Karlsruhe Institute of Technology

Energy Network enhancement and stability

Diploma Thesis of

David Oertel

At the Department of Informatics
Institute for Theoretical Informatics - ITI Wagner (KIT)
and Tepper School of Business (CMU)

Reviewer:	Prof. D. Wagner
Second reviewer:	Prof. P. Sanders
Advisor:	Prof. R. Ravi
Second advisor:	Dr. Ignaz Rutter

Duration: Nov 2011 – April 2012

Eidesstattliche Erklärung
Ich erkläre hiermit, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 25. April 2012

Oertel, David

Abstract

This thesis examines the computational complexity of transmission network expansion planning which is defined as follows. Given a transmission grid of generators and consumers (nodes) connected by transmission lines (edges) with certain flow capacities, where should new lines be added at minimum cost such that all given capacity constraints are fulfilled? In its original mixed-integer non-linear programming formulation this is NP-hard since it contains the subproblem Steiner trees, the minimum cost connection of an initially unconnected set of both mandatory and optional nodes. By using electrical network theory is is shown why NP-hardness still holds when this subproblem is omitted by considering (highly) connected networks only and focussing on the satisfaction of the flow constraints. This refers to the more realistic case of extending a long working transmission grid for increased future demand. It will be achieved by showing that this case is computationally equivalent to 3-SAT, the classic satisfiability of Boolean formulas.

Additionally, is is evaluated how much effort in computation and implementation is really necessary to solve realistic scenarios in practice. The original mathematical formulation is evaluated by using an appropriate state-of-the-art mixed-integer non-linear programming solver that can guarantee to find a *global* solution.

Zusammenfassung

In dieser Diplomarbeit wird die algorithmische Komplexität von transmission network expansion planning untersucht, was folgendermaßen definiert ist. Gegeben ein Hochspannungs-Netz aus Erzeugern und Verbrauchern (abstrahiert als Knoten), die durch Stromleitungen (Kanten) verbunden sind: An welchen Stellen sollen mit möglichst geringen Kosten neue Leitungen eingefügt werden, sodass vorgegebene maximale Belastungen der Leitungen nicht überschritten werden? Dieses Problem ist in der ursprünglichen Formulierung als mixed-integer non-linear program NP-schwer, da es als Untermenge das Steinerbaumproblem enthählt, die kostengünstigste Verbindung von zunächst unverbundenen obligatorischen und optionalen Knoten. Durch Gebrauch der Theroie über elektrische Netzwerke wird gezeigt, warum das Problem weiterhin NP-schwer ist, wenn die Unterklasse von Steinerbäumen ignoriert wird, indem nur (hochgradig) zusammenhängende Graphen betrachtet werden und der Fokus auf die Einhaltung der maximalen Kantenbelastung gelegt wird. Diese Einschränkung spiegelt den realistischeren Fall wieder, wenn ein lange existierendes Hochspannungsnetz wegen eines steigenden zukünftigen Strombedarfs erweitert werden muss. Die algorithmische Schwierigkeit wird bewiesen, indem gezeigt wird, dass dieses Problem algorithmisch äquivalent zu 3-SAT ist, der klassischen Erfüllbarkeit von Boole'schen Formeln.

Darüberhinaus wird untersucht, wie viel Aufwand für Berechnung und Implementierung wirklich nötig ist, um realistische Szenarios in der Praxis zu lösen. Die ursprüngliche mathematische Formulierung wird dazu durch einen zeitgemäßen Optimierer für mixed integer non-linear programs ausgewertet, welcher in der Lage ist, garantiert *globale* Optima für jene Problemklasse zu finden.

Acknowledgements

I would like to thank my advisor Prof. R. Ravi from CMU for his continuous support, supervision and suggestions, Prof. Wagner and Dr. Ignaz Rutters from KIT for initiation and mentoring of this work, Jay Apt from CMU for valuable references, Arne Lüllmann from Fraunhofer ISI for the topic suggestion, Michael Poss from VUB/ULB for making the test data available, Dr. Matthias Oertel for initial support in electrical engineering, the NEOS project and interACT with the Baden-Württemberg Stipendium for funding this work through a scholarship program with CMU.

Contents

List of Figures

1. Introduction

Power generation and transmission is one of the most important issues of electrical engineering today. It features many challenges on each level of abstraction, from physical/technical details to mathematical backgrounds, see [WW96] or [vM06]. One of its most important aspects is the so-called Transmission Network Expansion Planning (TNEP). The aim of TNEP is to enhance an existing (electrical) transmission network such as to meet the requirements of future (increased) power demands. This may involve the installation of new power plants as well as the installation of new transmission lines at possibly low cost. By installing new transmission lines (edges) the goal is to redirect the power in such a way that certain constraints are fulfilled. One of those is the constraint to not have a particular edge transport too much power, as it may overheat otherwise. The problem arises mostly when the total supply of power in a transmission grid is increased, e.g. when a new power plant is installed to meet a rising total demand. Like most related works, this work assumes the power plants and consumers (nodes and demands/supplies) to be given externally and focuses on the installation of new transmission lines in order to meet thermal constraints. TNEP has been widely researched for years, related work can be found in Chapter 4. In this work, an abstract simplified version of TNEP is considered. In [RMGH02] this is presented as the DC[1] system.

In the major part of this work, this model is examined in terms of theoretical computational complexity. In [MPS10] it is shown that TNEP is NP-hard due to its subproblem of Steiner trees, i.e. connecting a set of (initially unconnected) terminal and optional nodes at minimal cost. In the following sections however, it is analyzed if NP-hardness of TNEP still holds when this subproblem is omitted by demanding the graph to be connected in the beginning, which reflects the problem of enhancing a transmission grid that has already been working for years. As will be shown in Chapter 3, this still yields an NP-hard problem. In fact, it will even be shown that TNEP is NP-complete.

In Chapter 2, electrical networks are introduced and the mathematical formulation of TNEP is given. As will be seen, the theory on electrical networks as found in [Bol98] can be used to examine the computational complexity of TNEP in the DC system formulation in Chapter 3. It will be shown that NP-hardness still holds for connected graphs by using a polynomial-time reduction of 3-SAT to TNEP.

Given the theoretical computational complexity of TNEP, in Chapter 4 it is then evaluated how much effort in implementation and computation is really necessary in practice

[1]direct current

to solve realistic TNEP scenarios. Many related works on TNEP feature sophisticated optimization and approximation techniques with possibly elaborate implementation cost. These techniques are mostly designed to find an approximate solution or a local optimum in order to reduce computation time. In contrast to that, in Chapter 4 it is examined how well the original mixed-integer non-linear programming (MINLP) formulation of TNEP can be solved by a general state-of-the-art MINLP solver that can guarantee to find a *global* optimum. This was done in order to see up to which point a rather small implementation effort combined with the long term "experience" of a general-purpose solver is sufficient to solve TNEP optimally and at which input sizes more sophisticated optimization techniques are inevitable to solve/approximate TNEP in a tolerable amount of time.

The original TNEP model was implemented in the General Algebraic Modelling System (GAMS) and evaluated by the Network-Enabled Optimization System (NEOS) using the Branch-and-Reduce Optimization Navigator (BARON). The evaluation was performed for some standard test cases which were also used in [MPS10].

1.1. Prerequisites

This work assumes knowledge of basic graph theory as found in [Die06], basic knowledge of boolean (3-SAT) formulas where an extensive introduction can be found in the first chapters of [Weg87]. Moreover, it is required to be roughly familiar with the fundamental complexity classes P and NP and the big-O-notation of which a thorough description can be found in [CLRS01] for example. For details concerning the physical background of electrical transmission grids see [vM06].

Table 1.1 provides an overview for the notations used throughout this work.

name	description
G	an (electrical) network
V	set of nodes from G
E	set of undirected edges (or connections) between nodes V
$\vec{E}$	E with an arbitrary (but fixed) orientation
n	number of nodes, $\lvert V\rvert$
c	$: E \to \mathbb{R}_{>0}$ conductance of the edges
b	$\in \mathbb{R}^n$ supply vector
L	Laplacian matrix of an electrical network
p	$\in \mathbb{R}^n$ voltage potentials for a network
x	$: \vec{E} \to \mathbb{R}$ an electrical current
x_{ij}	current from node i to j w.r.t x
$n_{ij}^{(0)}$	the initial number of edges between nodes i and j
$\vec{n}$	$: V^2 \to \mathbb{N}$ number of edges between node pairs
n_{ij}	the number of edges between nodes i and j w.r.t. $\vec{n}$
$\bar{n}_{ij}$	the maximum number of edges between i and j
$c_{ij}^{(0)}$	the conductance of each (i,j)-edge
w_{ij}	the cost of each (i,j)-edge added
F	a 3-SAT formula
C_j	a clause in F
m	number of clauses in F
k	number of variables in F

Table 1.1.: notation overview

1.2. Matlab and Scilab

For the analysis, both Matlab and Scilab have been used to prove the complexity of TNEP. Scilab is a free open source software for numerical computations. The syntax is very similar to the syntax used in Matlab although the scope of functionality of Scilab is generally smaller than the one of Matlab, as of the submission date of this work. Scilab is available at [Sci11]. For this work, Scilab version 5.2.1 has been used. Matlab, including licenses, is available at [MAT11]. For this work, version 2011b (7.13.0.564) has been used, together with the Matlab *Symbolic Math Toolbox*.

2. Electrical Networks and TNEP

Electrical networks offer a (theoretical) framework that can be used to formulate and analyse TNEP. At first, the definitions and properties of electrical networks will be given. Afterwards they will be used to formally introduce TNEP.

2.1. Introduction

Electrical networks are suitable to model real-world electrical transmission systems. The model used in this work corresponds with the one used in [Bol98] for mostly theoretical issues. A graph representing an electrical networks is described by a set of electrical devices (e.g. generators and consumers, modelled as nodes) that are connected by wires to carry current between the nodes in order to meet demands and supplies of the connected devices. This model omits engineering details like thermal dependencies of the wire's resistances (temperature is assumed to be constant) and assumes the current to be directed rather than alternating, where the latter is common in most electrical transmission grids. Nevertheless, this model is widely used to approximate stationary processes that occur in (high voltage) transmission grids. For physical and engineering details, see [WW96] for example. Related works concerning TNEP in practice are listed in Chapter 4.

Classical transportation networks are a common model of various applications related to transferring certain amounts of a (dimensionless) entity f (e.g. water, natural gas, packages) per second from one node to another. In these applications one is usually able to direct the flow of this entity very accurately - provided one respects the physical constraints of the pipes/edges which are usually modelled by an upper bound (capacity, pipe's diameter) of f passing the edge.

Example 1. *A classical transportation network is shown in Figure 2.1. The node s denotes the source and t denotes the sink. The edges are labelled by a tuple $(f_e, \bar{f}_e)$ representing the flow f_e on the edge and the maximum flow (capacity) $\bar{f}_e$. Furthermore, this example shows a max-flow of value 7.*

At first glance, electrical networks are also transportation networks with energy being the transported entity. However, there is one major difference between electrical and classical transportation networks: Though flow conservation, *Kirchhoff's Current law*, does hold in both types of networks, *Kirchhoff's Potential law* in general only holds in electrical networks making the energy flow uniquely dependant of conductances and supplies in the

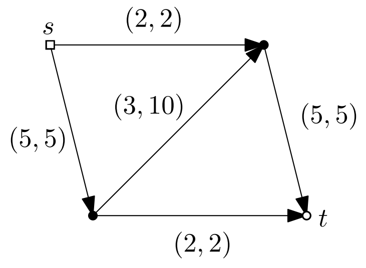

Figure 2.1.: A classical transportation network

network, ignoring possibly desired edge capacities. This property leads to a whole variety of (mathematical) problems in the engineering of energy transmission networks.

Before introducing the model, here are a few common notations from graph theory similar to those in [BE05].

For an (undirected) graph $G = (V, E)$, denote the following (for $v \in V$):

- adjacent edges $E(v) = \{e \in E \mid e \text{ is incident to } v\}$
- neighbourhood $\Gamma(v) = \{w \in V \mid w \text{ and } v \text{ share an edge}\}$
- node pairs $V^2 := \{\{v, w\} \mid v, w \in V,\ v \neq w\}$

All considered graphs are supposed to be free of loops and they usually have to be at least 1-connected (if not mentioned otherwise).

Example 2. *Figure 2.2 provides an example of a general undirected graph. With the notation above:*

$$\begin{aligned}
E(v_1) &= \{e_1, e_6, e_7\} \\
\Gamma(v_4) &= \{v_2, v_3, v_5\} \\
V^2 &= \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_1, v_5\}, \{v_1, v_6\}, \\
&\quad \{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_2, v_6\}, \{v_3, v_4\}, \\
&\quad \{v_3, v_5\}, \{v_3, v_6\}, \{v_4, v_5\}, \{v_4, v_6\}, \{v_5, v_6\}\}
\end{aligned}$$

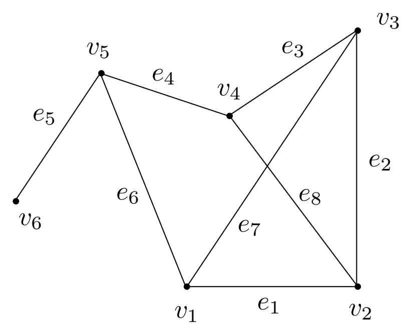

Figure 2.2.: General undirected network

2.2. Basic Definitions

The definition of an electrical network used throughout this work is basically the same as the one from [Bol98].

Definition 1 (Electrical Network). *A (classical) electrical network $G = ((V, E), c, b)$ is an undirected, loop-free (multi)-graph (V, E) with weight function $c : E \to \mathbb{R}_{>0}$ called the* conductance *and a vector $b \in \mathbb{R}^n$ with $\sum_i b_i = 0$ called* supply-vector*, where $n = |V|$.*

Each entry of b corresponds to the supply/demand of exactly one node in V, denoted by $b(v)$ for short. A vertex v with $b(v) > 0$ is called a *source* and a vertex v with $b(v) < 0$ is called a *sink*. They correspond to current entering and leaving the network. Unless mentioned otherwise, it is always required that $\sum_{i \in V} b(i) = 0$, so that current is neither "created" nor "destroyed" within the network.

Remark 1 (Notation). *Throughout this work, the different types of nodes are drawn in the following way*

□	*boxes denote sources* ($b_v > 0$)
○	*circles denote sinks* ($b_v < 0$)
●	*filled dots denote other nodes* ($b_v = 0$)

Example 3 (Electrical Network). *A small example of an electrical network can be seen in Figure 2.3. Supplies are notated in square brackets: $[+i]$ or $[-i]$, where the notation is omitted for nodes of* 0 *supply. The weights of edges denote the conductances c. This example features one source v_s and two sinks v_{t_1} and v_{t_2}.*

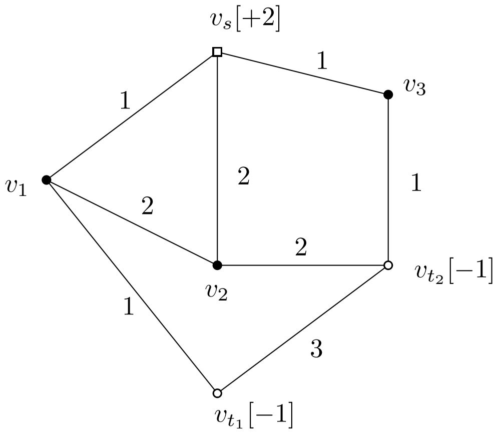

Figure 2.3.: Example of an electrical network

Though G itself is undirected, one should always imagine G to have an arbitrary but *fixed orientation* of its edges E, call this orientation $\vec{E}$. For any edge $\vec{e} \in \vec{E}$ being incident to two nodes v_1, v_2 with orientation from v_1 to v_2, call $h(\vec{e}) = v_1$ the head (initial vertex) and $t(\vec{e}) = v_2$ the tail (end vertex) of $\vec{e}$.

Definition 2. *Let $v \in V$ from $G = (V, E)$ with a fixed orientation of edges $\vec{E}$, then call: $E^-(v) := \{\vec{e} \in \vec{E} \mid t(\vec{e}) = v\}$ the incoming edges of v. $E^+(v) := \{\vec{e} \in \vec{E} \mid h(\vec{e}) = v\}$ the outgoing edges of v.*

The following definition substantiates both of Kirchhoff's laws one of which was already mentioned in section 2.1. Kirchhoff's current law (KCL) also holds for classical transportation networks and signifies that the transported entity (i.e., energy/charges) must be conserved at all nodes except for the sources and sinks, where according to the supply-vector an amount of the entity enters (positive value) or leaves (negative value) the network.

Definition 3 (Kirchhoff's laws)**.** *An (electrical) current x on an electrical network $G = ((V,E),c,b)$ is a vector $x : \vec{E} \to \mathbb{R}$ that obeys both*

- *Kirchhoff's current law (KCL): for any $v \in V$ it holds*

$$b(v) + \sum_{\vec{e} \in E^+(v)} x(\vec{e}) = \sum_{\vec{e} \in E^-(v)} x(\vec{e}) \tag{2.1}$$

- *Kirchhoff's potential law (KPL): for any (undirected) cycle $K = (e_1, e_2, \dots, e_k)$ it holds*

$$\sum_{i=1}^{k} \frac{x_K(\vec{e}_i)}{c(e_i)} = 0, \tag{2.2}$$

where $x_K(\vec{e}_i) := \begin{cases} x(\vec{e}_i), & \textit{if } K(e_i) \equiv \vec{e}_i \\ -x(\vec{e}_i), & \textit{otherwise} \end{cases}$ [1] *(for $e_i \in K$)*

See example 5 for an illustration of an electrical current which uses the same network as shown in Figure 2.3.

2.2.1. Merging parallel Edges

Since in general multi-graphs are considered here, parallel edges are allowed. For the sake of simplicity one can map any electrical network to a corresponding simple graph by combining parallel edges to one single edge using the basic laws of physics. For two adjacent vertices $i, j \in V$ denote the combined single edge by $\{i, j\}$ (or (i, j) if direction matters) and its combined conductance by $c(i, j)$ or c_{ij}. The current on (i, j) is denoted by $x(i, j)$ or x_{ij} and always refers to the actual current *from i to j*, thus $x_{ij} = -x_{ji}$.

Remark 2. *For $i, j \in V$ with $i \in \Gamma(j)$ call the combination of all edges between i and j the* connection *from i to j or* $\text{conn}(i, j)$ *for short:*

$$\text{conn}(i, j) \quad := \quad \{e \in E \mid e \in E(i) \cap E(j)\}$$

If the network is considered to have all parallel edges combined to connections, then the connections are also denoted by E for short.

If not mentioned otherwise, a connection is treated like a regular edge following the rules above. When a direction of all edges is needed, it is assumed that also the combined edges (=connections) receive an arbitrary but fixed direction.

Basic physics for $i, j \in V$ tells us that the combined conductance of parallel edges is just the sum of the individual conductances and it may also be verified for this model, see [Bol98] for details:

$$c(i, j) := c_{ij} := \sum_{e \in E(i) \cap E(j)} c(e)$$

Analogously, the combined current is just the sum of the individual currents directed from i to j:

$$x(i, j) := x_{ij} := \sum_{\vec{e} \in E^+(i) \cap E^-(j)} x(\vec{e}) \quad - \sum_{\vec{e} \in E^-(i) \cap E^+(j)} x(\vec{e})$$

[1] that means: if the (implied) direction of e_i in K coincides with the direction of $\vec{e}_i$ (the arbitrary direction of e_i in E)

Definition 4 (Merging nodes)**.** *For an electrical network $G = ((V, E), c)$ and $e \in E$ denote by:*

- *$G - e$ the network that is obtained by deleting e from G*
- *G/e the network that is obtained by contracting e (that is: merge the end-vertices of e and remove possibly arising loops)*

2.3. Basic Properties

An electrical current that obeys KPL only (and not necessarily KCL, because it usually violates the supply-vector) can be obtained by assigning every node of V a voltage potential $p : V \to \mathbb{R}$ and using Ohm's law to define a current. That is: $U \cdot c = I$ for a voltage potential difference U, conductance c and a current I.

Corollary 1 (Electrical current by Ohm's law)**.** *For any potential $p : V \to \mathbb{R}$, Ohm's law defines an electrical current on G by the equation:*

$$x_{ij} = (p_i - p_j) \cdot c_{ij} \quad \forall (i, j) \in \vec{E} \tag{2.3}$$

if and only if (iff) one sets the supply-vector b according to KCL (Equation 2.1). In particular, any such p obeys KPL.

Proof. sketch: For any cycle K with respective node potentials p, that is

$$K = (e_1, e_2, \ldots, e_k), \quad p = (p_1, p_2, \ldots, p_k, p_1)$$

which leads to a telescoping series

$$\begin{aligned} \sum_{i=1}^{k} \frac{x_K(\vec{e}_i)}{c(e_i)} &= \sum_{i=1}^{k} (p_i - p_{i+1}) \\ &= p_1 - p_1 = 0 \end{aligned}$$

Thus, any potential obeys KPL. The other part follows immediately from the definition of KCL. □

On the other hand, given a supply-vector b a proper electrical current obeying both KCL and KPL can be found by using the Laplacian matrix of G. For that, identify each node uniquely with a fixed number between 1 and n, so V is represented by its indices $V = \{1, 2, \ldots, n\}$.

Definition 5 (Laplacian matrix)**.** *The Laplacian matrix $L \in \mathbb{R}^{n \times n}$ of an electrical network $G = ((V, E), c)$ with $|V| = n$ is defined as*

$$L_{ij} := \begin{cases} \sum_{k \in \Gamma(i)} c_{ik}, & \text{if } i = j \\ -c_{ij}, & \text{if } i \in \Gamma(j) \text{ (and } i \neq j\text{)} \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

Example 4 (Laplacian matrix)**.** *Table 2.1 shows the Laplacian matrix of the network seen in Figure 2.3.*

Note that if D is the diagonal degree matrix of G and A is the adjacency matrix (both weighted with the conductances of the edges instead of the mere degrees), then $L = D - A$.

	v_s	v_{t_1}	v_s	v_1	v_2	v_3
v_s	4	0	0	−1	−2	−1
v_{t_1}	0	4	−3	−1	0	0
v_{t_2}	0	−3	6	0	−2	−1
v_1	−1	−1	0	4	−2	0
v_2	−2	0	−2	−2	6	0
v_3	−1	0	−1	0	0	2

Table 2.1.: Laplacian matrix of the network in Figure 2.3

Theorem 1. *For an electrical network $G = ((V,E),c)$ one can find a proper electrical current x given a supply-vector b by solving the linear equation*

$$L \cdot p = b \tag{2.5}$$

and defining an electrical current x by Ohm's law $x_{ij} = (p_i - p_j) \cdot c_{ij}$ for all $(i,j) \in \vec{E}$.

