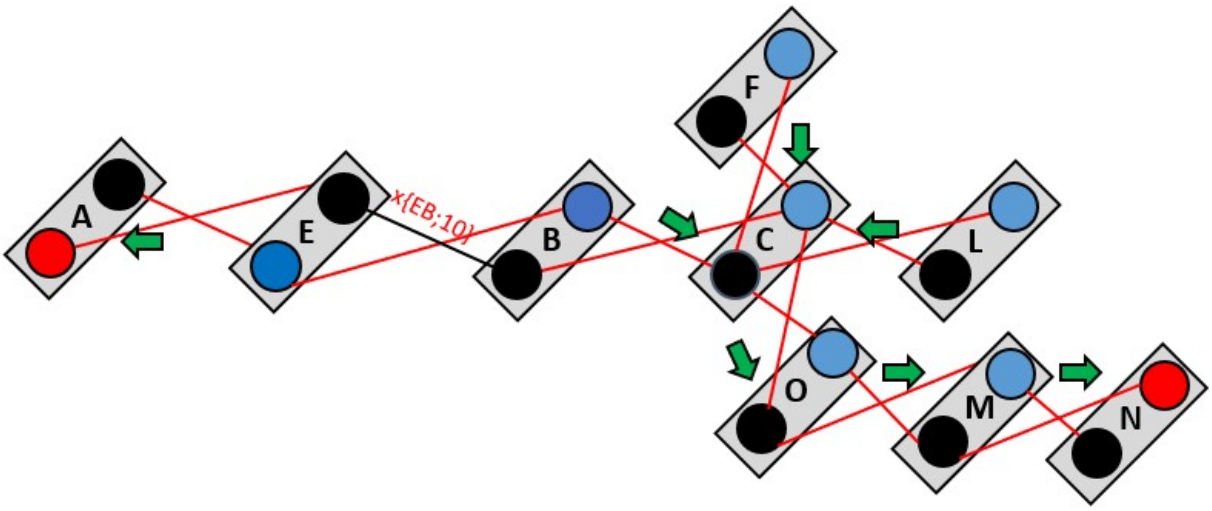
Algorithm 6 Changing direction for entering var edges and for leaving var edges

1: $lwev$ ▷ (line with entering variable)
2: $lwlv$ ▷ (line with leaving variable)
3: $nodeOldTree \leftarrow lwev.nodefrom$
4: $nodeNewTree \leftarrow lwev.nodeto$
5: $nodeOldTreeLeaving \leftarrow lwlv.nodefrom$
6: $nodeNewTreeLeaving \leftarrow lwlv.nodeto$
7: $node \leftarrow nodeOldTreeLeaving$
8: **while** $node \neq nodeNewTree$ **do**
9: $parent \leftarrow node.parent$
10: $line \leftarrow$ `line between node and parent`
11: `line.ChangeDirection`
12: $node \leftarrow node.parent$
13: **end while**
14: $lwev.ev.color \leftarrow lwlf.if.color$
15: $lwlv.lf.color \leftarrow$ "red"
16: $nodeOldTreeLeaving.parent \leftarrow nodeNewTreeLeaving$

2) Leaving variable is a black edge



In this situation we should do the similar things, that in previous sittation, but before this we need to swap colors in all nodes of the remaining tree, that we join to another tree. And we will start changing colors of nodes from a node, where is starting a line, which has entering variable. And then we perform the algorithm 6, and change directions of all edges, which goes from leaving variable to entering variable. And we get this result:



And after this step, we take the next entering variable, which has minimum cost. And search for it leaving variables.

5. Implementation

For implementation of this algorithm I used a programming language C++. For this implementation I have made a visualisation, which shows how the graph representing a simplex method state looks like. And for the creation of the application, which performs visualization I used Visual Studio environment.
For a graph is used class, which is called "graph". Graph had lists of all lines and of all nodes in this graph, and have all methods, that we choose in our algorithm, which was described in description of the algorithm.

```
public:

    bool insert(int x, int y,std::string color0, std::string color1, int cost0, int cost1);

    bool insertline(int xfrom,int yfrom,int xto,int yto,std::string color10,std::string color01, int cost01, int cost10);

    void makealltrees();

    void eliminate_basic();

    void find_pivot_for_node(sub_node* node);

    void find_pivot_for_line(sub_line* pivot);

    void distract_values_with_pivot_node(sub_node* pivot, int times);

    void distract_values_with_pivot_line(sub_line* pivot, int times);

    void change_colors_node_node(sub_node* raw, sub_node* pivot);

    void change_colors_line_node(sub_line* raw, sub_node* pivot);

    void change_colors_line_line(sub_line* raw, sub_line* pivot);

    void change_colors_node_line(sub_node* raw, sub_line* pivot);
public:
    node* root = nullptr;
    node* allnodes[100][100] = {nullptr};
    std::map<std::pair<std::pair<int, int>, std::pair<int, int>>, line*> alllines;
    std::vector<sub_node*> pq_nodes;
    std::vector<sub_line*> pq_lines;
    int width = 0;
    int height = 0;
};
```