Proof. sketch: According to Corollary 1 p obeys KPL and, by that, so does x. By rearranging, each line of Equation 2.5 represents exactly KCL. Since there is one such equation for each node in V, x obeys KCL. Hence, x is an electrical current. □

Example 5. *Using the electrical network of example 3 and Figure 2.3, Figure 2.4 shows a proper electrical current on the network (values rounded). Round brackets behind the node names denote voltage potentials. The labels on the edges denote currents with respect to the (arbitrarily chosen) direction of the edges which can be seen by the arrow heads. In*

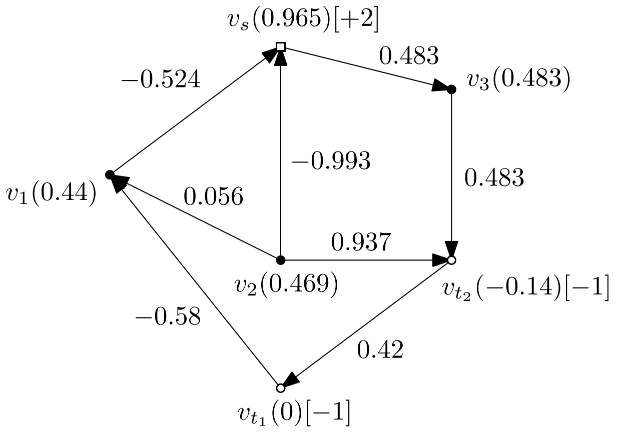

Figure 2.4.: Voltage potentials and resulting currents (rounded)

this example, according to Corollary 2 the node v_{t_1} was chosen to have a 0 *potential.*

Remark 3. *Call a vector p solving Equation 2.5 a* (valid) voltage potential *with respect to b.*

Recall that the graph is assumed to be at least 1-connected. Otherwise the following is valid for the graph's connectivity components.

Corollary 2 (Uniqueness)**.** *Given an arbitrary (1-connected) electrical network $G = ((V,E),c)$. For each supply-vector b ($\sum_{i \in V} b(i) = 0$) it holds:*

- *there is exactly one distribution of currents x satisfying both KPL and KCL.*

- *p solves Equation 2.5, iff (=if and only if) $p' = p + d \cdot (1, \dots, 1)^{\mathrm{T}}$ also solves the equation for all $d \in \mathbb{R}$*
- *when fixing one component of p, the equation $L \cdot p = b$ has exactly one solution.*

The proof follows from $(1, \dots, 1)^{\mathrm{T}} \in \ker(L)$ and from the 1-connectivity of (V, E). For details see [Big93].

Observation 1 (principle of superposition). *Electrical currents follow the principle of superposition. That is: Given an electrical network $G = ((V, E), c)$ and supply vectors $\vec{b}_1, \dots, \vec{b}_\kappa$*[2] *with respective electrical currents $\vec{x}_1, \dots, \vec{x}_\kappa$ on G, then the (weighted) sum of currents $x := \sum_{i=1}^{\kappa} \alpha_i \vec{x}_i$ serves as the (unique) electrical current of $b = \sum_{i=1}^{\kappa} \alpha_i \vec{b}_i$, where $\alpha_i \in \mathbb{R}$.*

This observation follows immediately from voltage potentials p_i associated with the $\vec{x}_i$ and the linear Equation 2.5. Thus, the total current can be found by finding currents for reduced single-source-single-target networks, where all other sources and sinks of the original graph are treated as nodes with zero supply. When multiple pairs (s_i, t_j) of sources and sinks are chosen as a partition of the total supply, the total current is the sum of the single currents from s_i to t_j.

Definition 6 (Energy of a network). *The total energy of an electrical network $G = ((V, E), c, b)$ with current x is defined as*

$$\begin{aligned} E(x) &= \sum_{\vec{e} \in \vec{E}} \frac{x(\vec{e})^2}{c(e)} && (2.6) \\ &= \sum_{(i,j) = \vec{e} \in \vec{E}} (p_i - p_j)^2 \cdot c(e) && (2.7) \\ &= \sum_{(i,j) = \vec{e} \in \vec{E}} (p_i - p_j) \cdot x(\vec{e}) && (2.8) \\ & && (2.9) \end{aligned}$$

Given the physical background, the term *power* instead of *energy* would be more appropriate here, but it does not matter in this context. The measured energy/power is related to the dissipated power within the network. See [vM06] for details of dissipated power.

The following Theorem states that electrical currents are distributed in such a way that the *total energy* of the network is minimized.

Theorem 2 (Thomson's principle). *Let $G = ((V, E), c, b)$ be an electrical network (so especially $\sum_v b_v = 0$). For any flow x satisfying KCL (and not necessarily KPL) consider the energy function*

$$E(x) = \sum_{\vec{e} \in \vec{E}} \frac{x(\vec{e})^2}{c(e)}$$

There is such a flow x that minimizes $E(x)$. This x also satisfies KPL, so it is an electrical current.

The proof of Theorem 2 can be found in [Bol98], chapter IX.1.

[2] so especially $\sum_{v \in V} \vec{b}_i(v) = 0$ for $i = 1, \dots, \kappa$

2.4. Advanced Properties

Symmetry of the network is often used for both analysis and simplification, see for example [BB09]. Symmetry allows to short-cut and identify two formerly distinct nodes for the sake of simplicity. In the following paragraphs, these techniques are demonstrated and proven for the model used so far.

2.4.1. Symmetry

Assume for now that parallel edges are combined to one single edge by adding their conductances.

Definition 7 (Symmetry). *For an electrical network $G = ((V, E), c, b)$, two nodes $v_1, v_2 \in V$ are called* symmetric *iff there exists a permutation $\pi : (V, E) \to (V, E)$ of nodes and connections (with $\pi(V) = V$ and $\pi(E) = E$) such that for each connection $\{v, w\} \in E$:*

$$\pi(v_1) = v_2 \tag{2.10}$$
$$\pi(v_2) = v_1 \tag{2.11}$$
$$\pi(\{v, w\}) = \{\pi(v), \pi(w)\} \tag{2.12}$$
$$c(v, w) = c(\pi(v), \pi(w)) \tag{2.13}$$
$$b(v) = b(\pi(v)) \quad \forall v \in V \tag{2.14}$$

Thus, the two nodes play similar roles within the networks.

Example 6 (Node symmetry). *Figure 2.5 shows a network where the nodes v_1, v_2, v_3 are symmetric by Definition 7. Conductances are given within round brackets. A permutation π that shows the symmetry of node v_1 and v_2 is given by simply switching the appearance of v_1 and v_2 in each element of the network:*

$$\begin{aligned}
\pi((v_1, v_2, v_3)) &:= (v_2, v_1, v_3)\\
\pi((e_{s,v_1}, e_{s,v_2})) &:= (e_{s,v_2}, e_{s,v_1})\\
\pi((e_{v_1,v_3}, e_{v_2,v_3})) &:= (e_{v_2,v_3}, e_{v_1,v_3})\\
\pi((e_{v_1,t}, e_{v_2,t})) &:= (e_{v_2,t}, e_{v_1,t})\\
\pi &\equiv id \quad \textit{for all other nodes and edges}
\end{aligned}$$

The symmetry of v_1 to v_3 and v_2 to v_3 can be shown in the same way.

Definition 7 provides the following lemma using the Laplacian matrix L of G.

Lemma 1. *For two symmetric nodes $v_1, v_2 \in V$ there exists a permutation matrix $Q \in \{0, 1\}^{n\times n}$ that permutes v_1 and v_2 and for which it holds:*

$$Q \cdot L \cdot Q^{\mathrm{T}} = L \tag{2.15}$$

Proof. The symmetry from 7 gives a (not necessarily unique) permutation $\pi \in \mathrm{perm}(n)$ that instantly defines a permutation matrix $Q = Q(\pi)$ of the nodes by using the node ordering $v_1, \ldots, v_n$ of the Laplacian matrix L. The only thing that needs to be shown now is that this Q satisfies Equation 2.15. Recall the following property of permutation matrices: For $A \in \mathbb{R}^{n\times n}$ with a_i being the rows of A, Q multiplied from the left permutes the rows of A, such that $Q \cdot A = A'$ with $a'_i = a_{\pi(j)}$. Simultaneously, Q^{T} multiplied to a matrix A from the right will permute A's columns in the fashion of π. With $L = (\lambda_{i,j})_{i=1,\ldots,n,\, j=1,\ldots,n}$:

$$\begin{aligned}
Q \cdot L \cdot Q^{\mathrm{T}} &= (Q \cdot L) \cdot Q^{\mathrm{T}}\\
&= (\lambda_{\pi(i),j})_{i,j} \cdot Q^{\mathrm{T}} = (\lambda_{\pi(i),\pi(j)})_{i,j}
\end{aligned}$$

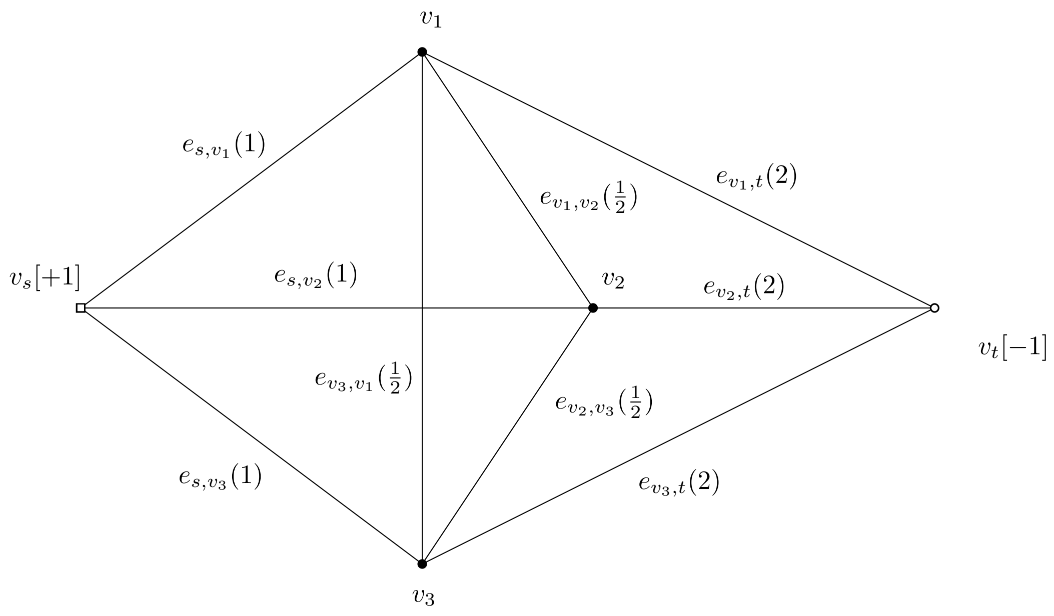

Figure 2.5.: Network with three symmetric nodes v_1, v_2, v_3

(I) Now let $i, j \in \{1, \dots, n\}$ with $i \neq j$. By Definition 5, it holds:

$$\begin{aligned} \lambda_{i,j} &= -c_{i,j} \\ \overset{2.12,2.13}{\Rightarrow} \quad -c_{\pi(i)\pi(j)} &= -c_{ij} \\ \Rightarrow \quad \lambda_{\pi(i),\pi(j)} &= \lambda_{i,j} \end{aligned}$$

(II) For $i = j$ it holds:

$$\begin{aligned} \lambda_{i,i} &= \sum_{\kappa=1,\kappa\neq i}^{n} c_{i,\kappa} = -\sum_{\kappa=1,\kappa\neq i}^{n} \lambda_{i\kappa} \\ &\overset{(I)}{=} -\sum_{\kappa=1,\kappa\neq i}^{n} \lambda_{\pi(i)\pi(\kappa)} = -\sum_{\kappa=1,\kappa\neq\pi(i)}^{n} \lambda_{\pi(i)\kappa} \\ &\overset{\text{Def.}}{=} \lambda_{\pi(i)\pi(i)} \end{aligned}$$

Thus, from (I) and (II) follows the claim $Q \cdot L \cdot Q^{\mathrm{T}} = L$ for the $Q = Q(\pi)$ as defined above. □

Notice that Lemma 1 can solely be proven by using a permutation of nodes that obeys Equation 2.12 and 2.13. The other properties from Definition 7 are only needed to deduce the following.

Corollary 3. *If nodes $v_1, v_2 \in V$ are symmetric, then they have the same voltage potential $p_{v_1} = p_{v_2}$ in any solution to KCL and KPL, Equation 2.5.*

Proof. Due to symmetry, Lemma 1 gives a permutation matrix Q for which Equation 2.15 holds. Let p solve the equation $L \cdot p = b$. Denote the voltage potentials of v_1, v_2 (w.r.t. p) by p_{v_1}, p_{v_2}. Then:

$$\begin{aligned} L \cdot p &= b \\ \overset{2.15}{\Rightarrow} Q \cdot L \cdot Q^{\mathrm{T}} \cdot p &= b \\ \Rightarrow Q^{\mathrm{T}} \cdot Q \cdot L \cdot Q^{\mathrm{T}} p &= Q^{\mathrm{T}} \cdot b \\ \Rightarrow L \cdot p' &= b' \quad (\text{for } p' = Q^{\mathrm{T}} p,\ b' = Q^{\mathrm{T}} \cdot b) \end{aligned}$$

Moreover Equation 2.14 yields:

$$Q \cdot b = b \quad \Rightarrow \quad b = Q^{\mathrm{T}} \cdot b \quad \Rightarrow \quad b' = b$$

Since p' also solves the equation $Lp = b$, by Corollary 2 it follows that $p' = p + d \cdot (1, \dots, 1)^{\mathrm{T}}$ for a constant $d \in \mathbb{R}$. Together with $Q \cdot d \cdot (1, \dots, 1)^{\mathrm{T}} = d \cdot (1, \dots, 1)^{\mathrm{T}}$, this yields:

$$\begin{array}{rcl} p' & = & p + d \cdot (1, \dots, 1)^{\mathrm{T}} \\ \Rightarrow \quad Q \cdot p' & = & Q \cdot p + d \cdot Q \cdot (1, \dots, 1)^{\mathrm{T}} \\ \Rightarrow \quad p & = & Q \cdot p + d \cdot (1, \dots, 1)^{\mathrm{T}} \\ \Rightarrow \quad p_i & = & p_{\pi(i)} + d \quad \text{for } i = 1, \dots, n \\ \Rightarrow \quad \Sigma_{i=1}^{n} p_i & = & \Sigma_{i=1}^{n} p_{\pi(i)} + n \cdot d \\ \Rightarrow \quad d & = & 0 \\ \Rightarrow \quad p & = & p'. \end{array}$$

So it especially holds that $p_{v_1} = p_{\pi(v_1)} = p_{v_2}$. Since p was an arbitrary solution of Equation 2.5, the claim follows immediately. □

2.4.2. Merging Nodes

From a physical point of view, if two nodes v_1, v_2 of an electrical network share the same voltage potential $p_{v_1} = p_{v_2}$, they can be short-cut by a wire of arbitrary conductance which on the one hand will not have any current running through it (following Ohm's law (Equation 2.3) $x_{v_1,v_2} = c_{v_1,v_2} \cdot (p_{v_1} - p_{v_2}) = 0$) and on the other hand will not change the currents within the rest of the network.

Lemma 2 (Nodes of equal potential)**.** *Let $G = ((V, E), c, b)$ be an electrical network with a valid voltage potential p. Let $v_1, v_2 \in V$ have the same potential $p_{v_1} = p_{v_2}$. Then by adding an edge e_{v_1,v_2} of arbitrary conductance between v_1 and v_2, the valid voltage potential (and thus the currents) on G will be the same*[3] *as without the edge.*

The following example illustrates the lemma.

Example 7. *Figure 2.6 shows an example of two nodes v_1, v_2 having the same potential although they are not symmetric in the sense of Definition 7 since the adjacent edges of v_1 can not be mapped to the adjacent edges of v_2 without violating Equation 2.13. The dashed line hints to an edges that can be inserted without any effect on the currents. (Node potentials are again denoted by round brackets behind the node names whereas edge labels denote conductance and current (c, x_{ij})).*

Proof of Lemma 2. Let G_0 be the original network and G_1 the one with an edge e_{v_1,v_2} inserted. Further, let $\widehat{p}$ and $\tilde{p}$ be valid voltage potentials on G_0 and G_1 respectively. Let $\widehat{p}_{v_1} = \widehat{p}_{v_2}$ and assume now, that $\tilde{p}_{v_1} \neq \tilde{p}_{v_2}$. By Equation 2.7, we get for the energy E_{G_i}:

$$\widehat{p} \in \arg\min_{p} E_{G_0}(p) \quad \text{and } \tilde{p} \in \arg\min_{p} E_{G_1}(p)$$

and by Thomson's principle (Theorem 2):

$$\begin{array}{rcl} E_{G_1}(p) & = & \displaystyle\sum_{(i,j)=\vec{e}\in\vec{E}_1} (p_i - p_j)^2 \cdot c(e) \\ & = & \displaystyle\sum_{(i,j)=\vec{e}\in\vec{E}_0} (p_i - p_j)^2 \cdot c(e) + (p_{v_1} - p_{v_2})^2 \cdot c(e_{v_1,v_2}) \\ & = & E_{G_0}(p) + (p_{v_1} - p_{v_2})^2 \cdot c(e_{v_1,v_2}) \end{array}$$

[3]up to an additive constant as in Corollary 2

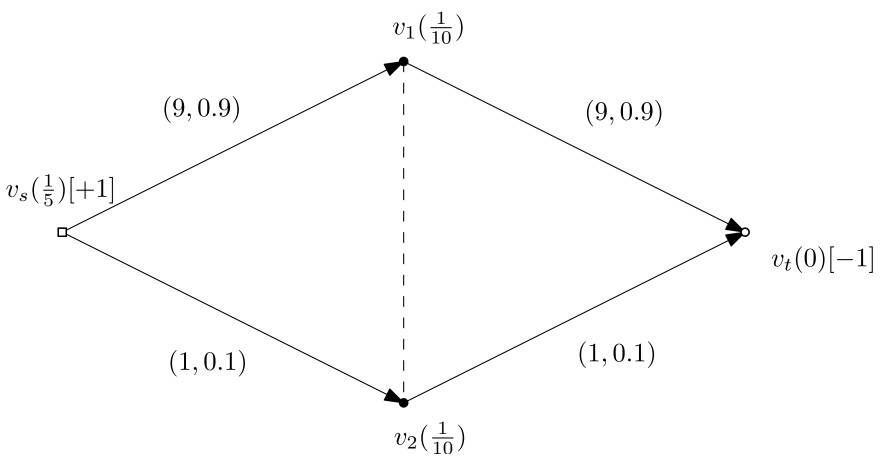

Figure 2.6.: Two nodes of same potential v_1, v_2 (not symmetric by Def. 7)

Since due to Equation 2.3 $\widehat{p}$ also defines a voltage potential following both KCL and KPL on G_1, we may assume that $\min_p E_{G_1}(p) \leq \min_p E_{G_0}(p)$. Thus:

$$\begin{aligned} E_{G_0}(\tilde{p}) + (\tilde{p}_{v_1} - \tilde{p}_{v_2})^2 \cdot c(e_{v_1,v_2}) &= E_{G_1}(\tilde{p}) \\ = \min_p E_{G_1}(p) &\leq \min_p E_{G_0}(p) = E_{G_0}(\widehat{p}) \\ \overset{\tilde{p}_{v_1} \neq \tilde{p}_{v_2}}{\Rightarrow} \quad E_{G_0}(\tilde{p}) &< E_{G_0}(\widehat{p}) \end{aligned}$$

Which contradicts the prerequisite that $\widehat{p} \in \arg\min_p E_{G_0}(p)$, thus the assumption is false and the original claim follows: $\tilde{p}_{v_1} = \tilde{p}_{v_2}$. □

Since nodes of equal potential can be connected without changing the currents in the network, one can virtually connect these nodes by an edge of infinite conductance merging the nodes to one single node, as the next Theorem will show.

Theorem 3 (Identifying and merging nodes)**.** *For $G = ((V, E), c, b)$ with a valid voltage potential p let $v_1, v_2 \in V$ have the same potentials $p_{v_1} = p_{v_2}$. Let $\tilde{G} = G/\{v_1, v_2\}$ be the network obtained by merging v_1 and v_2 to a node called $\tilde{v}$. Then p instantly defines a valid voltage potential on $\tilde{G}$ by* $\tilde{p}_v := \begin{cases} p_v, & \text{if } v \neq \tilde{v} \\ p_{v_1}, & \text{if } v = \tilde{v} \end{cases}$

Proof. The theorem follows by Thomson's principle, theorem 2, by which the total energy of G is

$$E_G(x) = \sum_{\vec{e} \in \vec{E}} \frac{x(\vec{e})^2}{c(e)} \tag{2.16}$$

$$= \sum_{\vec{e} \in \vec{E} \setminus (\vec{E}(v_1) \cap \vec{E}(v_2))} \frac{x(\vec{e})^2}{c(e)} + \sum_{\vec{e} \in (\vec{E}(v_1) \cap \vec{E}(v_2))} \frac{x(\vec{e})^2}{c(e)} \tag{2.17}$$

Since $p_{v_1} = p_{v_2}$ it follows that $x(\vec{e}) = 0$ for all $\vec{e} \in \vec{E}(v_1) \cap \vec{E}(v_2)$. Hence both G and $\tilde{G}$ have the same total energy which would immediately lead to a contradiction of Thomson's principle if p did not contribute a valid voltage potential on $\tilde{G}$. □

The previous results describe powerful techniques that are widely used among electrical engineers in order to simplify and analyse electrical circuits. The theorems stated here justify mathematically the use of these techniques within electrical networks as they were defined in this chapter. This will be useful in the following sections.

2.5. Transmission Network Expansion Planning (TNEP)

As already mentioned in the Introduction, Chapter 1, TNEP examines the problem of adding new transmission lines in order to maintain a reliable operating network at a possibly low investment cost. The problem arises especially from an increased future power demand. The reliability constrains considered here are thermal constraints (capacities) exclusively. A comprehensive introduction to power generation and transmission can be found in [vM06]. This book is especially useful for non-professionals in the field of electrical engineering with an academic background.

TNEP in real world applications needs to take into account a lot of technical and mathematical considerations that go beyond the scope of this work. However, as can be seen in the papers cited in Chapter 4, the model presented here is the one of most frequently used abstractions in TNEP research.

Consider a simplified version of TNEP in which every edge is supposed to carry at most a certain amount of current[4].

Definition 8 (TNEP Scenario). *Let (V, b) be a set of nodes and a supply-vector together with a function $N : V^2 \longrightarrow (\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{R}_{>0} \times \mathbb{R}_{>0})$, $\{i, j\} \mapsto (n^{(0)}_{ij}, \bar{n}_{ij}, c^{(0)}_{ij}, w_{ij})$[5] denoting for each pair $\{i, j\}$ of nodes:*

- *$n^{(0)}_{ij}$ the initial number of edges between i and j*
- *$\bar{n}_{ij}$ the maximum number of edges between i and j ($n^{(0)}_{ij} \leq \bar{n}_{ij}$)*
- *$c^{(0)}_{ij}$ the conductance of each (i, j)-edge*
- *w_{ij} the cost of each (i, j)-edge added*

A TNEP Scenario *is a tuple $G = (V, b, N, \bar{f})$ with $\bar{f} : V^2 \longrightarrow \mathbb{R}_{>0}$ denoting an upper bound on the current per edge.*

This model assumes all edges of each connection to have the same conductance $c^{(0)}_{ij}$.

Example 8 (Garver's network). *An example of a TNEP scenario is shown in Figure 2.7. This network was initially introduced to TNEP in [Gar70]. The corresponding data is available in [RMGH02]. The connection data according to Definition 8 is given in Table 2.2. Notice that this example contains an initially isolated node v_6.*

Definition 9 (TNEP instance). *Given a TNEP Scenario $G = (V, b, N, \bar{f})$ as in Definition 8, a vector $\vec{n} : V^2 \longrightarrow \mathbb{N}_0$ is called a* TNEP instance *of G iff $\vec{n}$ defines a valid set of edges for each $\{i, j\} \in V^2$, that is iff:*

$$n^{(0)}_{ij} \leq n_{ij} \leq \bar{n}_{ij} \tag{2.18}$$

Furthermore, define the $cost_G(\vec{n})$ of an instance $\vec{n}$ as the number of added edges times the cost per edge, that is:

$$cost_G(\vec{n}) = \sum_{\{i,j\} \in V^2} (n_{ij} - n^{(0)}_{ij}) \cdot w_{ij} \tag{2.19}$$

[4]the physical quantities appearing in the real TNEP are different, but the equations and constraints are the same within this model

[5]Notice that by this definition $n^{(0)}_{ij} = n^{(0)}_{ji}$, $\bar{n}_{ij} = \bar{n}_{ji}$ etc.

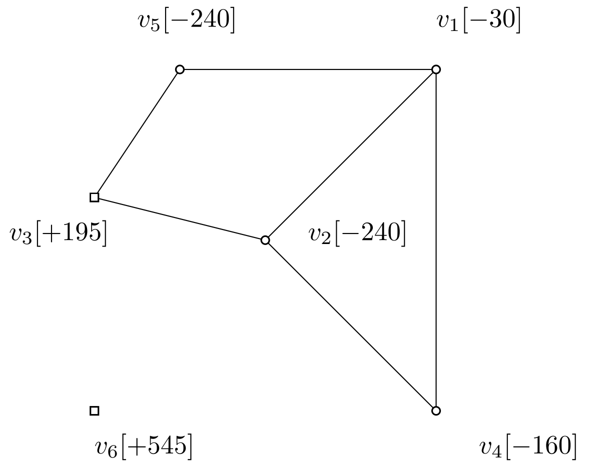

Figure 2.7.: Initial set-up of Garver's network

i	j	$\frac{1}{c_{ij}^{(0)}}$	$\bar{f}_{ij}$	w_{ij}	$n_{ij}^{(0)}$	$\bar{n}_{ij}$
1	2	0.40	100	40	1	5
1	4	0.60	80	60	1	5
1	5	0.20	100	20	1	5
2	3	0.20	100	20	1	5
2	4	0.40	100	40	1	5
3	5	0.20	100	20	1	5
1	3	0.38	100	38	0	4
1	6	0.68	70	68	0	4
2	5	0.31	100	31	0	4
2	6	0.30	100	30	0	4
3	4	0.59	82	59	0	4
3	6	0.48	100	48	0	4
4	5	0.63	75	63	0	4
4	6	0.30	100	30	0	4
5	6	0.61	78	61	0	4

Table 2.2.: Connection data of Garver's network

Definition 10 (TNEP Solution)**.** *Given a TNEP Scenario $G = (V, b, N, \bar{f})$ and a TNEP instance $\vec{n}$ as in Definition 9, call $\vec{n}$ a* TNEP Solution *iff for a valid voltage potential p on G, it holds for all $\{i, j\} \in V^2$*

$$|(p_i - p_j)| \cdot (n_{ij} \cdot c_{ij}^{(0)}) \leq n_{ij}\bar{f}_{ij} \tag{2.20}$$

Thus, a TNEP solution is an instance which fulfils a set of given constraints on the maximum currents (power) running through the edges.

Example 9. *Garver's network (Figure 2.7) can be solved by the instance shown in Figure 2.8. Added edges are denoted by dashed lines. With the connection alignment of Table 2.2, this corresponds to a vector $\vec{n} = (1, 1, 1, 1, 1, 2, 0, 0, 0, 4, 0, 0, 0, 2, 0)$ with 7 edges added at total a cost of 200.*

In the following the "natural" mathematical problems arising from the scenario of Definition 8 are stated.

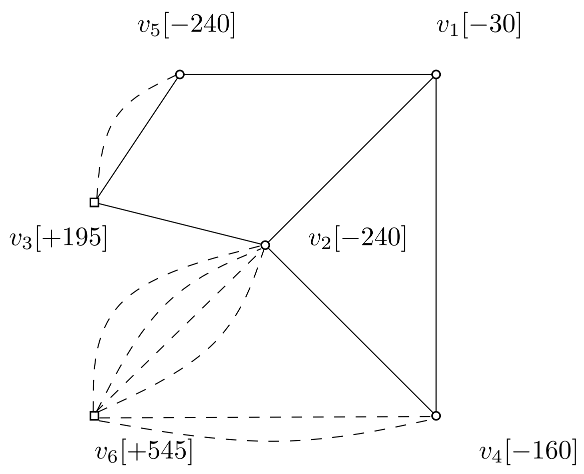

Figure 2.8.: Solution to Garver's network

Definition 11 (TNEP problems)**.** *For each TNEP Scenario $G = (V, b, N, \bar{f})$ as in Definition 8 denote by*

- Decision problem*: Is there a (TNEP) solution $\vec{n}$ to G with $cost_G(\vec{n}) \leq k$*
- Optimization problem*: Which solution $\vec{n}$ has minimal cost*

Lemma 3. *The decision problem of Definition 11 is in NP.*

Proof. Given a vector $\vec{n}$ and a cost k one can compute all the necessary steps to check if $\vec{n}$ is a solution of certain cost in deterministic polynomial time. In particular,

1. checking if $\vec{n}$ is an instance: $n_{ij}^{(0)} \leq n_{ij} \leq \bar{n}_{ij}$ takes $O(|V|^2)$ steps,
2. checking if $cost_G(\vec{n}) \leq k$ takes $O(|V|^2)$ steps,
3. computing the Laplacian matrix takes $O(|V|^2)$ steps,
4. solving for p takes $O(|V|^3)$ steps and
5. checking the current constraints 2.20 by Ohm's law takes $O(|V|^2)$ steps.

□

2.5.1. Non-monotonic behaviour

This section provides some more observations that will help to examine the complexity of TNEP.

Lemma 4. *Consider an electrical network $G = ((V, E), c)$ with supply b, with sources $S \subset V$ and sinks $T \subset V$ ($b_i = 0$ for all other nodes). Let p solve the equation $L \cdot p = b$, then the total energy of the network sums up to:*

$$E(p) = \sum_{s \in S} p_s \cdot b_s + \sum_{t \in T} p_t \cdot b_t \tag{2.21}$$

The proof is just a generalization of a similar lemma in [Bol98].

Let us consider a single-source-single-sink electrical network $G = ((V, E), c)$ with unique source s and unique sink t and supply b. For simplicity, let's fix the voltage potential p_t of t to $p_t = 0$.

Lemma 5 (min and max potential)**.** *For the voltage potential p defining the currents in G according to b, it holds:*

$$p_s = \max_{i \in V}\{p_i\}$$
$$p_t = \min_{i \in V}\{p_i\}$$

Proof. Assume there is a node $v \in V$ with $p_v > p_s$. Without restriction, assume $p_v = \max_{i \in V}\{p_i\}$ and v has at least one neighbouring node with smaller potential[6]. Then KCL provides: $\sum_{w \in \Gamma(v)} (p_v - p_w)c_{vw} > 0$ since $p_v \geq p_w$ for all neighbours and $p_v > p_w$ for at least one neighbour. Thus, v itself is a source, contradicting the assumption. The second part of the proof uses the other direction. □

By Rayleigh's principle of monotonicity (see [Bol98]), the effective conductance does *not* decrease if conductance in the network is increased or added. Thus, by Thomson's principle it holds:

Lemma 6. *Let $G = ((V, E), c)$ be an electrical network and let $\widehat{G} = ((V, E), \widehat{c})$ be the same network on which one arbitrary connection has more conductance, i.e. $c_{ij} < \widehat{c}_{ij}$ for one pair $i, j \in V$ and $c_{xy} = \widehat{c}_{xy}$ for all $\{x, y\} \neq \{i, j\}$. For a fixed supply b and corresponding current x on G and $\widehat{x}$ on $\widehat{G}$, it holds:*

$$E(x) \geq E(\widehat{x}) \tag{2.22}$$
$$E(x) > E(\widehat{x}) \iff x_{ij} \neq \widehat{x}_{ij} \tag{2.23}$$

The proof of this follows from a variation of [Bol98] Theorem 2 in chapter IX.1, page 300.

Using Equation 2.21 for a fixed $b_t = b_s$ (single-source-single-sink) leads to the following.

Corollary 4. *In the setting of lemma 6 with fixed $p_t = 0 = \widehat{p}_t$ (single-source-single-sink) it holds:*

$$p_s \geq \widehat{p}_s \tag{2.24}$$
$$p_s > \widehat{p}_s \iff x_{ij} \neq \widehat{x}_{ij} \tag{2.25}$$

Thus, for every added edge, the voltage potential of the source will drop if the newly added edge is run through by current.

Since $p_s \geq p_i \geq p_t = 0$, it may seem as if all voltage potentials tend to decrease componentwise. However, this is not true in general, illustrated by the following example.

Example 10 (Non-monotonicity)**.** *Figure 2.9 shows a network with potentials and currents before adding a new edge (i.e., increasing the conductance) between v_1 and v_2.*

Figure 2.10 shows the same network after adding an edge. Note the following things:

- *the voltage potential of v_2 has increased[7] from $\frac{1}{2}$ to $\frac{6}{11}$*
- *the (absolute) current from v_2 to v_t has increased from $\frac{1}{2}$ to $\frac{6}{11}$*
- *the voltage potential of p_s has decreased from $\frac{3}{2}$ to $\frac{15}{11}$*

Thus, also the (absolute) currents on edges do not necessarily decrease monotonically by adding new edges. Moreover, the total energy of the network has also decreased from $\frac{3}{2}$ to $\frac{15}{11}$.

[6] otherwise the graph can not be 1-connected

[7] compared to the minimum potential $p_t = 0$

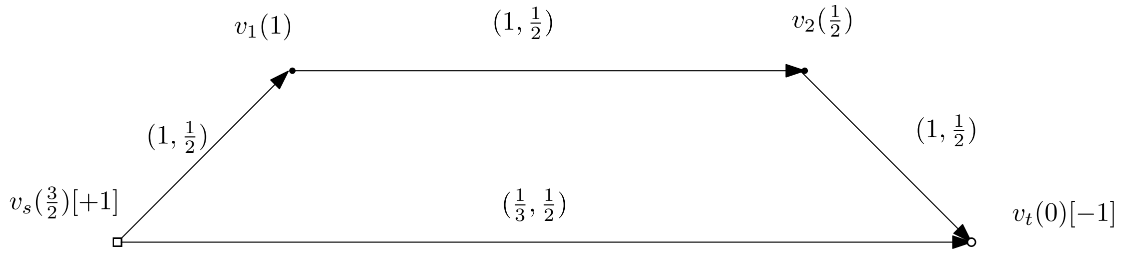

Figure 2.9.: Network before adding an edge

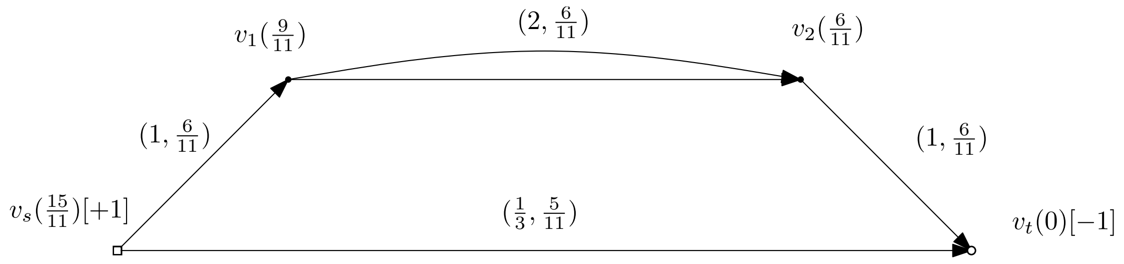

Figure 2.10.: Figure 2.9 after adding an edge

3. Complexity of TNEP

In [MPS10] TNEP is shown to be NP-hard by reducing Steiner trees to TNEP. This is done by taking a Steiner tree instance and defining a TNEP Scenario by assigning all terminal nodes a power-supply and assigning all edges that can be inserted a (trivial) current constraint. The solution to the TNEP instance is then achieved on behalf of having to connect the entire network (with edges of summed minimum weight) in order to fulfil the supply of each terminal node ($b_v \neq 0$). Thus, TNEP contains the NP-hard subproblem of Steiner trees. However, this does not immediately involve the original aim of relieving edges by enhancing the network, i.e., obeying current constraints on the edges. So it should be examined, if NP-hardness still holds when the subproblem of connecting the terminal nodes is omitted, e.g., by demanding the network to be sufficiently connected in the first place.

This section examines the complexity of TNEP for rather realistic scenarios by assuming the networks to be at least 1-connected[1], such that all prerequisites to the power/current supplies are already fulfilled in advance. This refers to the case when an existing power grid has been operating for years but needs to be enhanced now to meet an increased demand and supply in the future.

As we will see by the end of this chapter, the restriction to "realistic" scenarios (highly connected graphs) of TNEP is NP-complete. This will be shown by a polynomial-time reduction of 3-SAT to TNEP.

3.1. Basic Ideas

As seen in the previous sections, electrical network may behave in a somewhat non-intuitive non-monotonic fashion that makes it difficult to predict the results of adding certain edges with respect to the TNEP problem. Since symmetry as in Definition 7 provides a powerful tool to analyse and simplify electrical networks, it is most convenient to use a highly symmetrical and thus predictable network that can be dealt with by these methods.

Given a boolean formula, we find a bijective mapping between satisfying a formula by setting literals and reducing the (absolute) current on an edge by adding new edges. This reduction from 3-SAT to TNEP is achieved using two key steps

- translate "satisfaction of a 3-SAT formula" to "obeying current-constraints"

[1]the network used in the analysis is in fact 2-connected

- translate “setting boolean variable exclusively to true or false” to “adding one of two possible edges exclusively”

One idea to achieve the first of these steps is illustrated by Figure 3.1: current entering the network in s_C via e_1 into node v_1 is running to the sink node t_C via two paths - via the direct route e_2 and via an alternate path over e_3 and e_4 using v_2. If one enhances the alternate path by increasing its conductance (i.e. adding the dashed edge and not allowing addition of any other edge), by Thomson’s principle more current will run over that alternate path, which will eventually reduce the current on e_2. This leads to the

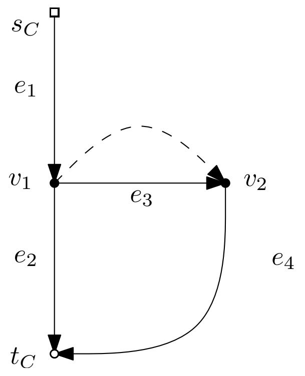

Figure 3.1.: Redirecting current to t_C via v_2

following observation:

Observation 2. *In Figure 3.1: by identifying the* adding of the dashed edge *with* setting a boolean variable *to true or false and by identifying the* satisfaction of a clause *with requiring the* current on e_2 to go under a certain limit*, it is possible to achieve the first step of the mapping from* 3*-SAT to TNEP.*

Continuing with Figure 3.1, each node of the type of T_C together with the edge e_2 shall be called a *clause-block* and may represent a clause of a formula. For that, t_C is connected to nodes that represent the *associated literals* (x_i or $\neg x_i$ etc.), such that when one or more of the literals is activated (i.e., an edge is added), the clause benefits of this by receiving more current via the edge connected to the literal than via e_2, so a certain constraint on e_2 is obeyed.

To achieve the second step of the bijection, it is necessary to model the exclusive choice of setting boolean variables x_i to either true or false. So far, adding an edge only corresponds to activating a certain literal so it could be possible to enhance both of the connections that correspond to x_i and to $\neg x_i$. It has to be made sure that the one excludes the other. Since the general TNEP scenario does not allow restrictions like “add at most k edges in total but only 1 edge in this sub-graph”, the restriction needs to be made at the level of current constraints.

This can be done by the construction seen in Figure 3.2: Only the dashed edges are allowed to be added. Adding an edge at the connection e_{x_i} or $e_{\neg x_i}$, the variable is assumed to be set to true or false, respectively. A current is running from s_v to t_v. Depending on the number of edges inserted (0, 1 or 2), the current will favour e_1 over e_2 to reach t_v. Now the maximum currents on these edges e_1 and e_2 must be set in such a way that the constraint on e_2 is violated iff no edge has been inserted and the constraint on e_1 is violated iff two (ore even more) edges were inserted.

Observation 3. *Figure 3.2 illustrates how it is possible to map the exclusive choice of setting x_i to either true or false by using current constraints on a pair of edges.*

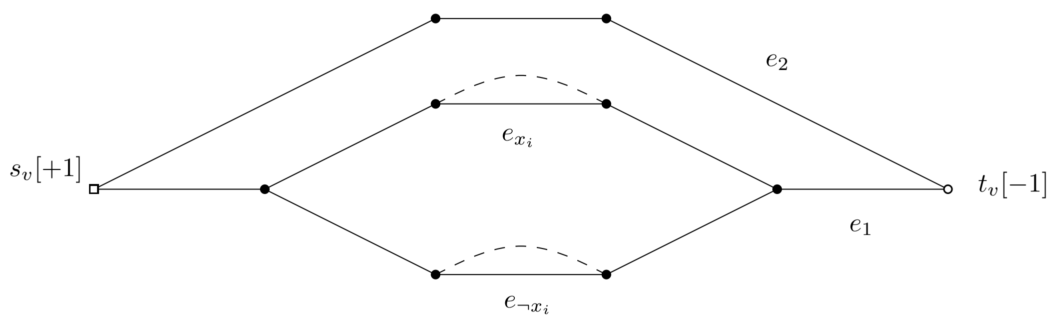

Figure 3.2.: Modelling an exclusive choice of literals

Moreover, this construction is able to enforce at least one of the two options, so a third option of not enhancing either connection (which may correspond to a “don’t care” state) is excluded.

However, as will be shown later, it is more convenient to combine all subgraphs corresponding to boolean variables with each other, as shown in Figure 3.3. Here, the constraints

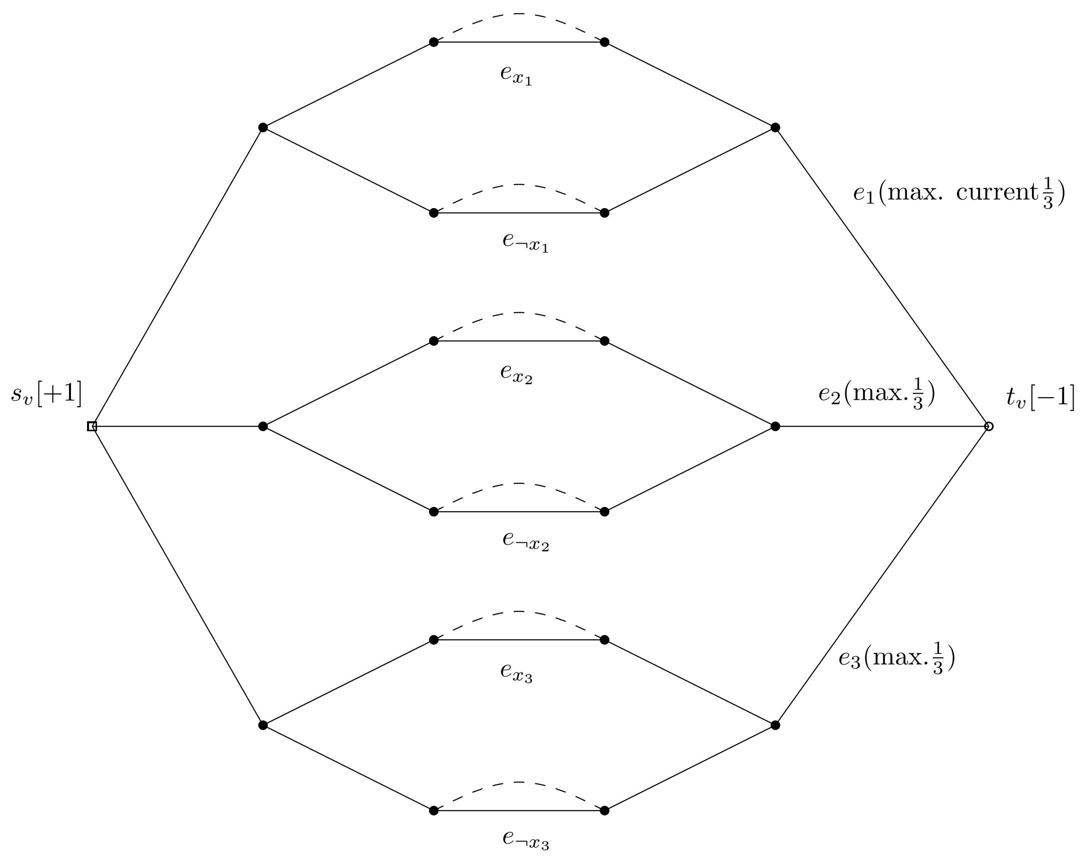

Figure 3.3.: Combined variable-blocks for $k = 3$ variables

on the edges e_i are simply to carry at most $\frac{1}{k}$ of current, where k is the total number of variables. Each subgraph consisting of six nodes associated with a boolean variable will be called *variable-block* or *variable-cell.* That way, the constraints on $e_1, \dots, e_k$ work like a *symmetry sensor*: the constraints are fulfilled iff each variable-cell has the same amount of added edges (all $0, 1$ or 2), where there can only be k edges added in total. In the case of Figure 3.3 where the variable-blocks are not connected to clause-blocks etc., the analysis of the network can be achieved by intuitively combining edges in series and in parallel.