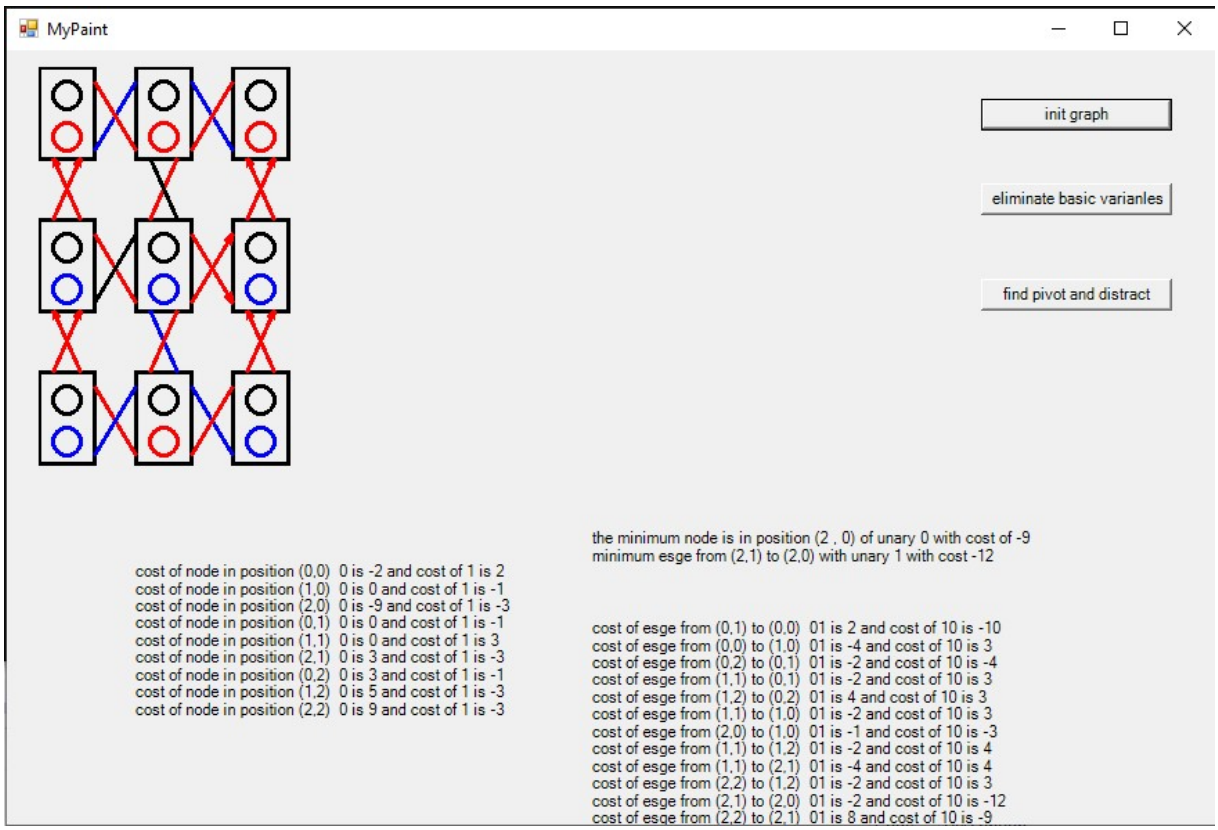
Edge is represented as a structure, which has coordinates of two nodes, where lies our line. Also this edge have two subnodes, which obtain two edges. It's our edge variables. And these variables have in their structure: cost and their color.

```
struct sub_line
{
    int unary;
    int cost;
    struct line *parent;
    std::string color;
};

struct line
{
    int xfrom;
    int yfrom;
    int xto;
    int yto;
    sub_line *edge_01;
    sub_line *edge_10;
    bool fromstarttoend = false;
};
```

A node is represented by its coordinates. It has also information about its parent, and about the root of the tree, where this node lies. And also has two subnodes, these is its node variables, and they have information about their costs and colors.

```
struct sub_node
{
    int unary;
    int cost;
    struct node *parent;
    std::string color;
};

struct node
{
    node* root_of_the_tree = nullptr;
    int posistion_x_maze;
    int posistion_y_maze;
    sub_node *node_0;
    sub_node *node_1;
    int num_of_neighbours = 0;
    node* neighbours[10] = { nullptr };
    node* parent = nullptr;
    bool visited = false;
    node* root;
};
```

The full application has three possible actions. First action is initialisation of graph from a text file. Second action "eliminate basic variable". It's the first thing, that we should do before starting iterations of simplex algorithm. And third action make one iteration of simplex algorithm, and write chosen pivot on a standard output. Bellow graph I have information about all nodes and lines, which has this graph.