The ideas from Figures 3.1 , 3.2 and 3.3 can be merged into a single electrical network featuring two currents "crossing" each other: A current running from "top to bottom" that supplies the clause-blocks and another current running from "left to right" that supplies the variable-cells. Both currents will – in parallel – benefit from adding new edges in

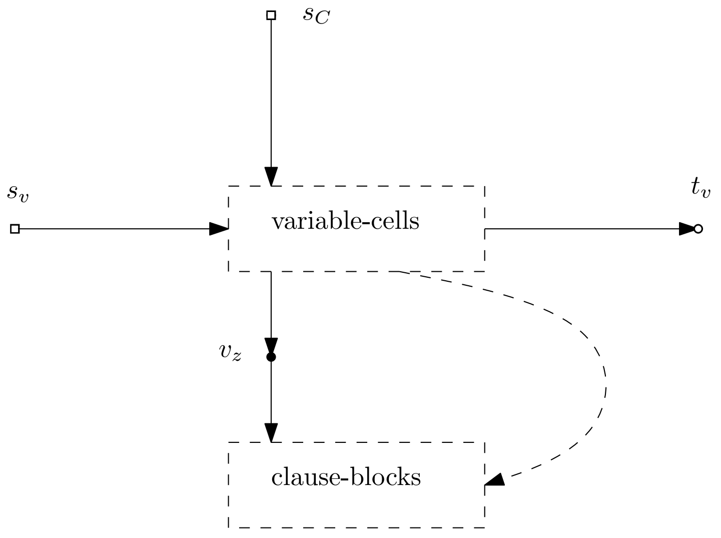

Figure 3.4.: Crossing circuits, edge directions follow currents direction

the manner described above. This idea and the exact mapping from 3-SAT to TNEP are explained in the following section together with Figure 3.4.

Given 3-SAT formula F with k variables in total, the Laplacian matrix L of the corresponding TNEP scenario will take all $8\binom{k}{3}$ clauses into account that can be built out of k variables. However, only the clause-blocks corresponding to F will be given the constraints on the maximum currents (on edge e_2 in Figure 3.1). By this exhaustive construction the network becomes highly symmetric and predictable and the analysis is only dependent on k.

3.2. Reduction procedure

The complete transformation of a boolean formula to a TNEP scenario is achieved by the procedure following Table 3.1 which contains an overview of the node names used.

Name	Description	supply
s_C	(super)-source for all clause blocks	$[+8\binom{k}{3}]$
s_v	source for all variable blocks	$[+1]$
t_v	sink for all variable blocks	$[-1]$
$v_{x_i}^{(1)}$	first node of variable-cell i	$[0]$
...	...	$[0]$
$v_{x_i}^{(6)}$	last node of variable-cell i	$[0]$
v_z	interconnect-node from variable-cells to clause-blocks	$[0]$
v_{C_j}	first node of clause-block j	$[0]$
t_{C_j}	second node (sink) of clause-block j	$[-1]$

Table 3.1.: Node names of 3-SAT to TNEP scenario

Let a formula F be given in 3-SAT conjunctive normal form (CNF). One may assume without loss of generality that each clause contains exactly three variables. Otherwise simplify clauses like $x_j \vee x_i \vee x_i$ to $x_j \vee x_i$ if necessary and inflate them artificially until they contain exactly three variables. For example, the clause $x_j \vee x_i$ could be inflated to $(x_j \vee x_i \vee x_{k_1})$ together with the clauses $(\neg x_{k_1} \vee x_{k_2} \vee x_{k_3})$, $(\neg x_{k_1} \vee x_{k_2} \vee \neg x_{k_3})$, $(\neg x_{k_1} \vee \neg x_{k_2} \vee x_{k_3})$ and $(\neg x_{k_1} \vee \neg x_{k_2} \vee \neg x_{k_3})$ where $x_{k_1}, x_{k_2}, x_{k_3}$ denote new, unused variables. The latter clauses are just to make sure x_{k_1} is bound to be set as “false”, thus to original clause is only satisfied if x_j or x_i is set as “true”.

Trivial clauses like $x_j \vee x_i \vee \neg x_i$ which are equivalent to 1 can be ignored.

After making sure each clause contains exactly three variables, the four meta-nodes are created:

s_C	(the super-source)
$s_v \quad t_v$	(variable-source and -sink)
v_z	(the interconnect-node)

See Table 3.1 for a quick explanation.

Following the previous section, the general purpose of these nodes can be seen in Figure 3.4: A (large) current is introduced on the “top” from s_C and carried “down” via the variable-cells towards the clause-blocks by passing the interconnect-node v_z or the alternative dashed route. Another (smaller) current enters from the “left” (s_v) and is carried towards the “right” to t_v mostly via the variable-cells. Following these directions, the two currents are said to *cross* each other here. This idea is justified by the principle of superposition explained in Observation 1.

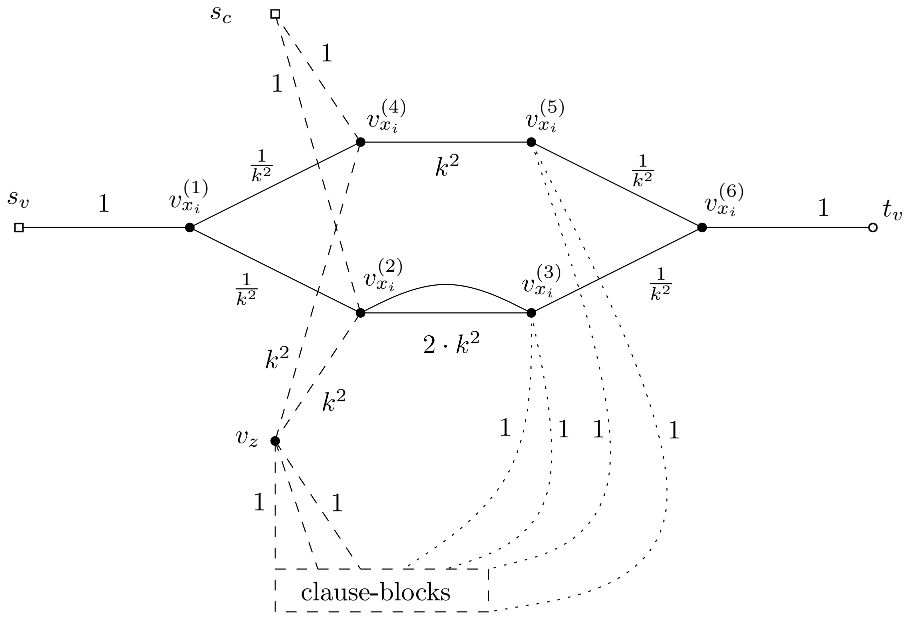

Figure 3.5.: Specification: conductances for variable-cell i

After having created the four meta- or helper-nodes s_C, s_v, t_v, v_z, for each boolean variable x_i in F one variable-cell is created. The specific conductances can be seen in Figure 3.5. Node $v_{x_i}^{(1)}$ of the variable-cell i is connected to the variable-source s_v and $v_{x_i}^{(6)}$ is connected to the variable-sink t_v. After that, each cell is connected to the both s_C and v_z via the nodes $v_{x_i}^{(2)}$ and $v_{x_i}^{(4)}$. The low-conductance parts of the variable-cell are supposed to restrict the effect of clause-currents running from the enhanced-part to the non-enhanced part of the variable-cell. It works like a barrier which the clause-currents will avoid. On the opposite side, the variable-current has “no other choice” than to use these low-conductance connections to run to the variable-sink. The illustration may be helpful when applying the principle of superposition: consider the total current to be composed by the variable-source and -sink on the one hand and the clause-source and clause-sinks on the other hand. Then the clause-currents will avoid taking a detour via the low-conductance connections as it requires a lot of (dissipated) energy to pass these edges.

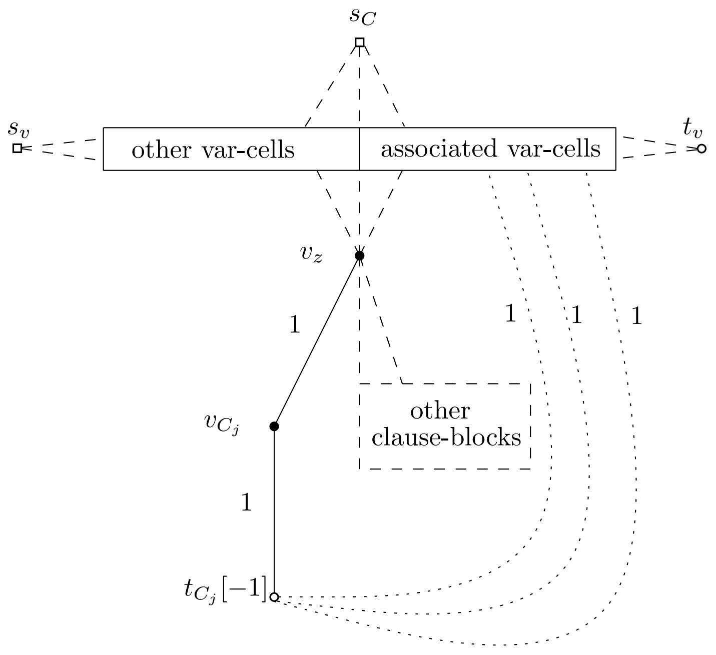

Figure 3.6.: Specification: conductances for clause-block j

Then, the clause-blocks are taken care of. For each 3-SAT formula containing $k \geq 3$ variables, there are at most

$$|C_{\text{tot}}(k)| = 2^3 \cdot \binom{k}{3} = \frac{4}{3}k(k-1)(k-2) \tag{3.1}$$

total clauses that can be build from k variables containing exactly 3 variables each. Each of these clauses will be considered in the resulting network. Notice that since only 3-SAT formulas are considered here, the total number of clauses built from k variables is still polynomial in k. For each clause $C_j = \{(\neg)x_{j_1}, (\neg)x_{j_2}, (\neg)x_{j_3}\} \in C_{\text{tot}}(k)$ a clause-block according to Figure 3.6 is created. The clause's sink t_{C_j} is connected to the variable-cells according to the literals in C_j: if $\neg x_{j\theta} \in C_j$ then t_{C_j} is connected to $v^{(3)}_{x_{j\theta}}$ otherwise if $x_{j\theta} \in C_j$ it is connected to $v^{(5)}_{x_{j\theta}}$ (for $\theta = 1, 2, 3$).

Algorithm 1 describes the transformation in Pseudo-Code.

An implementation of Algorithm 1 in Scilab/Matlab can be found in the Appendix.

In order to receive a proper TNEP scenario, the data (L, b) received from Algorithm 1 needs to be enriched (informally) in the following way, summarized by Definition 12. Only the connections e_{x_i} and $e_{\neg x_i}$ may be enhanced. This is done by an edge of conductance k^2 and there is only one such edge allowed for each connection $e_{(\neg)x_i}$. This way, each assignment α of the boolean variables can be translated into a unique set of edges to add at the variable-cells (according to which literal is activated). Additionally, all edges connected to t_v get the current constraint $\bar{f}_v = \frac{1}{k}$. This is supposed make sure that invalid assigning of both (or none of the) literals of a variable will cause at least one of the current constraints to fail, so only "real" variable assignments are possible.

At last, the current constraint $\bar{f}_C$ for the connection $e_C = \{v_{C_j}, t_{C_j}\}$ of the clause-blocks must be determined. Obeying these represents the satisfaction of a clause. It will be shown

Algorithm 1: 3-SAT-to-TNEP-scenario(A)

Data: 3-SAT formula matrix A

Result: Laplacian matrix L and supply vector b of corresponding TNEP

```
$k \leftarrow \#$ columns($A$) ;                                  /* number of variables */
$m \leftarrow \#$ rows($A$) ;                          /* number of important clauses */
$n \leftarrow 1 + 2 + 6 \cdot k + 1 + 2 \cdot |C_{\text{tot}}(k)|$ ;                          /* number of nodes */
$V \leftarrow (v_1, \dots, v_n)$ sorted dummy nodes;
$E \leftarrow \emptyset$ initial edges;
$G \leftarrow (V, E)$ initial graph;
$v_1 \leftarrow s_c$ ;                                                  /* super-source */
$v_2 \leftarrow s_v$ ;                                       /* variable-block-source */
$v_2 \leftarrow t_v$ ;                                         /* variable-block-sink */
for $i = 1, \dots, k$ do
    $\theta \leftarrow 4 + 6(i - 1)$;
    $(v_\theta, \dots, v_{\theta+5}) \leftarrow$ nodes of variable-block $i$;
    connect $(v_\theta, \dots, v_{\theta+5})$ according to Figure 3.5 ;      /* nodes $v_{x_i}^{(1)}$ to $v_{x_i}^{(6)}$ */
    Connect variable-block $i$ to $s_c, s_v, t_v$;
end
$v_{4+6k} \leftarrow t_v$ ;                              /* create interconnect-node $v_z$ */
Connect $v_z$ to variable-blocks ;
$j \leftarrow 0$;
foreach $C \in C_{\text{tot}}(k)$ do
    $j \leftarrow j + 1$;
    $\theta \leftarrow 5 + 6k + 2(j - 1)$;
    $(v_\theta, v_{\theta+1}) \leftarrow$ nodes of variable-block $j$;
    connect $v_{C_j}$ to interconnect-node $v_z$;
    foreach $x_i \in C$ do
        connect $t_{C_j}$ to $v_{x_i}^{(5)}$ ;            /* unnegated branch of variable-cell $i$ */
    end
    foreach $\neg x_i \in C$ do
        connect $t_{C_j}$ to $v_{x_i}^{(3)}$ ;              /* negated branch of variable-cell $i$ */
    end
end
$L \leftarrow$ Laplacian matrix of created $G$;
Switch clause-blocks corresponding to $A$ to the first $m$ blocks in $L$ ;
Set $b$ according to Table 3.1;
return $L, b$
```

later, that determining a proper $\bar{f}_C$ can be done by plugging in an arbitrary assignment of variables, computing the resulting current x and examining the values as follows. Define

$$\begin{aligned} x^{(0)}_{v_C,t_C} &:= x_{v_{C_j},t_{C_j}} \quad \text{for any } j \text{ with 0 enhanced literals connected} \\ x^{(1)}_{v_C,t_C} &:= x_{v_{C_j},t_{C_j}} \quad \text{for any } j \text{ with 1 enhanced literal connected} \end{aligned}$$

and set the constraint

$$\bar{f}_C := \frac{x^{(1)}_{v_C,t_C} + x^{(0)}_{v_C,t_C}}{2} \tag{3.2}$$

This constraint makes use of a monotonic behaviour of the clause-currents[2]: the more connected literals are activated by the respective assignment, the lower the clause-current will be. This and the fact that $\bar{f}_C$ is well defined will be formally shown later by Proposition 4.The constraint $\bar{f}_C$ has to be set on the clauses associated with the original formula F *only.* All other clause-blocks do not receive a constraint as it is not important if they are satisfied.

Definition 12 (3-SAT to TNEP scenario)**.** *The TNEP scenario associated with the given* 3*-SAT scenario A with k variables is defined by* (L, b) *from Algorithm 1 and the following:*

1. *at most one edge can be added to each connection* $(v^{(2)}_{x_i}, v^{(3)}_{x_i})$ *(negated literal) and* $(v^{(4)}_{x_i}, v^{(5)}_{x_i})$ *(unnegated literal) for* $i = 1, \ldots, k$*.*
2. *No other edges are allowed to be added.*
3. *each added edge has conductance* $c = k^2$ *and cost* $w = 1$*.*
4. *all edges connected to* t_v *get the current constraint* $\bar{f}_v = \frac{1}{k}$*.*
5. *all clause-blocks corresponding to clauses from the given* 3*-SAT scenario A receive the current constraint* $\bar{f}_C$ *from Equation 3.2* [3]*.*

The associated decision question is:
Is there a set of edges with cost $\leq k$ that fulfils all current constraints? *(4 and 5 in enumeration above)*

This section is followed by some examples after which the correctness of the reduction will be proved.

3.3. Examples

In order to adapt a more intuitive approach to the transformation, this will provide an example of the complete transformation for a given 3-SAT formula F to a TNEP scenario.

Example 11 (Formula to be transformed)**.** *Consider the* 3*-SAT formula*

$$F = (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

with $k = 3$ *variables. The input for Algorithm 1 is the matrix*

$$A = \begin{pmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}$$

[2]that is: the current from v_{C_j} to t_{C_j}
[3]those are the first m clause-blocks

For $k = 3$ variables it holds that there are $|C_{\text{tot}}(3)| = 2^3 \cdot \binom{3}{3} = 8$ total clauses that can be built. Thus, Algorithm 1 delivers an electrical network that consists of

- 1 super-source s_c
- 1 variable-source s_ν and 1 variable-sink t_ν
- 3 variable-blocks with 6 nodes each
- 1 interconnection-node v_z
- 8 clause-blocks with 2 nodes each

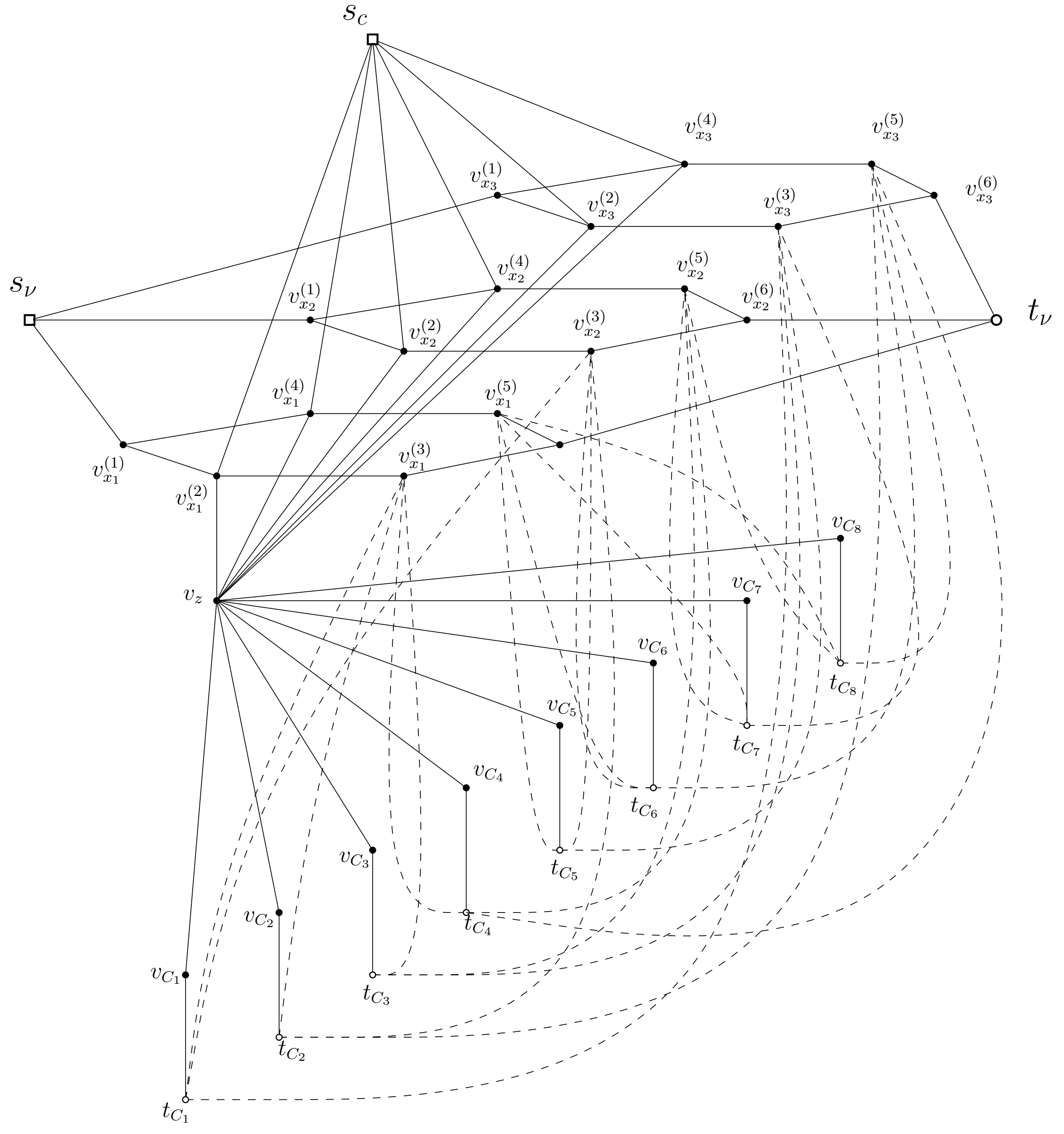

Figure 3.7.: transformation for $k = 3$ variables

Making up for $3 + 3 \cdot 6 + 1 + 2 \cdot 8 = 38$ nodes in total. The complete resulting network can be seen in Figure 3.7. The plane of variable-cells from Figure 3.3 can be seen in the upper part of the figure and the clause-blocks are in the lower part. The dashed lines represent the connections from the clause-blocks to the respective literals appearing in the clauses. Neither the rearrangement of the clauses nor the conductances are displayed in the figure.

The clauses from formula F correspond to C_2, C_3 and C_6 within the figure. Thus, those are the only clause-blocks which are given current-constraints. To find out the the specific constants according to Equation 3.2, one can arbitrarily plug in the assignment $\alpha = (-1,-1,-1)$ assigning each variable the value “false”. After enhancing the connections $(v_{e^{(2)}_{x_\theta}}, v_{e^{(3)}_{x_\theta}})$ $(\theta = 1,2,3)$ which correspond to α, one gets the following (rounded) clause-currents for C_1 to C_8

$$\begin{matrix} 0.1933201 & 0.2033569 & 0.2033569 & 0.2133938 \\ 0.2033569 & 0.2133938 & 0.2133938 & 0.2234306 \end{matrix}$$

This list contains four different values

$$0.2234306 \quad 0.2133938 \quad 0.2033569 \quad 0.1933201$$

that correspond to clauses with 0, 1, 2 and 3 enhanced literals connected. Thus, set

$$\bar{f}_C := \frac{(0.2234306 + 0.2133938)}{2} = 0.2184122$$

Moreover, the three edges connected to t_v are given the current constraint of $\frac{1}{3}$.[4]

Remark 4. *Within the implementation in the appendix the previous steps can be performed by the instructions*

$$\begin{aligned} [L, b] &= \mathit{canonical3SATtoTNEP}(3); \\ \mathit{alpha} &= [-1-1-1]; \\ [L1, p, \mathit{varCur}, \mathit{clauseCur}] &= \mathit{assignVariablesNEvaluate}(L, b, \mathit{alpha}); \end{aligned}$$

where clauseCur will contain the clause-current for each clause-block after enhancing the edges corresponding to the assignment $\alpha = (-1,-1,-1)$ (last argument in second function call).

As can be checked from varCur the variable-currents are all at 0.3333333. Notice that the conductance values in the implementation are slightly different than those mentioned in the previous section 3.2 which is irrelevant for the general behaviour of the graph since the conductances' asymptotic classes like $\Theta(k^{-2})$ stay the same.

As can be seen from the previous lists, the arbitrarily chosen assignment α already fulfils the current-constraints given to the clause-blocks of C_2, C_3, C_6 in Figure 3.7 which reflects the fact that α satisfies all clauses in F. In fact, in this very small example for $k = 3$, any assignment will violate exactly one clause, where in general the number of violated clauses is $|C_{viol}(k,\alpha)| = \binom{k}{3}$ for any α (so $\frac{1}{8}$ of the all clauses are violated in general).

3.4. Analysis

Theorem 4. *Definition 12 describes a polynomial reduction from 3-SAT to TNEP.*

Corollary 5. *TNEP is NP-complete.*

Proof of Corollary 5. On the one hand, according to Lemma 3 TNEP is part of the complexity class NP. On the other hand, from Theorem 4 it follows that TNEP is NP-hard. From both facts follows that TNEP is NP-complete. □

[4]to make the procedure of solving for these currents numerically more stable one would have to choose variable-currents that are slightly above the $\frac{1}{k}$ limit. However, numerical stability is not the issue here.

To prove Theorem 4 consider the following. For a TNEP scenario S given by Definition 12, define any TNEP instance $\vec{n}$ for S to be *SAT-consistent* if it represents a valid assignment of the respective boolean variables $x_1, \dots, x_k$. That is: $\vec{n}$ provides exactly one additional edge for each variable-block and can thus be uniquely translated to an assignment of variables $\alpha : \{x_1, \dots, x_k\} \to \{-1, 1\}$. Call $\vec{n}$ *trivial* if it represents an instance without adding edges.

There will be five separate Propositions presented here in order to prove Theorem 4 step by step. This will done by showing the following:

- the reduction procedure is polynomial in k (Proposition 1),
- only SAT-consistent instances (and the trivial instance) can obey all variable-constraints $\bar{f}_v$ at once (Proposition 3),
- in each scenario there are exactly four distinct clause-currents for all SAT-consistent instances. These currents are dependent of the number of connected enhanced literals in a monotonic way (Proposition 4)
- the trivial instance does not obey any clause-current constraint $\bar{f}_C$ (Proposition 5).

For these steps, Proposition 2 will make use of the symmetry in order to reduce the general case to a network of at most 38 nodes for any k. This will allow the use of a computer aided solver (Matlab) to compute the general voltage potentials symbolically.

Proposition 1. *The procedure described in Algorithm 1 can be executed in time polynomial in k.*

Proof of Proposition 1. Given a 3-SAT formula $F = \{C_1, \dots, C_m\}$ with k variables we may assume that $k \leq 3m$ and $m \leq 8\binom{k}{3}$ by considering the minimum and maximum number of clauses possible. Thus, the input size $|F|$ is in $\Omega(k)$ and $m \in O(\binom{k}{3}) = O(k^3)$, so $|F|$ is at most in $O(k^4)$. It is not important which specific format F is given in. In each case, the matrix $A = A(F) \in \{-1, 0, 1\}^{m \times k}$ representing F can be extracted in polynomial time with respect to $|F|$ and k.

Given the matrix A, line 3 of Algorithm 1 defines the the number of nodes n of the network to be created, where $n \in O(k^3)$, so the network size is polynomial in $|F|$ and so is the resulting Laplacian $L \in \mathbb{R}^{n \times n}$.

Each instruction in line 7 to 29 takes constant amount of time. Since the loops in these lines need to file through A when creating the variable-blocks and the clause-blocks, the total amount of time taken here sums up to at most $O(k^4)$.

Sorting the clauses in line 32 according to their original appearance in A takes at most $O(k^4 \log k)$ steps although the implementation does not explicitly need to perform a sorting or rearrangement of the clause-blocks.

At last, setting the supply-vector $b \in \mathbb{R}^n$ requires $O(n)$ steps.

Altogether extracting A and running Algorithm 1 takes time and space polynomial in $|F|$. Here $O(k^4 \log k)$ is a generous upper bound on the time needed and $O(n^2) = O(k^6)$ limits the required space for L in a naive (adjacency) matrix representation, ignoring time needed to initialise and reserve space.

To complete the transformation to the desired TNEP scenario S, we need to determine the bounding constraints for S. Although Algorithm 1 delivers a Laplacian matrix L for S only, the vectors as given in Definition 8 can be extracted in a straightforward way: For each connection $\{i, j\} \in V^2$, set the initial number of edges $n_{ij}^{(0)} := 1$, the cost $w_{ij} := 1$ and the conductance per edge $c_{ij}^{(0)} := |L_{ij}|$. Furthermore set

$$\bar{n}_{ij} := \begin{cases} 2, & \text{if } \{i,j\} \in \{\{v_{x_\theta}^{(2)}, v_{x_\theta}^{(3)}\}, \{v_{x_\theta}^{(4)}, v_{x_\theta}^{(5)}\} \mid \theta = 1, \dots, k\} \\ 1, & \text{otherwise} \end{cases}$$

This needs at most $O(n^2)$ time.

To compute the current constraints for the clause-blocks as described in the previous sections one needs to plug in an arbitrary assignment like $\alpha = (-1, \dots, -1)$. Updating L and solving the linear equation $L \cdot p = b$ is achievable in $O(n^3)$ steps. This provides $\bar{f}_C$ by searching for the biggest and second biggest current running through the clause-blocks which takes $O(m)$ steps (proving this method to be well-defined will be done later in Proposition 4). The other constraint $\bar{f}_v = \frac{1}{k}$ for the variable-blocks can be plugged in right away. For the other edges the maximum current constraint can be omitted or set to a trivial value like $m + 1$ which takes at most $O(n^2)$ steps.

This concludes Proposition 1 with a generous polynomial upper bound of $O(n^3)$ on the time and $O(n^2)$ on the space. □

To continue the analysis for the variable- and clause-currents the following Proposition on symmetry needs to be proven first. Originally, given a formula F with k variables, the respective TNEP scenario given from Definition 12 features a network made out of $4 + 6 \cdot k + 2 \cdot 8\binom{k}{3} \in O(k^3)$ nodes. Hence, the corresponding Laplacian matrix L has a size depending of k (of order $O(k^6)$). As already mentioned before, the network created by Definition 12 is highly symmetric in the sense of Definition 7 and can consequently be drastically scaled down using Theorem 3.

Proposition 2 (Symmetry in 3-SAT to TNEP reduction). *For each k in the network obtained by Algorithm 1 all variable-cells with the same number of edges added are symmetric to each other, making up for at most* 3 *types of variable-cells. Similarly all clause-blocks are symmetric by the number of connections to enhanced literals, so there are at most* 20 *types of clause-blocks. By that the network can be reduced to a network of size at most* 38.

Thus, it will be possible to perform the further analysis for a network of fixed size for any k which allows the use of computer aided solving of symbolic linear equations for the following propositions.

Proof of Proposition 2. Denote by

$$n_\theta \quad := \quad \text{number of variable-cells with } \theta \text{ edges added } (\theta = 0, 1, 2)$$

According to the number $\theta \in \{0, 1, 2\}$ of added edges in a variable-cell i this cell is called to be of (enhancement) type 0,1 or 2 which reflect all possible cases of enhancing a particular variable-cell due to the defined boundaries. As this represents a partitioning of the variable-cells, it holds

$$k = n_0 + n_1 + n_2$$

Now it is shown why two variable-cells x_1, x_2 of same enhancement type θ are symmetric. This is achieved by a permutation π_θ which is set by switching the variable-cells and all clauses in which x_1, x_2 are appearing. This is equivalent to switching variables by renaming them within the original formula F.

Together with the notation

$$\text{variable-block}(x_i) := \begin{cases} (v_{x_i}^{(0)}, v_{x_i}^{(1)}, \dots, v_{x_i}^{(6)}) & \text{if } (v_{x_i}^{(2)}, v_{x_i}^{(3)}) \text{ is enhanced} \\ (v_{x_i}^{(0)}, v_{x_i}^{(1)}, v_{x_i}^{(4)}, v_{x_i}^{(5)}, v_{x_i}^{(2)}, v_{x_i}^{(3)}, v_{x_i}^{(6)}) & \text{otherwise} \end{cases}$$

define π_θ to be

$$\pi_\theta(\text{variable-block}(x_1)) := \text{variable-block}(x_2) \tag{3.3}$$
$$\pi_\theta(\text{variable-block}(x_2)) := \text{variable-block}(x_1) \tag{3.4}$$
$$\pi_\theta(\text{variable-block}(x_i)) := \text{variable-block}(x_i) \text{ if } x_i \neq x_2, x_1 \tag{3.5}$$

Furthermore, denote by $\pi_\theta(x_i)$ and $\pi_\theta(\neg x_i)$ the variables/literals on which variable-cell i is mapped to by π_θ according the previous equations. Then for all clauses C_j map the clause-block j to the clause-block that is defined by mapping all literals from C_j individually:

$$\begin{aligned} \text{for all } C_j &= \{(\neg)x_{j1}, (\neg)x_{j2}, (\neg)x_{j3}\} \\ \text{and } \widehat{C}_j &= \{\pi_\theta((\neg)x_{j1}), \pi_\theta((\neg)x_{j2}), \pi_\theta((\neg)x_{j3})\} \\ \pi_\theta((v_{C_j}, t_{C_j})) &:= (v_{\widehat{C}_j}, t_{\widehat{C}_j}) \\ \pi_\theta &\equiv id \quad \text{for all other nodes} \end{aligned}$$

and letting π_θ map the connections accordingly. This defines a permutation of nodes according to Definition 7 which - due to the restriction to conserve the conductances when mapping connections (Equation 2.13 and 2.12) - only works when x_1 and x_2 have exactly the same amount of enhanced edges. Equation 3.3 and 3.4 make sure that the enhanced parts are always mapped on each other. Notice that this may also include switching the role of an negated literal towards an non-negated one and vice versa.

Hence, all variable-cells of the same enhancement type are symmetric and thus have the same voltage potentials. From this follows that the respective nodes of the same potential can be merged together. Consequently, *for any TNEP instance the corresponding network can be mapped to have at most three variable-cells* which are scaled by the respective numbers n_0, n_1, n_2.

By a similar permutation it can be shown that there is only a fixed number of clause-types depending on how many connections to each of the three types of variable-cells they have. There are four types of connections from the clause-blocks towards the three variable-cells:

$$\begin{aligned} t_0 &:= \text{connections to variable-cell of type 0 (both branches)} \\ t_1^+ &:= \text{connections to non-enhanced branch of variable-cell of type 1} \\ t_1^- &:= \text{connections to enhanced branch of variable-cell of type 1} \\ t_2 &:= \text{connections to variable-cell of type 2 (both branches)} \end{aligned}$$

Two clause-blocks v_{C_1}, t_{C_1} and v_{C_2}, t_{C_2} that share the same amount of connections to each of the four connection types of the variable-blocks are symmetric by a permutation π. Here $C_\theta = ((\neg)x_{\theta,1}, (\neg)x_{\theta,2}, (\neg)x_{\theta,3})$ for $\theta = 1, 2$ are assumed to be sorted by two keys: ascending by the amount of enhanced variable-branches in $x_{\theta,i}$ (according to t_0, t_1^+, t_1^- and t_2) and secondly by some arbitrary but fixed ordering of the variables (for example by the ordering in which they appear in L). Then let π switch both the roles of C_1 and C_2 and the roles of its respective literal-types while leaving all other variable-blocks fixed.

$$\begin{aligned}
\text{for } \eta = 1,2,3 &\quad : \\
\pi(\text{variable-block}(x_{1,\eta})) &:= \text{variable-block}(x_{2,\eta}) \\
\pi(\text{variable-block}(x_{2,\eta})) &:= \text{variable-block}(x_{1,\eta}) \\
\pi(\text{variable-block}(x_i)) &:= \text{variable-block}(x_i) \text{ if } x_i \neq x_{2,\eta}, x_{1,\eta} \\
\text{for all } C_j &= \{(\neg)x_{j_1}, (\neg)x_{j_2}, (\neg)x_{j_3}\} \\
\text{and } \widehat{C}_j &= \{\pi((\neg)x_{j_1}), \pi((\neg)x_{j_2}), \pi((\neg)x_{j_3})\} \\
\pi((v_{C_j}, t_{C_j})) &:= (v_{\widehat{C}_j}, t_{\widehat{C}_j}) \\
\pi &\equiv id \quad \text{for all other nodes}
\end{aligned}$$

and letting π map the connections accordingly. Hence, all clause-blocks that represent clauses containing the variables $x_{1,\eta}$ or $x_{2,\eta}$ (for $\eta = 1,2,3$) are swapped. All other clause-blocks remain in their original position. Notice that this explicitly means C_1 and C_2 are mapped on each other. Again, the map only works because the variable-cells are mapped towards blocks of equal types and conductances. Using basic combinatorics yields to $\binom{6}{3} = 20$ different clause-types since a total of three connections are chosen from four connection-types. With the notation above, for each clause-type $t = (t_0, t_1^+, t_1^-, t_2)$[5] the total number of clauses $cl(t)$ belonging to this type is

$$cl(t) = \binom{n_0}{t_0} 2^{t_0} \cdot \binom{n_1}{t_1^+}\binom{n_1 - t_1^+}{t_1^-} \cdot \binom{n_2}{t_2} 2^{t_2}$$

provided that the binomial expressions exist which is always the case when $n_i \geq 3$. Otherwise $cl(t) = 0$ for some t leads to a smaller amount of clause-types. Now, determining the conductances of the combined edges can also be achieved by basic combinatorics. For each clause-type t the connections from t_{C_j} to the variable-cell types have the following conductances

$$\begin{aligned}
&\frac{t_i}{2} \cdot cl(t) &&\text{to both } v_{x_i}^{(3)} \text{ and } v_{x_i}^{(5)} \text{ for } i = 0,2 \\
&t_1^- \cdot cl(t) &&\text{to } v_{x_i}^{(3)} \\
&t_1^+ \cdot cl(t) &&\text{to } v_{x_i}^{(5)}
\end{aligned}$$

Hence, by merging the symmetric parts using Theorem 3 so far the TNEP scenario can be reduced to a network of size at most $3 \cdot 6 + 4 + 20 \cdot 2 = 62$. Additionally, the clause-blocks can be reduced to t_{C_j} only by merging $\{v_z, v_{C_j}\}$ with $\{v_{C_j}, t_{C_j}\}$ due to laws of series conductances. Moreover, the variable-blocks of type 0 and 2 can also be reduced by merging $v_{x_i}^{(2)}$ with $v_{x_i}^{(4)}$ and $v_{x_i}^{(3)}$ with $v_{x_i}^{(5)}$ as these nodes are symmetric within the same variable-block (by renaming and swapping the literals). That even leads to a network of size at most $62 - 20 - 4 = 38$. □

Proposition 3. *A TNEP instance $\vec{n}$ is SAT-consistent (or trivial) iff all the current constraints $\bar{f}_v$ at the variable-sink t_v are fulfilled.*

All of the current-constraints at the variable-blocks are fulfilled when the blocks are symmetric which is the case for all SAT-consistent (and the trivial) TNEP instance since the variable-cells are symmetric as shown in Proposition 2. However, when they are not symmetric due to different numbers of edges added, a physical intuition indicates the current will favour those variable-blocks which are enhanced by more edges in order to reach its destination t_v. Hence in this case there will be a slightly higher current running from $v_{x_i}^{(6)}$ to t_v for the enhanced blocks compared to the less enhanced blocks.

[5] $t_0 + t_1^+ + t_1^- + t_2 = 3$

Proof of Proposition 3. The corresponding system of linear equations by Equation 2.5 has a fixed size (independent of k) but contains symbolic variables (n_0, n_1, n_2). The claim can be proven by solving the linear equations using a computer-aided symbolic system solver.

The respective variable-currents were compared and simplified using Matlab. It was observed that all three variable-current types share the same denominator which allowed simplification by omitting this denominator (after checking for zero points first). Afterwards it was noted that the *difference* between the current running through the variable-blocks of type t_0, t_1, t_2 is independent of n_0, n_1, n_2 and only depends on k. Denote by $\Delta_{i,j}$ the difference of currents between type i and j. Omitting the denominator, Matlab yielded the following expressions:

$$\begin{aligned}
\Delta_{1,0} &= (7 \cdot k \cdot (k^4 + 3 \cdot k^2 + 3) \cdot (32 \cdot k^5 - 120 \cdot k^4 + 152 \cdot k^3 - 69 \cdot k^2 + 14 \cdot k + 12) \\
&\quad \cdot (27 \cdot k^8 - 60 \cdot k^7 + 57 \cdot k^6 - 30 \cdot k^5 + 51 \cdot k^4 - 30 \cdot k^3 + 27 \cdot k^2 + 7) \\
&\quad \cdot (44 \cdot k^8 - 66 \cdot k^7 + 212 \cdot k^6 - 286 \cdot k^5 + 369 \cdot k^4 \\
&\qquad - 374 \cdot k^3 + 261 \cdot k^2 - 132 \cdot k + 45)) \cdot \frac{1}{3} \\
\Delta_{2,1} &= (7 \cdot k \cdot (k^4 + 3 \cdot k^2 + 3) \cdot (32 \cdot k^5 - 120 \cdot k^4 + 152 \cdot k^3 - 69 \cdot k^2 + 14 \cdot k + 12) \\
&\quad \cdot (44 \cdot k^8 - 90 \cdot k^7 + 84 \cdot k^6 - 30 \cdot k^5 + 65 \cdot k^4 - 30 \cdot k^3 + 27 \cdot k^2 + 7) \\
&\quad \cdot (27 \cdot k^8 - 44 \cdot k^7 + 133 \cdot k^6 - 198 \cdot k^5 + 243 \cdot k^4 \\
&\qquad - 286 \cdot k^3 + 189 \cdot k^2 - 132 \cdot k + 45)) \cdot \frac{1}{3}
\end{aligned}$$

Using Horner's scheme, it can be shown that these polynomials are greater zero for all $k \geq 3$. Thus, the variable-currents for the three different types have to be strictly different. Together with the fact that the sum of these currents is still exactly 1 (since the variable-sink has demand 1) it follows that at least one variable-current is greater than $\frac{1}{k}$, so at least one current constraint at the variable-blocks is violated.

The number of clause-types is assumed to be the general number of 20 for the computations presented. This implicitly assumes the numbers of variable-blocks n_2, n_1, n_0 are greater or equal 3. The other cases where at least one of these numbers is equal $0, 1$ or 2 (provided $n_1 < k$ to not allow SAT-consistency) are just specializations of this general case which are easier to solve. For the sake of space, they are omitted here.

□

Remark 5. *Despite the fact that the variable-currents are theoretically different for each k, theses differences still tend to 0 for $k \to \infty$. Thus, for testing big instances of the transformed TNEP scenario in practice, one would have to increase the variable-current depending on k and increase the variable-current-bound accordingly in order to highlight the differences for floating point applications.*

Proposition 4. *For each SAT-consistent instance $\vec{n}$ there are exactly four types of clause-currents x_i where i is the number of connected enhanced literals. They are decreasing monotonically: $0 < x_3 < x_2 < x_1 < x_0$.*

Proof of Proposition 4. According to Proposition 3 it is sufficient to only consider SAT-consistent TNEP instances in the following. Similar to the symmetry used in Proposition 2, this yields an even smaller amount of clause-block types of which there are four now.

Two clause-blocks are symmetric if they have the same number of enhanced literal branches connected to them. Analogue to Proposition 2 this number partitions the clause-blocks into four clause-block types (from 0 enhanced literals to 3). Again, due to Theorem 3 all clause-blocks of the same type can be merged together. The same can be said for *all* variable-cells. As should be clear by Proposition 2, in the case of a SAT-consistent TNEP instance there is only one type of variable-cell left, namely the one with a single edge enhanced. By merging the nodes accordingly there is a network of at most $4+6+4\cdot 2 = 18$ nodes left. Denote by x_0 to x_3 the respective currents that run through the four different clause-block-types (directed from v_C to t_C). The only thing that needs to be shown in this context is that these currents are in fact different and greater than zero. The resulting network is small enough to let the currents be computed by a state-of-the-art symbolic equation solver. Again, Matlab was used (See the Appendix for code used).

The respective clause-currents $x_0, \ldots, x_3$ are rational functions of k. It emerged that the differences $\delta_i := x_i - x_{i+1}$ for $i = 0, 1, 2$ are all equal to a function δ independent of i, where

$$\begin{aligned}
\delta &= \frac{p(k)}{q(k)} \\
p(k) &:= k(k^2+k+1)(k^2-k+1)\cdot \\
&\quad (32\cdot k^5 - 120\cdot k^4 + 152\cdot k^3 - 69\cdot k^2 + 14\cdot k + 12) \\
q(k) &:= 3(1024\cdot k^{10} - 3204\cdot k^9 + 4900\cdot k^8 - 4404\cdot k^7 + 3991\cdot k^6 \\
&\quad -3228\cdot k^5 + 2869\cdot k^4 - 1410\cdot k^3 + 757\cdot k^2 - 210\cdot k + 140)
\end{aligned}$$

Both $p(k)$ and $q(k)$ are greater 0 for all $k \geq 3$. For example this can be seen easily by taking a look at Horner's scheme representation of both polynomials for $k \geq 4$ and checking additionally for $k = 3$. Also Matlab finds 10 values each for which $p(k)$ or $q(k)$ are 0. All of these values have an Euclidian norm of less than 2 within the complex plane. Thus, δ is greater 0 for all $k \geq 3$ which proves that $x_3 < x_2 < x_1 < x_0$.