MyPaint
init graph
eliminate basic varianles
find pivot and distract
cost of node in position (0,0) 0 is -2 and cost of 1 is 2
cost of node in position (1,0) 0 is 0 and cost of 1 is -1
cost of node in position (2,0) 0 is -9 and cost of 1 is -3
cost of node in position (0,1) 0 is 0 and cost of 1 is -1
cost of node in position (1,1) 0 is 0 and cost of 1 is 3
cost of node in position (2,1) 0 is 3 and cost of 1 is -3
cost of node in position (0,2) 0 is 3 and cost of 1 is -1
cost of node in position (1,2) 0 is 5 and cost of 1 is -3
cost of node in position (2,2) 0 is 9 and cost of 1 is -3
the minimum node is in position (2 , 0) of unary 0 with cost of -9
minimum esge from (2,1) to (2,0) with unary 1 with cost -12
cost of esge from (0,1) to (0,0) 01 is 2 and cost of 10 is -10
cost of esge from (0,0) to (1,0) 01 is -4 and cost of 10 is 3
cost of esge from (0,2) to (0,1) 01 is -2 and cost of 10 is -4
cost of esge from (1,1) to (0,1) 01 is -2 and cost of 10 is 3
cost of esge from (1,2) to (0,2) 01 is 4 and cost of 10 is 3
cost of esge from (1,1) to (1,0) 01 is -2 and cost of 10 is 3
cost of esge from (2,0) to (1,0) 01 is -1 and cost of 10 is -3
cost of esge from (1,1) to (1,2) 01 is -2 and cost of 10 is 4
cost of esge from (1,1) to (2,1) 01 is -4 and cost of 10 is 4
cost of esge from (2,2) to (1,2) 01 is -2 and cost of 10 is 3
cost of esge from (2,1) to (2,0) 01 is -2 and cost of 10 is -12
cost of esge from (2,2) to (2,1) 01 is 8 and cost of 10 is -9

6. Conclusion

In this thesis we have delivered the implementation of graph-based simplex algorithm for solving submodular instances of binary energy minimization problem. Idea of using this algorithm for those instances have some potential, because it decreases memory needed for storing data and can be useful for solving these instances.

Unfortunately we didn't test enough this algorithm, so we can not assert, that this algorithm is efficient for big datasets.

But we believe that this graph-based algorithm can be very useful for solving submodular instances of binary energy minimization problem, because of reducing amount of memory, than in standard simplex algorithm. And if this algorithm is not efficient, there could possibly be another initialization of the data structures, which would be more appropriate for solving this problem.

References

[1] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 2008.

[2] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *Association for Computing Machinery*, 2004.

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

[4] Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123:155 – 225, 2002.

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

[6] G. Dantzig and M. Thapa. Linear programming 1: Introduction. *Springer*, 1997.

[7] Daniel Prusa. Graph-based simplex method for pairwise energy minimization with binary variables. *Conference: IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[8] Ekaterina Tiuzhina. Application of simplex algorithm for submodular discrete energy minimization with binary variables; bachelor thesis. 2018.

[9] Katta G Murty. Linear programming. *New York: John Wiley and Sons, Inc.*, 1983.

Appendix

Example of the input file

Examples of input file for grid of 2x2.
The first line contains two integers.
"Number of nodes" "Number of edges"
After we have next "Number of nodes" lines, where color is presented as $\{0, 1, 2\}$,where 0 means "red", 1 means "blue", 2 means "black", and costs are integer variables and line has format:
"position y" "position x" "color of label 0" "color of label 1" "cost of 0" "cost of 1"
After these lines of nodes we have "Number of edges" lines, and each of them are presented as,
"position yfrom" "position xfrom" "position yto" "position xto" "color of 10" "color of 01" "cost of 10" "cost of 01"
And full input file looks like:
4 4
0 0 2 0 0 -3
1 0 2 0 0 -4
0 1 2 1 -1 0
1 1 2 1 0 0
1 0 0 0 1 0 2 -1
1 1 1 0 0 0 -5 6
0 1 0 0 0 0 -2 4
0 1 1 1 1 0 0 1

Appendix

Contents of the enclosed CD

`Graph-BasedSimplexAlgorithmforDiscreteEnergyMinimization.pdf` . The file with pdf of BP

`Project1` ..... The direction with source files

- `graph.cpp` ..... The file, with all implemented methods for the object graph
- `graph.h` .. The file, which have initialization of all data structures, that are used in the algorithm
- `MyForm.h` The file, with application, with all posiible methods, which we used for a visualization
- `vstup3.txt` ... The input file for initialization of instance, in which we control how our algorithm works.
- `vstup4.txt` ... The input file for initialization of instance, in which we control how our algorithm works.
- `vstup5.txt` ... The input file for initialization of instance, in which we control how our algorithm works.
- `vstup6.txt` ... The input file for initialization of instance, in which we control how our algorithm works.