It remains to be shown that $x_3 > 0$. From the same Matlab computations, it emerged

$$\begin{aligned}
x_3 &= 3\cdot\frac{p_3(k)}{q(k)} \\
p_3(k) &:= 224\cdot k^{11} - 876\cdot k^{10} + 1572\cdot k^9 - 1455\cdot k^8 + 999\cdot k^7 \\
&\quad -846\cdot k^6 + 840\cdot k^5 - \frac{723}{2}\cdot k^4 + 168k^3 - 9\cdot k^2 + 56\cdot k + 21
\end{aligned}$$

In the same fashion as before it can be seen that in fact $x_3 > 0$ for all $k \geq 3$. Consequently, this proves the claim

$$0 < x_3 < x_2 < x_1 < x_0$$

□

Proposition 4 finally yields that the clause-current constraint $\bar{f}_C = \frac{x_0+x_1}{2}$ is well defined and is fulfilled for each clause in F when the respective clause-block is connected to at least one enhanced literal. The only part which is left to prove is why the trivial instance does not obey the clause-current constraints. This is achieved by Proposition 5.

Proposition 5. *The trivial TNEP instance does not obey the clause-current-constraints $\bar{f}_C$.*

Proof of Proposition 5. For the trivial instance, the general case involving n_0, n_1, n_2 can be simplified to $n_1, n_2 = 0$ and $n_0 = k$ which leads to an even smaller TNEP scenario. Here, there is only one single variable-block-type and also one single clause-block-type leading to a size of at most $6+4+2 = 12$ nodes. The respective linear equations feature

k as the only symbolic variable involved. The solution received for the voltage potential of v_C by a computer-aided solver is a rational function of k that can be compared to the results of Proposition 4. Denote by x_t the current running through each clause-block of the trivial instance, then Matlab provided:

$$\begin{aligned}
x_t &= \frac{p_t(k)}{q_t(k)} \\
p_t(k) &:= (20 \cdot k)\frac{1}{11} - (30 \cdot k^2)\frac{1}{11} + \frac{3}{2} \\
q_t(k) &:= (k \cdot (11 \cdot k^2 - 12 \cdot k + 8)) + \frac{3}{11}
\end{aligned}$$

and the difference to x_0 (omitting the denominator) is given by $\delta_{t,0}$, where

$$\begin{aligned}
\delta_{t,0} &= ((32 \cdot k^5 - 120 \cdot k^4 + 152 \cdot k^3 - 69 \cdot k^2 + 14 \cdot k + 12) \\
&\quad \cdot (70 \cdot k^8 - 168 \cdot k^7 + 152 \cdot k^6 - 78 \cdot k^5 + 134 \cdot k^4 - 78 \cdot k^3 + 73 \cdot k^2 + 21))
\end{aligned}$$

which is greater zero for all $k > 3$ which can be seen using the same technique as above. Thus, it also holds that

$$x_0 < x_t$$

which yields the claim. □

Consequently, to solve the TNEP scenario it is always required to add at least one edge to the variable-cells. This makes sure the trivial instance can never be a solution.

All the previously proven Propositions can now be combined to prove Theorem 4.

Proof of Theorem 4. From Proposition 1 to 5 it follows

- Given a 3-SAT-matrix A, a corresponding TNEP scenario S can be obtained in polynomial time.
- A TNEP instance $\vec{n}$ for S obeys the constraints $\bar{f}_v$ iff $\vec{n}$ is a SAT-consistent instance (or trivial).
- The trivial TNEP instance does not obey any clause-constraint $\bar{f}_C$, hence at least one variable-cell needs to be enhanced.
- For any SAT-consistent instance $\vec{n}$ a clause-constraint $\bar{f}_C$ on clause-block j is fulfilled $\iff$ t_{C_j} is connected to at least one enhanced literal node w.r.t. $\vec{n}$.

Thus, F is satisfiable iff there exists a solution to the respective TNEP scenario S. Hence Algorithm 1 and Definition 12 describe a polynomial reduction from 3-SAT to TNEP, q.e.d. □

4. Evaluation

Many approaches have been tried to deal with TNEP in practice. In its original form as presented in this work, TNEP is a Mixed Integer Non-Linear Programming (MINLP) problem. Due to the generally high computational complexity of this type of problem, it is often the case that simplified versions of TNEP are considered which include linearization and/or relaxtions. Different ways to model TNEP can be found in [RMGH02] together with a set of test systems. The model used in this work is generally referred to as the DC system as in [RMGH02]. An overview of works on TNEP can be found in [LCAV03] or [LNZW06] where the latter includes considerations on regulated vs. deregulated (market) environments. Usually, the published algorithms on TNEP are classified into *mathematical optimization*, *heuristic* or *meta-heuristic* approaches.

A classical heuristic approach can be found in the original Garver paper [Gar70]. These types of algorithms usually rely on (engineering) experience to simplify the search. The downside of a low computational effort is that they may lead to poor solutions in larger networks.

Meta-heuristic approaches try to combine the benefits of both mathematical and heuristic approaches. This includes the use of Genetic Algorithms as well as Simulated Annealing [RGM96] or Particle-swarm optimization [TCPG11].

For the mathematical approaches, TNEP is solved by classical optimization techniques that generally use simplifications of the problem by linearizing and relaxing it so that linear programming (LP) or mixed integer linear programming (MIP) can be applied [AMC03]. Depending on the model chosen purely mathematical approaches tend to require a high computational effort for large networks. Usually this is done for DC systems, however [TH11] for example includes the use of AC[1] systems.

The big-M-notation is a typical linearization of the integer choice constraints also used widely for TNEP. A comprehensive study using big-M-notation including a survey on the quality of the results can be found in [MPS10].

[RGR08] uses a branch-and-bound algorithm for solving MINLP specialized for TNEP. In this evaluation however, it should be examined how well a general but highly sophisticated MINLP solver in a state-of-the-art hardware environment can deal with the original formulation of the problem without any further simplifications/relaxations of Definition 8 in order to see at which input sizes the problem becomes practically infeasible with this

[1]alternate current

approach. As shown in the previous sections, TNEP inevitably embodies the computational difficulty of NP-completeness. Thus, on the one hand it is unlikely that an efficient algorithm optimally solving all TNEP scenarios in general is ever found and on the other hand TNEP may benefit of using a generic MINLP solver which is generally designed and suitable to tackle NP-hard and NP-complete problems (among others).

Moreover, unlike in [RGR08], the MINLP solver used in this work is able to guarantee that the optimum found is global.

4.1. Model adjustment NR/WR

The DC model considered in the previous chapters assumes the supplies to be fixed with $\sum_i b_i = 0$. In the literature, e.g. [RMGH02], this is called a DC model *without redispatch* meaning it is not allowed to change the (power) supply/output of any node. It is also possible to set a maximum supply $\bar{b}_i$ for each node with variable b_i ($0 \leq b_i \leq \bar{b}_i$ for supply nodes) where the sum of maximum supplies is being greater or equal than sum of demands. This is called a DC model *with redispatch*. In the following sections, the term *without redispatch* will be abbreviated by *NR* as opposed to *WR* for the model *with redispatch*.

In terms of *practical* computational complexity, NR is considered to be more difficult than WR which can be seen in [RMGH02]. WR includes the related problem of Optimal Power Flow (analysis), which can be solved efficiently even for AC systems, see [DT68]. This is the probable reason why computational complexity does *not* increase when switching from NR to WR. Nevertheless, the *theoretical* complexity of the WR model is still NP-complete which can be seen by two simple adaptions of the NP-hardness proof in chapter 3: The first option is to set trivial upper supply bounds equal to the given supply levels so each source needs to operate at its boundary supply level anyway to fulfil all demands. The second option featuring more variety is to split up the super-source into multiple nodes, let them have a supply between 0 and m and connect them to the graph via one single edge with capacity m. This way, the redispatch can be chosen freely although it does not affect the network's behaviour. This is illustrated in Figure 4.1. h_C denotes the position of the original super-source s_C which has been split up into s_C^1, s_C^2, s_C^3. Here, m denotes the total number of 3-SAT clauses for a given k.

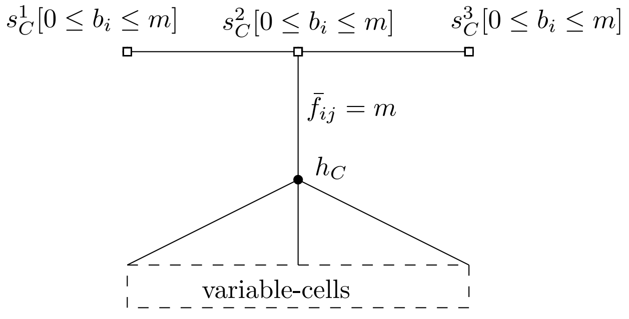
$s_C^1[0 \leq b_i \leq m]$
$s_C^2[0 \leq b_i \leq m]$
$s_C^3[0 \leq b_i \leq m]$
$\bar{f}_{ij} = m$
h_C
variable-cells

Figure 4.1.: NP-hardness of WR model from the NR model

The focus in this chapter will thus be on the NR model but the WR model was also tested where applicable. As will be seen for the test cases the WR model was always solved faster than the NR model.

4.2. BARON and the NEOS server

The Network-Enabled Optimization System (NEOS) is an internet-based client-server optimization service. It is free to use and offers a whole variety of sophisticated solvers for mathematical optimization problems like LP, MILP, MINLP and many more. A design and implementation overview is available in [CMM98] together with an administrative guide in [Dol01]. Further discussion on NEOS can be found in [PBI97]. Problem formulations are typically written using the General Algebraic Modelling System (GAMS). GAMS documentation together with an Integrated Development Environment is available at [gam]. The modelling language featured by GAMS is very close to typical scientific formulations used to describe mathematical (optimization) problems. Documentation and tutorials are also available at [gam].

For the tests, the Branch-and-Reduce Optimization Navigator (BARON) has been used. BARON is a GAMS solver preferably designed for MINLP. A quick online documentation is also available at [gam]. Details on its features and illustrations of its use can be found in [TS02] which was written by the BARON developers. Unlike most other MINLP solvers, BARON is capable of guaranteeing to provide a global optimum under fairly general assumptions, as stated in the BARON documentation. The most important assumption for practical use is probably that every variable and expression needs to be bounded strictly. See 4.2.1 for details.

If desired by the user, BARON is suitable for finding the the best, second best, third best, etc. solution. It can also be used to search for all feasible solutions. See the BARON documentations for details.

The GAMS code used for the tests in these sections can be found in appendix C. All tests have been performed using the NEOS-server with BARON.

4.2.1. Practical Concerns

The performed tests have shown that using the NEOS-server with BARON in practice may require a little numerical “sensitivity”.

Here is some practical advice that may be helpful when trying to reproduce our results:

- Define costs as integers
- Bound all variables explicitly (if necessary to the default bound of ± 1000)
- For scaled values: feed sufficiently big mantissa to BARON
- Relieving flow bounds e.g. by 5% may compensate numerical inaccuracy

BARON allows the user to define branching priorities for use in the computation. The default value is 1 for each variable where higher values denote higher priorities [2]. After some tests it emerged that a branching priority in $[2, 10]$ for n_{ij} and leaving the others (for f_{ij} and p_i) at 1 generally delivered promising results in terms of running time although the differences from the default setting were not too extensive and varied for different test cases. For the WR model, the branching priority of the supply b_i was mostly set to 2 acknowledging the fact that the search space is also dependent of b_i for this model.

Another important BARON option is the *relative termination tolerance* ϵ_r. This makes BARON stop its search whenever $U - L \leq \epsilon_r \cdot |L|$ where U is the cost of the best solution (or upper bound) found so far and L is the tightest lower bound found so far. Depending on the costs of the edges, it may be necessary to set this parameter to a smaller value than the default 0.1 in order to ensure the solution found by BARON is more likely to be a global optimum. In the test cases considered, the global optimum was always found for $\epsilon_r \leq 0.01$ (with an exception for the Brazil South East system, see below). In addition to the relative one there is also an *absolute termination tolerance* ϵ_a with a default value

[2]this is the opposite in GAMS itself which may be confusing

of 10^{-9}. BARON terminates if $U - L \leq \epsilon_a$. When integer costs are considered only, a sufficiently small ϵ_r together with an ϵ_a slightly smaller than 1 will always deliver a global optimum if the computation finishes in time.

4.3. Test cases

The test cases considered here are the same as those in [MPS10]. Most of the cases are also noted in [RMGH02].

Table 4.1 contains an overview of the data considered here.

Name	$\|V\|$	$\|conn(E)\|$	references
Garver	6	15	[RMGH02], [Gar70]
IEEE 24 bus	24	34	[AMC03], ([MPS10])
Brazil South	46	79	[Bin00], ([MPS10])
Brazil South East	79	143	[Bin00], ([MPS10])

Table 4.1.: Test cases considered

In all the test cases, each connection was allowed to have at most 3 more edges independent of the number of initial edges of that connection. It is known from previous works that the optimal solution (or one of the optimal solutions) is contained in this search space for each of the test cases. These bounds were also used in [MPS10]. Restrictions like these are essential for the use of a global optimization routine as they determine the size of the search space. Nevertheless too strict bounds for the number of edges added may lead to sub-optimal results or even infeasible scenarios. Thus, in general this upper bound should be chosen carefully.

Theoretically, the TNEP search space R in the NR model for each scenario is the space of its (valid) instances $\vec{n} \in \mathbb{N}_0^m$, i.e. $R = \{\vec{n} \mid n_{ij}^{(0)} \leq n_{ij} \leq \bar{n}_{ij} \quad \forall n_{ij} \in \vec{n}\}$. Here f_{ij} and p_i are only auxiliary variables being solely determined by $\vec{n}$ and the fixed parameters due to $L \cdot p = b$, equation 2.5. This yields table 4.2

Name	search space size	
Garver	3^{15}	$\approx 1.43 \cdot 10^7$
IEEE 24 bus	3^{41}	$\approx 3.65 \cdot 10^{19}$
Brazil South	3^{79}	$\approx 4.93 \cdot 10^{37}$
Brazil South East	3^{143}	$\approx 1.7 \cdot 10^{68}$

Table 4.2.: Theoretical search space sizes

In order to enable comparability with future improved computing hardware, future BARON versions and similar branching algorithms, the pure computation time results are accompanied by a (typical) number of branch-and-reduce (BaR) iterations performed by BARON.

The relative termination tolerance ϵ_r has a crucial influence on the runtime of BARON while at the same time increasing the probability of finding a global optimum. It was hence used as the most relevant benchmark parameter in this chapter. Additionally, tests featuring an absolute termination tolerance ϵ_a of 0.999 (with integer costs and sufficiently small ϵ_r) were also performed in order to illustrate the maximum runtime needed to find a global optimum.

The solution values listed in this chapter are given in the same way as in [MPS10] for comparability. Nevertheless they were actually computed after scaling all cost values to integers first.

4.3.1. Garver

This test case has become one of the most prominent examples considered among TNEP researchers. Since it is very small and easy to solve it is only suitable for code testing and illustrating ideas. It has been used in example 8 and 9 as a TNEP scenario with solution.

NR model:
According to [RMGH02] there are five optimal solutions for the NR model. In each solution there are exactly 7 edges added at a total cost of 200. However, only two of these solutions are contained in the search space considered here where there are only three edges allowed to be added for each connection. To find the other solutions, this bound would have to be increased to five edges per connection.

In the tests applied, BARON found these supposedly optimal solutions for three addable edges to be infeasible. Instead, depending on the relative termination tolerance ϵ_r the best solutions found had a cost of 248 (for $\epsilon_r = 0.1$) or 231 (for $\epsilon_r \in \{10^{-2}, 10^{-3}\}$). This may be due to rounding errors in the model parameters. Nevertheless, further tests with Matlab provided that the proposed solutions for three addable edges resulted in a current that violated the capacity of the three edges added between node v_2 and v_6 (see Figure 2.7 and 2.8) by about 3 to 5 units per edge. However, some related works allow a small power shortage at each node that is not penalized, which may be the reason why these solutions are considered feasible there.

The optimal solution of 200 for three addable edges was only found if the flow bounds $\bar{f}_{ij}$ were relieved by 6.5%. However, for five addable edges an optimal solution at cost 200 was found indeed *without* relieving the flow bounds.

The test using $\epsilon_a = 0.999$ also provided the same solution of cost 231 so it has to be considered the optimal solution for the Garver case using the possibly inaccurate test parameters and strict fulfilling of the current demands/supplies.

All solutions for the NR model here were computed within less than 1 second by NEOS/BARON. The number of Branch-and-Reduce (BaR) iterations performed is given in Table 4.3

ϵ_r	0.1	0.01	0.001	$\epsilon_a = 0.999$
BaR	29	53	75	144

Table 4.3.: Garver NR: BaR iterations

WR model:
The optimal value given in [RMGH02] and [MPS10] is 110. In this case, the inaccuracy of the parameters was not an issue as the optimal value was found immediately by BARON for all $\epsilon_r \in \{10^{-1}, 10^{-2}, 10^{-3}\}$

Like in the NR model all solutions were found within less than 1 second.

See Table 4.4 for the number of BaR iterations.

ϵ_r	0.1	0.01	0.001	$\epsilon_a = 0.999$
BaR	5	11	11	11

Table 4.4.: Garver WR: BaR iterations

4.3.2. IEEE 24 bus

This is a typical test system related to electrical transmission network also suitable for questions beyond TNEP. The complete data including a topological overview can be found in [GWA+99].

NR model
The model used for TNEP in literature is usually considered for the WR model only, thus the test for the NR model used fixed b_i in the following manner: The supply of each node was set according to its share of the maximum total supply.

The optimal solution found is of cost 310. All considered ϵ_r found this solution.The computation times ranged from about 1.7 to 3.95 seconds for $\epsilon_r \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. It took about 5.2 seconds for $\epsilon_a = 0.999$.

The number of Branch-and-Reduce (BaR) iterations performed is given in Table 4.5

ϵ_r	0.1	0.01	0.001	$\epsilon_a = 0.999$
BaR	125	389	441	543

Table 4.5.: IEEE 24 NR: BaR iterations

WR model
The optimal value reported in [MPS10] is 152 which was found by BARON for all ϵ_r considered here.

The computation times ranged from approx. 1 to 1.5 seconds. See Table 4.6 for the number of BaR iterations.

ϵ_r	0.1	0.01	0.001	0.0001	$\epsilon_a = 0.999$
BaR	103	229	175	183	151

Table 4.6.: IEEE 24 WR: BaR iterations

It is a bit surprising that in this case the number of BaR iterations (and total computation time) is smaller for $\epsilon_r = 10^{-3}$. This behaviour was confirmed by several tests for this case. It may be due to a slightly increased preprocessing time resulting in tighter initial bounds.

4.3.3. Brazil South

This test case is taken from a realistic TNEP scenario. It is a medium sized network that has a few unconnected parts in it. A topological overview can be seen in [HMG+00]. This example is the first one of the ones considered which requires notable computation time.

NR model:
The optimal value of 154.4 reported in [MPS10] was found for all ϵ_r considered. See Table 4.7 for BaR iterations and computation times.

ϵ_r	0.1	0.01	0.001	$\epsilon_a = 0.999$
BaR	27786	137627	157313	151590
Time (s)	971	5346	6417	5936

Table 4.7.: South Brazil NR: BaR it. and computation time

WR model:
The optimal value of 72.87 reported in [MPS10] was found for all ϵ_r considered.

ϵ_r	0.1	0.01	0.001	0.0001	$\epsilon_a = 0.999$
BaR	2301	3187	2049	4050	5120

Table 4.8.: South Brazil WR: BaR it.

The computations times ranged from roughly 50 to 100 seconds for $\epsilon_r \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. For $\epsilon_a = 0.999$ it took about 200 seconds. See Table 4.8 for BaR iterations.

Just like in the 24 bus system, the BaR iterations (and computation time) decreased when switching ϵ_r from 10^{-2} to 10^{-3} in a counter-intuitive way. Again, this behaviour was confirmed by multiple additional tests. However, as can be seen for $\epsilon_r = 10^{-4}$, further tightening of the bound results in an increased number of BaR iterations again for this case.

4.3.4. Brazil South East

Like the previous one, this case is also taken from a realistic scenario.

NR model
All test cases for this system were stopped when NEOS' default runtime barrier of 8 hours was reached. The best solution found until that point had a cost of 463.3. The optimal value reported in [MPS10] is at 424.8 so the best value found is roughly 9% more expensive than the optimal solution. However, as further tests have shown, this value of 463.3 is already found by BARON after approximately $1h : 15m$. The search space seems too large to find a better value within the remaining $6h : 45m$. Moreover, the solution quality worsened when starting with tighter ϵ_r bounds.

Apart from that, it is not known, if the TNEP solution of cost 424.8 would be considered feasible by BARON within the data used in these tests. BARON typically terminated after 8 hours with a lower bound of about 276.1 so the relative gap is still at about 0.6776 as opposed to the desired ϵ_r of at most 0.1.

See Table 4.9 for BaR iterations and best solutions found after $8h$ of computation time.

ϵ_r	0.1	0.01	0.001	$\epsilon_a = 0.999$
BaR	198334	243748	250893	175667
best found	463.3	505.6	510.3	498.0

Table 4.9.: South Brazil NR: BaR it. and best solution found after $8h$

4.4. Result overview

Table 4.10 illustrates the typical runtime of NEOS using BARON for different relative termination tolerances ϵ_r.

Thus, in terms of computation time, all test scenarios excluding the Brazil South East system can could be solved very efficiently. The computation times for both the Garver and the IEEE 24 bus system are so small that hundreds of these systems could be solved within minutes [3]. For the medium sized Brazil South system (46 nodes) the WR model could be solved very quickly too, whereas the NR system already took considerable amounts of time to be solved with tighter termination tolerances. The South East system, however, took quite long and reached the default computation limits of NEOS. Since on the one

[3]in this work, NEOS was used manually via a web form but it is also possible to submit data automatically using an interface called Kestrel. See NEOS documentation for details.

	for $\epsilon_r =$			for $\epsilon_a =$
Name	0.1	0.01	0.001	0.999
Garver NR	$0:00:01$	$0:00:01$	$0:00:01$	$0:00:01$
Garver WR	$0:00:01$	$0:00:01$	$0:00:01$	$0:00:01$
IEEE 24 NR	$0:00:02$	$0:00:04$	$0:00:04$	$0:00:05$
IEEE 24 WR	$0:00:01$	$0:00:01$	$0:00:01$	$0:00:01$
Brazil South NR	$0:16:12$	$1:29:07$	$1:46:58$	$1:38:56$
Brazil South WR	$0:01:22$	$0:01:38$	$0:00:51$	$0:03:22$
Brazil South East NR	$8:00:00$*	$8:00:00$*	$8:00:00$*	$8:00:00$*

Table 4.10.: typical BARON runtime $h:mm:ss$ on NEOS

hand, the planning process in real-world applications of TNEP is usually performed over a period of weeks, months or even years this computation time should still be tolerable. On the other hand, the upper and lower bound given by BARON can still provides valuable information for the planning process.

Table 4.11 denotes for which ϵ_r the optimum was found (first value in table) and for which cases ϵ_r is small enough to ensure the solution found is *trivially* the global optimum (second value). Here, *trivially* means the termination tolerances ϵ_r or ϵ_a are tight enough to ensure there exists no other TNEP instance between the lowest bound and the best solution found.

	found/trivially global for $\epsilon_r =$			$\epsilon_a =$
Name	0.1	0.01	0.001	0.999
Garver NR	n/n**	y/n**	y/y**	y/y**
Garver WR	y/n	y/n	y/y	y/y
IEEE 24 NR	y/n	y/n	y/y	y/y
IEEE 24 WR	y/n	y/n	y/y	y/y
Brazil South NR	y/n	y/n	y/n	y/y
Brazil South WR	y/n	y/n	y/n	y/y
Brazil South East NR	unk/n	n/n	n/n	n/n

Table 4.11.: optimum found vs. proven global (trivially) ((y)es, (n)o, or (unk)nown)

The asterisks '*' and '**' denote the following:

'*' best solution already found after approx. $1h$ and $15min$ (for South East system)

'**' referencing an optimum of 231 instead of 200, see section 4.3.1

5. Conclusion

Transmission network expansion planning, as examined in this work, deals with the question of where to add new transmission lines at minimum cost in case the network needs to be expanded for future increased power demand without violating the physical upper bounds of the power transported in each line. It has been shown in this work, that by omitting the subproblem of Steiner trees by allowing (highly) connected networks only, TNEP is still an NP-complete problem. This was achieved by a polynomial-time reduction of 3-SAT to TNEP. To achieve this it was necessary to find a map from a 3-SAT to a TNEP scenario that can transport the semantics of boolean formulas to the semantics of TNEP. The key steps were as follows:

- Identifying boolean variables with *variable-cells* where adding an edge corresponds with setting the variable to true or false,
- use of a "symmetry sensor" to ensure each variable is set to either true or false exclusively and
- identifying clauses with *clause-blocks* whose flow constraints are fulfilled if the current entering them is running via sufficiently many enhanced variable-cells.

The analysis of this construction made use of electrical network theory. This enables the simplification of the highly symmetric graphs used for the analysis by short-cutting and merging symmetric nodes, which allowed to employ Matlab for solving a fixed size system of symbolic linear equations in order to prove correctness of the described reduction procedure.

This underlines the fact that, from a computational viewpoint, TNEP must inevitably[1] bear a high computational effort to solve optimally – even when the NP-complete subproblem of Steiner trees is omitted. This proves why theoretically TNEP is computationally equivalent to the well known classical NP-complete problems like SAT, the travelling salesman or Knapsack. Hence, in order to solve TNEP efficiently for big scenarios, it is necessary to make use of approximations or/and specific network structures and details. For example, all solutions found for the realistic test cases considered only need a relatively small amount of additional edges (8 edges for the 24 node system, 16 edges for the 46 node system and 28 edges for the 79 nodes system; all NR model).

[1]at least if $P \neq NP$

In the tests executed, a general sophisticated MINLP solver suitable to prove global optima (i.e. BARON) in a state-of-the-art hardware environment (i.e. NEOS) performed generally well. In details:

- test system with 6 nodes was solved within milliseconds
- test system with 24 nodes was solved within 4 seconds or less
- test system with 46 nodes was solved within less than 100 seconds for the WR model and took about up to 100 minutes to solve the NR system when optimality had to be proven
- test system with 79 nodes was too big to prove within $8h$ that the optimum found was global, however a solution with 9% higher cost than the reported optimum was already found after about 75 minutes.

Thus, according to the tests, the systems that can efficiently be solved with providing a global optimum are seemingly of size somewhere between 50 and 80 nodes. For bigger systems suitable solutions may still be found, however in order to guarantee the optimum found is global, the computation time seems to become too extensive.

Moreover, it seemed that the WR models in the tests could be solved rather efficiently. Although the focus of this work was on the NR model, this illustrates one of the biggest advantages of the MINLP approach: It can be adapted very easily by performing only slight changes to the GAMS model while keeping both the solver and the environment fixed. For example, generation costs and power shortage penalties can be introduced for the WR model or even the original AC formulation can be used.

5.1. Future work

Electrical transmission networks are typically planar as examined in [RCC09]. That is, the graph can be drawn on a plane without edges intersecting. However, this is not the case for the network used to illustrate NP-hardness of TNEP. Thus it may be interesting to see if NP-hardness can also be shown for the subclass of planar networks.

In the applied research it may be favourable to integrate MINLP solvers into methods that find approximative solutions for bigger scenarios than presented here, for example to find upper and lower bounds or to find suitable starting points. Also approaches that need the use of solving smaller subproblems of a big TNEP scenario can make extensive use of the MINLP approach with high quality (or optimal) solutions of the subproblems within low computation time requiring only a small implementation effort.

Bibliography

[AMC03] N. Alguacil, A. Motto, and A. Conejo, "Transmission expansion planning: a mixed-integer lp approach," *Power Systems, IEEE Transactions on*, vol. 18, no. 3, pp. 1070 – 1077, aug. 2003.

[BB09] U. Bakshi and A. Bakshi, *Network analysis & synthesis.* Technical Publications, 2009. [Online]. Available: http://books.google.de/books?id=Yw9dTgZI8e8C

[BE05] U. Brandes and T. Erlebach, *Network Analysis - Methodological Foundations.* Springer, 2005.

[Big93] N. Biggs, *Algebraic graph theory*, ser. Cambridge mathematical library. Cambridge University Press, 1993. [Online]. Available: http://books.google.com/books?id=6TasRmIFOxQC

[Bin00] S. Binato, "Optimal expansion of transmission networks by benders decomposition and cutting planes," Ph.D. dissertation, Federal University of Rio de Janeiro, 2000.

[Bol98] B. Bollobás, *Modern Graph Theory.* Springer, 1998.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[CMM98] J. Czyzyk, M. P. Mesnier, and J. J. Moré, "The neos server," *IEEE Comput. Sci. Eng.*, vol. 5, no. 3, pp. 68–75, Jul. 1998. [Online]. Available: http://dx.doi.org/10.1109/99.714603

[Die06] R. Diestel, *Graph theory*, ser. Graduate texts in mathematics. Springer, 2006. [Online]. Available: http://books.google.com/books?id=aR2TMYQr2CMC

[Dol01] E. D. Dolan, "Neos server 4.0 administrative guide," *CoRR*, vol. cs.DC/0107034, 2001.

[DT68] H. Dommel and W. Tinney, "Optimal power flow solutions," *Power Apparatus and Systems, IEEE Transactions on*, vol. PAS-87, no. 10, pp. 1866 –1876, oct. 1968.

[gam] [Online]. Available: http://www.gams.com

[Gar70] L. Garver, "Transmission network estimation using linear programming," *Power Apparatus and Systems, IEEE Transactions on*, vol. PAS-89, no. 7, pp. 1688 –1697, sept. 1970.

[GWA$^+$99] C. Grigg, P. Wong, P. Albrecht, R. Allan, M. Bhavaraju, R. Billinton, Q. Chen, C. Fong, S. Haddad, S. Kuruganty, W. Li, R. Mukerji, D. Patton, N. Rau, D. Reppen, A. Schneider, M. Shahidehpour, and C. Singh, "The ieee reliability test system-1996. a report prepared by the reliability test system task force of the application of probability methods subcommittee," *Power Systems, IEEE Transactions on*, vol. 14, no. 3, pp. 1010 –1020, aug 1999.

[HMG$^+$00] S. Haffner, A. Monticelli, A. Garcia, J. Mantovani, and R. Romero, "Branch and bound algorithm for transmission system expansion planning using a transportation model," *Generation, Transmission and Distribution, IEE Proceedings-*, vol. 147, no. 3, pp. 149 –156, may 2000.

[LCAV03] G. Latorre, R. Cruz, J. Areiza, and A. Villegas, "Classification of publications and models on transmission expansion planning," *Power Systems, IEEE Transactions on*, vol. 18, no. 2, pp. 938 – 946, may 2003.

[LNZW06] C. Lee, S. Ng, J. Zhong, and F. Wu, "Transmission expansion planning from past to future," pp. 257 –265, 29 2006-nov. 1 2006.

[MAT11] MATLAB, *version 7.13.0 (R2011b).* Natick, Massachusetts: The MathWorks Inc., 2011. [Online]. Available: http://www.mathworks.com

[MPS10] L. S. Moulin, M. Poss, and C. Sagastizábal, "Transmission expansion planning with re-design," *Energy Systems*, vol. 1, no. 2, pp. 113–139, 2010.

[PBI97] M. Powell, M. Buhmann, and A. Iserles, *Approximation Theory and Optimization: Tributes to M.J.D. Powell.* Cambridge University Press, 1997. [Online]. Available: http://books.google.com/books?id=IchVIe9AZk8C

[RCC09] M. Rosas-Casals and B. Corominas, "Assessing european power grid reliability by means of topological measures," *WIT transactions on ecology and the environment*, vol. 121, pp. 527–537, 2009.

[RGM96] R. Romero, R. Gallego, and A. Monticelli, "Transmission system expansion planning by simulated annealing," *Power Systems, IEEE Transactions on*, vol. 11, no. 1, pp. 364 –369, feb 1996.

[RGR08] M. Rider, A. Garcia, and R. Romero, "Transmission system expansion planning by a branch-and-bound algorithm," *Generation, Transmission Distribution, IET*, vol. 2, no. 1, pp. 90 –99, january 2008.

[RMGH02] R. Romero, A. Monticelli, A. Garcia, and S. Haffner, "Test systems and mathematical models for transmission network expansion planning," *Generation, Transmission and Distribution, IEE Proceedings-*, vol. 149, no. 1, pp. 27 – 36, jan 2002.

[Sci11] Scilab Consortium, "Scilab: Free and open source software for numerical computation," 2011. [Online]. Available: http://www.scilab.org

[TCPG11] S. Torres, C. Castro, R. Pringles, and W. Guaman, "Comparison of particle swarm based meta-heuristics for the electric transmission network expansion planning problem," pp. 1 –7, july 2011.

[TH11] J. Taylor and F. Hover, "Linear relaxations for transmission system planning," *Power Systems, IEEE Transactions on*, vol. 26, no. 4, pp. 2533 –2538, nov. 2011.

[TS02] M. Tawarmalani and N. Sahinidis, *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*, ser. Nonconvex optimization and its applications. Kluwer Academic Publishers, 2002. [Online]. Available: http://books.google.com/books?id=MjueCVdGZfoC

[vM06] A. von Meier, *Electric Power Systems: A Conceptual Introduction.* Wiley-Interscience, 2006.

[Weg87] I. Wegener, *The Complexity of Boolean Functions.* John Wiley & Sons Ltd, and B.G. Teubner, Stuttgart, 1987.

[WW96] A. J. Wood and B. F. Wollenberg, *Power Generation, Operation, and Control.* Wiley-Interscience, Jan 1996.

Appendix

A. Code: Reduction 3-SAT to TNEP

The following listing contains source code written in Scilab that allows to perform the polynomial time transformation from a 3-SAT problem to a corresponding TNEP-problem. The input expects a 3-SAT matrix $A \in \{-1,0,1\}^{m\times k}$ for k variables and m clauses where $a_{ij} = -1$ denotes the negated ($1 =$ non-negated) appearance of variable j in clause i. "0" denotes the absence of a variable within a clause such that $\sum_i |a_{ij}| = 3$ for all $i = 1,\ldots,m$.

Notice that in the implementation, the conductances are slightly different than those mentioned in the the previous sections. This does not really matter since they are still within the same classes with respect to big-O-notation ($\Theta(k^2)$ etc.).

```
//////////////////////////////////////////////////////////////
// collection of functions to transform 3SAT—CNF
// to TNEP—problem
//////////////////////////////////////////////////////////////

///////////////////////////////////////
// global variables / constants
///////////////////////////////////////
global g_varBlockSize g_clauseBlockSize g_superSourceBlockSize
g_variableBlockSize = 6;
g_clauseBlockSize = 2;
g_superSourceBlockSize = 1;
g_superInterConSize = 1;
g_3clauseCombination=[ 1  1  1;
                  1  1 −1;
                  1 −1  1;
                  1 −1 −1;
                 −1  1  1;
                 −1  1 −1;
                 −1 −1  1;
                 −1 −1 −1];

///////////////////////////////////////
// number of total clauses for k variables
// in 3SAT scenario
///////////////////////////////////////
function y = numTotalClauses(k)
    y = 4*k*(k−1)*(k−2)/3;
endfunction

///////////////////////////////////////
// number of clauses connected to one literal
// for a k variables 3SAT scenario
///////////////////////////////////////
function y = numConnectedClauses(k)
    y = 2*(k−1)*(k−2);
endfunction
```

```
////////////////////////////////////////
// input: matrix in 3-SAT CNF. m Rows=clauses,
//         k columns=variables
//         format a_ij in {-1,0,1} for negated,
//         not used, unnegated occurence
// output: inflated 3-SAT formula such that EACH possible
//         clause is contained in y
//         (determined by number of variables only).
//          The first rows in y will be A, i.e. y(1:m, :) = A
// k >= 3 assumed
////////////////////////////////////////
function y = SAT3Inflate( A )
    [m k] = size(A);
    // dimensions of new matrix
    n = numTotalClauses(k);
    y = zeros(n,k);
    y(1:m, 1:k) = A;

    // generate temporary full matrix
    // representing all possible clauses
    allClauses = zeros(n,k);
    // represents all possible clauses made out of 3 variables
    combi = g_3clauseCombination;
    // iterate all variable-subsets of size 3
    row = 1;
    for i1 = 1:(k-2)
        for i2 = i1+1:(k-1)
            for i3 = i2+1:k
                allClauses(row:(row+7), i1) = combi(:,1);
                allClauses(row:(row+7), i2) = combi(:,2);
                allClauses(row:(row+7), i3) = combi(:,3);
                row = row+8;
            end
        end
    end

    // omit sorting to save space (does not copy
    // input matrix A)
    keepRow = ones(n,1);
    for i=1:m
        keepRow =..
             keepRow & or( allClauses*(A(i,:)') <> 3, 'c');
    end
    // save without duplicates
    y((m+1):n, : ) = allClauses(keepRow,:);
endfunction


////////////////////////////////////////
// transform inflated SAT3 to TNEP
////////////////////////////////////////
function [y, b] = SAT3InflatedToTNEP( A )
    [n,h] = size(A);
    // prepare block matrizes for variables (x_i), clauses
    [VarBlock, VarSupply] = createVariableBlock(h);
    [ClauseBlock, ClauseSupply ] = createClauseBlock(1,-1);
    [SuperSourceBlock, SuperSourceSupply] = ..
            createSuperSourceBlock(1, n);
    [vbRows vbCols] = size(VarBlock);
    [cbRows cbCols] = size(ClauseBlock);
    [ssRows ssCols] = size(SuperSourceBlock);

    icSize = g_superInterConSize; //size of interconnect-block
    totalSize = ssRows + h*vbRows + 2 + icSize + n*cbRows;
    y = zeros(totalSize, totalSize);
    b = zeros(totalSize, 1);

    // (1.1) create supersource
    b(1) = SuperSourceSupply;
    rowOffset = ssRows;

    // (1.2) create var-source and var-sink
    b(rowOffset+1) = 1;
    b(rowOffset+2) = -1;
    rowOffset = rowOffset + 2;

    // (2.1) create blocks for variables
    for i=1:h
        row = rowOffset+(i-1)*vbRows+(1:vbRows);
        col = rowOffset+(i-1)*vbCols+(1:vbCols);
        y( row, col ) = VarBlock;
    end

    // (2.2) connect variables to super-source
    affNodes = rowOffset + (-1+ (1:h))*vbRows +2;
    node = 1;
    c = 1;
    y(node, affNodes ) = y(node, affNodes ) - c;
    y(affNodes, node ) = y(affNodes, node ) - c;

    affNodes = affNodes +2;
    y(node, affNodes ) = y(node, affNodes ) - c;
    y(affNodes, node ) = y(affNodes, node ) - c;

    // (2.3) connect variable-blocks to var-source and -sink
    affNodes = rowOffset + (-1+ (1:h))*vbRows +1;
    node = ssRows + 1;
    c = 1;
    // conn to source
    y(node, affNodes ) = y(node, affNodes ) - c;
    y(affNodes, node ) = y(affNodes, node ) - c;

    // conn to sink
    affNodes = affNodes +5;
    node = node+1;
    y(node, affNodes ) = y(node, affNodes ) - c;
    y(affNodes, node ) = y(affNodes, node ) - c;

    rowOffset = rowOffset + h*vbRows;
    // (3.1) create super-interconnect
    // update: block is merged to size of 1

    // (3.2) connect super-interconnect to variables
    // scale conductance according to number of clauses served
    // by each literal
    sicCond = numConnectedClauses(h); //2*(h-1)*(h-2);
    affNodes =ssRows + 2 + (-1+ (1:h))*vbRows+2;
    node = rowOffset+1;
    c = sicCond;
    y(node, affNodes ) = y(node, affNodes ) - c;
    y(affNodes, node ) = y(affNodes, node ) - c;

    affNodes = affNodes +2;
    y(node, affNodes ) = y(node, affNodes ) - c;
    y(affNodes, node ) = y(affNodes, node ) - c;

    rowOffset = rowOffset + icSize;
    // (4.1) create clause-blocks
    for i=1:n
        row = rowOffset + (i-1)*cbRows + (1:cbRows);
        col = rowOffset + (i-1)*cbCols + (1:cbCols);
        y( row, col ) = ClauseBlock;
        b( row ) = ClauseSupply;
    end

    // (4.2) connect clauses to super-interconnect
    affNodes = rowOffset + (-1 +(1:n))*cbRows +1;
    node = rowOffset;
    c = 1;
    y(node, affNodes ) = y(node, affNodes ) - c;
    y(affNodes, node ) = y(affNodes, node ) - c;

    // (4.3) shortcut clauses to their literals
    // todo: optimize by using matrix operations?
    for clause=1:n
        cNode = rowOffset+clause*cbRows;
        for literal=1:h
            if A(clause, literal) <> 0 then
                if A(clause, literal) == -1 then  // negated
                    lNode = ssRows+2 + (literal-1)*vbRows+5;
                elseif A(clause, literal) == 1 then
                    lNode = ssRows+2 + (literal-1)*vbRows+3;
                else
                    error("A should only contain -1, 0, 1.")
                    lNode = cNode;
                end
                y( cNode, lNode ) = y( cNode, lNode ) - 1;
                y( lNode, cNode ) = y( lNode, cNode ) - 1;
            end
        end
    end
    // to complete Laplacian: compute diagonal elements
    for i=1:totalSize
        y(i,i) = -sum(y(i,:));
    end
endfunction

///////////////////////////////////////
// input: 3SAT matrix
// result: y - Laplacian matrix of corresponding TNEP
//      b - supply vector
///////////////////////////////////////
function [y, b] = SAT3toTNEP( A )
    B=SAT3Inflate(A);
    [y, b]=SAT3InflatedToTNEP(B);
endfunction

///////////////////////////////////////
// input: k number of variables to consider (>=3)
// result: canonical 3SAT-clauses
///////////////////////////////////////
function A = canonicalSAT3(k)
    if k>=3 then
        A = zeros(1,k);
        A(1,1:3) = [1 1 1];
        A = SAT3Inflate( A );
    else
        error("k should be greater or equal 3")
    end
endfunction

///////////////////////////////////////
```

```
// input: k number of variables to consider (>=3)
// result: y - Laplacian matrix of corresponding TNEP
//      b - supply vector
///////////////////////////////////////
function [y, b] = canonicalSAT3toTNEP(k)
    A = canonicalSAT3(k);
    [y, b] = SAT3toTNEP( A );
endfunction

///////////////////////////////////////
// input: k number of variables
//        (could as well be computed from the size of L)
// result: matrices of varCurrents and clauseCurrents
//         one row corresponds to one k
///////////////////////////////////////
function [varCurrentsRows, clauseCurrentsRows] = ..
         evaluateCanonial(L, p, k)
    //4*k*(k-1)*(k-2)/3
    [varCurrentsRows, clauseCurrentsRows] =..
       evaluateCastedSAT3toTNEP( L, p, k, numTotalClauses(k));
endfunction

///////////////////////////////////////
// input:
//    L laplacian of casted 3SAT
//    p voltage potentials
//    numVars number of variables in the inflated graph
//    numClauses number of clauses in the inflated graph
///////////////////////////////////////
function [varCurrents, clauseCurrents] = ..
         evaluateCastedSAT3toTNEP( L, p, numVars, numClauses )
    varCurrents = getVarCurrents(1,numVars,L,p);
    clauseCurrents = ..
          getClauseCurrents(1,numClauses,L,p,numVars);
endfunction

///////////////////////////////////////
// input: y Laplacian of old graph
//      b demand/supply vector
//      alpha variable assignment in {-1, 0, 1} for each
//      variable meaning {false, don't-care, true}
// result: y - Laplacian matrix of corresponding TNEP
//      b supply vector
//      p voltage potentials
//      varCurrents currents running through variable blocks
//      clauseCurrents currents running through clause edge
// remarks: alpha should contain an assignment for exactly
//      each variable (including artificial ones)
///////////////////////////////////////
function [y, p, varCurrents, clauseCurrents] = ..
         assignVariablesNEvaluate( L, b, alpha)
    y = L;
    numVars = length(alpha);
    condToAdd = numConnectedClauses(numVars);
    [nodes nodes] = size(y);

    ssSize = g_superSourceBlockSize;
    vbSize = g_variableBlockSize;
    icSize = g_superInterConSize;

    numClauses = (nodes-ssSize-2 - icSize - numVars*vbSize)/2;
    // number of first literal node (negative literal)
    literals = ssSize+2 + (-1+ 1:numVars)*vbSize + 2;
    enhancePositiveBranch = (alpha == 1) * condToAdd;
    enhanceNegativeBranch = (alpha == -1) * condToAdd;
    // update negative branches
    for i=1:numVars
        y(literals(i), literals(i)+1) =..
                 y(literals(i), literals(i)+1)..
                 - enhanceNegativeBranch(i);
        y(literals(i)+1, literals(i)) =..
                 y(literals(i)+1, literals(i))..
                 - enhanceNegativeBranch(i);
        // diagonal elements
        y(literals(i),literals(i)) =..
                 y(literals(i),literals(i))..
                 + enhanceNegativeBranch(i);
        y(literals(i)+1,literals(i)+1) =..
                 y(literals(i)+1,literals(i)+1)..
                 + enhanceNegativeBranch(i);
    end

    // update posistive branches
    literals = literals + 2;
    for i=1:numVars
        y(literals(i), literals(i)+1) = ..
                 y(literals(i), literals(i)+1)..
                 - enhancePositiveBranch(i);
        y(literals(i)+1, literals(i)) = ..
                 y(literals(i)+1, literals(i))..
                 - enhancePositiveBranch(i);
        // diagonal elements
        y(literals(i),literals(i)) = ..
                 y(literals(i),literals(i))..
                 + enhancePositiveBranch(i);
        y(literals(i)+1,literals(i)+1) = ..
                 y(literals(i)+1,literals(i)+1)..
                 + enhancePositiveBranch(i);
    end

    // voltage
    p = computeVoltagePotentials(y,b);
    // evaluate currents
    [varCurrents, clauseCurrents] = ..
        evaluateCastedSAT3toTNEP( y, p, numVars, numClauses );
endfunction

///////////////////////////////////////
// creates one typical variable-block (reuse),
// view specification
//
// input: c conductance of regular branches
//        scale number of variables in the formula (k)
///////////////////////////////////////
function [y, b]= createVariableBlock( scale)
    y = zeros(g_variableBlockSize,g_variableBlockSize);
    b = zeros(g_variableBlockSize);

    // connect source to x_i^0, !x_i^0 and h^0
    // (scale-1)*(scale-2)*2;
    connectedClauses = numConnectedClauses(scale);
    sCon = 1/connectedClauses;

    y(1,[2 4]) = -sCon;
    bCon = connectedClauses;
    // connect x_i internally (high conductance
    // due to symmetry)
    y(2,3) = -bCon;
    y(4,5) = -bCon;

    // connector (build resistance for literal-interflow)
    y([3 5],6) = -sCon;
    // add transpose to complete
    y = y + y';
endfunction

///////////////////////////////////////
// create typical clause block,
// view specification
///////////////////////////////////////
function [y, b] = createClauseBlock(c, supply)
    y = zeros(2,2);
    b = zeros(2,1);
    b(2,1) = supply;

    // upper node to lower node
    y(2,1) = -c;
    y(1,2) = -c;
endfunction

///////////////////////////////////////
// create typical clause block,
// view specification
///////////////////////////////////////
function [y, b] = createSuperSourceBlock(c, supply)
    y = zeros(1,1);
    b = supply;
endfunction


///////////////////////////////////////
// solve current flow
// input: laplacian matrix L
//        demand  vector b
// output:
//        voltage potential solving L*p=b
//        with p($)=0 ground node
///////////////////////////////////////
function p = computeVoltagePotentials(L, b)
    dim = length(b);
    p = zeros(dim,1);
    // let last node be the ground node
    p(1:(dim-1)) = L(:,1:(dim-1))\b;
endfunction

///////////////////////////////////////
// returns currents running through the variable-blocks.
// input: var0 - number of first var to check
//        vark - number of last var to check
// return:
//     rows: vars
//     cols: first entry = helper current
//           second entry = var-current
///////////////////////////////////////
function y = getVarCurrents( var0, vark, L, p)
    n = vark-var0+1;
    y = zeros(n);

    vbSize = g_variableBlockSize;
    ssSize = g_superSourceBlockSize;
    rowOffset = (var0-1)*vbSize + ssSize+2;
    for i=1:n
        varNode = rowOffset+(i-1)*vbSize+6;
        sinkNode = 3;
        y(i) = getCurrent( varNode, sinkNode, L, p );
    end
endfunction

///////////////////////////////////////
// returns currents running through the clause-blocks
// specified by the first two parameters.
// input: cl0 - number of first clause to check
//        clk - number of last clause to check
//        numVars - number of variables in inflated formula
// return:
//     rows: clause currents
///////////////////////////////////////
function y = getClauseCurrents( cl0, clk, L, p, numVars)
    n = clk-cl0+1;
    y = zeros(n,1);

    // get block matrix dimensions
    vbSize = g_variableBlockSize;
    cbSize = g_clauseBlockSize;
    ssSize = g_superSourceBlockSize;
    icSize = g_superInterConSize;

    rowOffset = numVars*vbSize + ssSize+2 + ..
        (cl0-1)*cbSize + icSize;
    for i=1:n
        clauseNode = rowOffset+(i-1)*cbSize+1;
        y(i, 1) = getCurrent(clauseNode, clauseNode+1, L, p );
    end
endfunction

///////////////////////////////////////
// get current from node i to node j
//    L laplacian matrix of graph
//    p voltage potentials computed by L*p=b
///////////////////////////////////////
function y = getCurrent(i,j,L,p)
    // u*c =i
    y = -(p(i)-p(j))*L(i,j);
endfunction
```

B. Complexity proof: Matlab

The configuration used for the Matlab computations was as following:

- 32 bit dual core CPU, 1.8 GHz each
- 2 GB of RAM
- Matlab version 2011*b* (7.13.0.564) together with the Matlab *Symbolic Math Toolbox*

Solving the biggest system of linear equations (38 nodes) took about 5 minutes. All other solving and simplifying steps took less than 5 minutes each.

The code to reproduce the results is listed here. The code is similar to the one written in Scialb, listed in the previous section. The symbolic transformation procedure for the general case is listed next.

```matlab
function [ L b] = canonical3SATtoTNEPSymb(k, n0, n1, n2)
%CANONIAL3SATTOTNEP creates symbolic Laplacian matrix for 3SAT to TNEP.
% Part of the NP–hard proof of TNEP. Creates standardized
% 62x62 Lacplacian matrix with merged var–blocks and clause–blocks.
% input: symbolic variable names
%   k – number of variables
%   n0, n1, n2 – number of variables of each type (unenhanced,
%     one literal enhanced or both literals enhanced. It holds: k=n0+n1+n2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

highCond = highConductance(k); % high conductance part in variable–block
% number of connections from each literal to the clauses (before merging
% clause types):

    function [ cl ] = totalClausesOfType( t0,t1p,t1m,t2 )
        %TOTALCLAUSESOFTYPE returns symbolic number of clauses of type t
        % type t uniquely defined by t=( t0,t1p,t1m,t2 ) with
        % t0 – number of edges (from clause t) to var–block of type 0
        % t1p – number of edges to unenhanced part of var–block type 1
        % t1m – number of edges to enhanced part of var–block type 1
        % t2 – number of edges to fully enhanced var–block
        cl = 1;
        next = nChooseKSymb( n0, t0  );     cl = cl*next;
        next = nChooseKSymb( n1, t1p );     cl = cl*next;
        next = nChooseKSymb( n1–t1p, t1m ); cl = cl*next;
        next = nChooseKSymb( n2, t2 );      cl = cl*next;
        cl = 2^(t0+t2)*cl;
    end

numClauseTypes = 20;
totalSize = 1 + 2 + 3*6 + 1 + numClauseTypes;

L = sym( zeros(totalSize,totalSize), 'r' );
b = sym( zeros(totalSize,1), 'r' );

totalClauses = numTotalClauses(k);

if( abbrSymbols == 1 )
    totalClauses = tc;
end

% (1) create super–source, var–source and var–sink
% 1.1 super–source
b(1) = totalClauses;
L(1,1) = 2*k;

% 1.1.1 connect to var–blocks
varNode = 1+2+2;
L(1,varNode) = –n0;    L(varNode,1) = L(1,varNode);
L(1,varNode+2) = –n0;    L(varNode+2,1) = L(1,varNode+2);
varNode = varNode +6;
L(1,varNode) = –n1;    L(varNode,1) = L(1,varNode);
L(1,varNode+2) = –n1;    L(varNode+2,1) = L(1,varNode+2);
varNode = varNode +6;
L(1,varNode) = –n2;    L(varNode,1) = L(1,varNode);
L(1,varNode+2) = –n2;    L(varNode+2,1) = L(1,varNode+2);

% 1.2 var–source
b(2) = 1;
L(2,2) = k;

% 1.2.1 connect to var–blocks
varNode = 1+2+1;
```

```
L(2,varNode) = -n0;    L(varNode,2) = L(2,varNode);
varNode = varNode +6;
L(2,varNode) = -n1;    L(varNode,2) = L(2,varNode);
varNode = varNode +6;
L(2,varNode) = -n2;    L(varNode,2) = L(2,varNode);

% 1.3 var-sink
b(3) = -1;
L(3,3) = k;

% 1.3.1 connect to var blocks
varNode = 1+2+6;
L(3,varNode) = -n0;    L(varNode,3) = L(3,varNode);
varNode = varNode +6;
L(3,varNode) = -n1;    L(varNode,3) = L(3,varNode);
varNode = varNode +6;
L(3,varNode) = -n2;    L(varNode,3) = L(3,varNode);

% (2) create var blocks (with final diagonal)
% 2.1 type 0
varBlock = varBlockSymb( 0, n0, k );
L(4:9, 4:9 ) = varBlock(:,:);

% 2.2 type 1
varBlock = varBlockSymb( 1, n1, k );
L(10:15, 10:15 ) = varBlock(:,:);

% 2.3 type 2
varBlock = varBlockSymb( 2, n2, k );
L(16:21, 16:21 ) = varBlock(:,:);

% (3) interconnect-block
L(22,22) = 2*k*highCond + totalClauses/2;

% 3.1 connect to var-blocks
varNode = 1+2+2;
L(22,varNode) = -n0*highCond;   L(varNode,22) = L(22,varNode);
L(22,varNode+2) = -n0*highCond; L(varNode+2,22) = L(22,varNode+2);

varNode = varNode+6;
L(22,varNode) = -n1*highCond;   L(varNode,22) = L(22,varNode);
L(22,varNode+2) = -n1*highCond; L(varNode+2,22) = L(22,varNode+2);

varNode = varNode+6;
L(22,varNode) = -n2*highCond;   L(varNode,22) = L(22,varNode);
L(22,varNode+2) = -n2*highCond; L(varNode+2,22) = L(22,varNode+2);

% (4) create and connect all clause-blocks
nodeOffset = 22;
clauseNumber =1;
for i1=0:3
    for i2=i1:3
        for i3=i2:3
            % count occurrences of types
            t0  = (i1==0)+(i2==0)+(i3==0);
            t1p = (i1==1)+(i2==1)+(i3==1);
            t1m = (i1==2)+(i2==2)+(i3==2);
            t2  = (i1==3)+(i2==3)+(i3==3);

            % type is now uniquely defined by vector (t0,t1p,t1m,t2) with sum
                ==3
            % calculate total number of clauses of this type
            clausesOfType = totalClausesOfType( t0,t1p,t1m,t2 );
            % tmp
            % cl = 't'+string(clauseNumber);
            if( abbrSymbols == 1 )
                clausesOfType = t(clauseNumber);
                clauseNumber = clauseNumber +1;
            end

            % supply
            %b(nodeOffset+2) = -clausesOfType;
            b(nodeOffset+1) = -clausesOfType;

            % 4.1 create clause-block
            L(nodeOffset+1,nodeOffset+1) = 7*clausesOfType/2;

            % 4.2 connect to interconnect node and internally
            L(nodeOffset+1,22) = -clausesOfType/2;
            L(22,nodeOffset+1) = -clausesOfType/2;

            % 4.3 connnect to respective var-blocks
            % 4.3.1 to var-block of type 0
            L(6,nodeOffset+1) = -t0*clausesOfType/2;
            L(nodeOffset+1,6) = L(6,nodeOffset+1);
            L(nodeOffset+1,8) = L(6,nodeOffset+1);
            L(8,nodeOffset+1) = L(6,nodeOffset+1);

            % 4.3.2 to var-block of type 1 (negated, then unnegated)
            L(12,nodeOffset+1) = -t1m*clausesOfType;
            L(nodeOffset+1,12) = L(12,nodeOffset+1);
            L(14,nodeOffset+1) = -t1p*clausesOfType;
            L(nodeOffset+1,14) = L(14,nodeOffset+1);

            % 4.3.3 to var-block of type 2
            L(18,nodeOffset+1) = -t2*clausesOfType/2;
            L(nodeOffset+1,18) = L(18,nodeOffset+1);
            L(nodeOffset+1,20) = L(18,nodeOffset+1);
            L(20,nodeOffset+1) = L(18,nodeOffset+1);
            % next clause
            nodeOffset = nodeOffset + 1;
        end
    end
end

end
```

A few helper functions used in the previous code are listed here:

```
function [ y ] = numTotalClauses( k )
%NUMTOTALCLAUSES number of total clauses in 3-SAT scenario
% featuring k variables
    y = 4*k*(k-1)*(k-2)/3;
end

function [ y ] = numConnClauses( k )
%NUMCONNCLAUSES
% number of clauses connected to one literal
% for a k variables 3SAT scenario
    y = 2*(k-1)*(k-2);
end

function [ y ] = nChooseKSymb( n, k )
%NCHOOSEKSYMB performs operation "n choose k" with symbolic var n
% and numeric var k
    y = 1;
    if k<0
        y = 0;
    elseif k==0
        y = 1;
    else
        y = n;
        for i=1:(k-1)
            y = y*(n-i)/i;
        end
        y = y/k;
    end
end

function [ y ] = mergeNodesSymb( i,j,L )
%MERGENODESSYMB merges two nodes i,j of laplacian matrix L
% ignores possibly arising loops
% deletes row and column j and sets i as the new combined node
[n m] = size(L);
if( m ~=n || i == j || i<1 || i>n ||j<1 || j>n)
    disp('error: nodes out of bound or equal; or invalid laplacian');
    %y = zeros(1,1);
else
    % make sure i < j
    if( i > j )
        tmp = j;
        j = i;
        i = tmp;
    end
    indices = [1:(j-1) (j+1):n];
    y = sym( zeros(n-1,n-1), 'r' );
    y(:,:) = L(indices, indices);
    % merge (possible) parallel edges
    y(i,:) = L(i, indices ) + L(j, indices);
    y(:,i) = L(indices ,i) + L( indices,j);
    % set node degree
    y(i,i) = L(i,i) + L(j,j) + L(i,j) + L(j,i);
end
end

function [ y ] = lowConductance( k )
%LOWCONDUCTANCE
    y = k^(-2);
end

function [ y ] = highConductance( k )
%HIGHCONDUCTANCE
    y = k^2;
end

function L = varBlockSymb( t, scaleName, k )
        %%%%%%%%%%%%%%%%%%%/
        % create raw Variable-Block.
        % Result will have the final diagonal elements
        % (to finish it, it will only have to be interconnected to
        % supersource, var-source, clauses etc.)
        %
        % input t - type of block to create (0,1 or 2)
        %       scale - name of variable to scale
        %               (for example 'n_0' for type 0)
        %%%%%%%%%%%%%%%%%%%/
        % t out of {0,1,2}

        L = zeros(6,6);
        L = sym(L,'r');
```

```
            highCond = highConductance(k);
            lowCond = lowConductance(k);
            numConnCl = numConnClauses(k);

        % low conductance connections
        L( 1,2 ) = -lowCond;
        L( 1,4 ) = -lowCond;
        L( 3,6 ) = -lowCond;
        L( 5,6 ) = -lowCond;

        % high cond connections depending on type
        L( 2,3 ) = -highCond;
        L( 4,5 ) = -highCond;

        enhancedCond = 2*highCond;
        %enhancedCondNeg = mulf(2*highCond, '-1');

        % type 0: no enhanced connections
        if( t == 0 )
            % ... do nothing
        elseif t == 1
            % type 1: one enhanced connection
            L(2,3) = -enhancedCond;
        elseif t == 2
            % type 2: both connections enhanced
            L(2,3) = -enhancedCond;
            L(4,5) = -enhancedCond;
        end

        L = L + L.';

        % diagonal elements (final)
        L(1,1) = 2*lowCond+1;

        L(2,2) = (2+(t>=1))*highCond + lowCond+1;
        L(3,3) = (1+(t>=1))*highCond + lowCond + numConnCl;

        L(4,4) = (2+(t>=2))*highCond + lowCond+1;
        L(5,5) = (1+(t>=2))*highCond + lowCond + numConnCl;
        L(6,6) = 2*lowCond+1;

        % scale all matrix elements
        L = scaleName*L;

    end
```

The simple case when only SAT-consistent variables are considered is listed next:

```
function [ L b] = simple3SATtoTNEPSymb(k)
%SIMPLE3SATTOTNEP creates symbolic Laplacian matrix for 3SAT to TNEP.
% Part of the NP-hard proof of TNEP. Creates reduced
% 18x18 Lacplacian matrix with single big merged var-block and 4 clause-blocks.
%
% input: symbolic variable names
%   k - number of variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

highCond = highConductance(k); % high conductance part in variable-block

    function [ cl ] = totalClausesOfType( t1p,t1m )
        %TOTALCLAUSESOFTYPE returns symbolic number of clauses of type t
        % t1p - number of edges to unenhanced part of var-block type 1
        % t1m - number of edges to enhanced part of var-block type 1
        cl = 1;
        next = nChooseKSymb( k, t1p );     cl = cl*next;
        next = nChooseKSymb( k-t1p, t1m ); cl = cl*next;
    end

numClauseTypes = 4;
totalSize = 1 + 2 + 1*6 + 1 + 2*numClauseTypes;

L = sym( zeros(totalSize,totalSize), 'r' );
b = sym( zeros(totalSize,1), 'r' );

totalClauses = numTotalClauses(k);

% (1) create super-source, var-source and var-sink
% 1.1 super-source
b(1) = totalClauses;
L(1,1) = 2*k;

% 1.1.1 connect to var-block
varNode = 1+2+2;
L(1,varNode) = -k;    L(varNode,1) = L(1,varNode);
L(1,varNode+2) = -k;    L(varNode+2,1) = L(1,varNode+2);

% 1.2 var-source
b(2) = 1;
L(2,2) = k;

% 1.2.1 connect to var-block
varNode = 1+2+1;
L(2,varNode) = -k;    L(varNode,2) = L(2,varNode);

% 1.3 var-sink
b(3) = -1;
L(3,3) = k;

% 1.3.1 connect to var block
varNode = 1+2+6;
L(3,varNode) = -k;    L(varNode,3) = L(3,varNode);

% (2) create var block (with final diagonal)
% 2.1 type 1 (only type)
varBlock = varBlockSymb( 1, k, k );
L(4:9, 4:9 ) = varBlock(:,:);

% (3) interconnect-block
L(10,10) = 2*k*highCond + totalClauses;

% 3.1 connect to var-block
varNode = 1+2+2;
L(10,varNode) = -k*highCond;   L(varNode,10) = L(10,varNode);
L(10,varNode+2) = -k*highCond; L(varNode+2,10) = L(10,varNode+2);

% (4) create and connect all clause-blocks
nodeOffset = 10;

for t1p=0:3
    t1m = 3-t1p;

            % type is now uniquely defined by vector (t0,t1p,t1m,t2) with sum
                ==3
            % calculate total number of clauses of this type
            clausesOfType = totalClausesOfType( t1p,t1m );

            % supply
            b(nodeOffset+2) = -clausesOfType;

            % 4.1 create clause-block
            L(nodeOffset+1,nodeOffset+1) = 2*clausesOfType;
            L(nodeOffset+2,nodeOffset+2) = 4*clausesOfType;

            % 4.2 connect to interconnect node and internally
            L(nodeOffset+1,10) = -clausesOfType;
            L(10,nodeOffset+1) = -clausesOfType;
            L(nodeOffset+1,nodeOffset+2) = -clausesOfType;
            L(nodeOffset+2,nodeOffset+1) = -clausesOfType;

            % 4.3 connnect to var-block
            % 4.3.2 to var-block of type 1 (negated, then unnegated)
            L(6,nodeOffset+2) = -t1m*clausesOfType;
            L(nodeOffset+2,6) = L(6,nodeOffset+2);
            L(8,nodeOffset+2) = -t1p*clausesOfType;
            L(nodeOffset+2,8) = L(8,nodeOffset+2);

            % next clause
            nodeOffset = nodeOffset + 2;
end
end
```

The following script can be run to recover the results for the general case:

```
% Main script to perform NP-hard proof
disp('(1) Creating symbolic variables.');
syms k n0 n1 n2;

disp('(2) Creating symbolic Laplacian matrix and supply vector.');
[ L b ] = canonical3SATtoTNEPSymb(k, n0, n1, n2);

disp('(3) Simplifying (first time). ');
L = simplify(L);
b = simplify(b);

disp('(4) Merging variable-blocks.');
L = mergeNodesSymb(15+3, 15+5, L );
L = mergeNodesSymb(15+2, 15+4, L );
% delete corresponding entries from b
b = b([1:19 21:end]);
b = b([1:18 20:end]);

L = mergeNodesSymb(3+3, 3+5, L );
L = mergeNodesSymb(3+2, 3+4, L );
% delete corresponding entries from b
b = b([1:7 9:end]);
b = b([1:6 8:end]);

disp('(5) substituting n2=k-n0-n1.');
L = subs(L,n2,k-n0-n1);
b = subs(b,n2,k-n0-n1);

disp('(6) simplifying (second time).');
L = simplify(L);
b = simplify(b);

disp('(7) solve for var-block currents (set p(var-sink)=0). This may take a while. Please wait.');

x = L(:, [1 2 4:end])\b;
disp('simplifying necessary results.');
res = sym( zeros(3,1), 'r' );
```

```
res(1) = simplify(x(7-1));
res(2) = simplify(x(13-1));
res(3) = simplify(x(17-1));

disp('computing differences.');
delta = sym( zeros(2,1), 'r' );
delta(1) = simplify(res(2)-res(1));
delta(2) = simplify(res(3)-res(2));

disp('done');
```

The following script can be run to recover the results for the simple case (only SAT-consistent instances):

```
syms k;

disp('Generating Laplacian matrix');
[ L b] = simple3SATtoTNEPSymb(k);
disp('Simplifying');
L = simplify(L);
b = simplify(b);

i = 3;

disp('Setting node 3 as 0 potential.');

L0 = L(:,[1:(i-1) (i+1):end]);
b0 = b;
disp('Solving equation system');
p0 = L0\b0;

p = simplify([ p0(1:(i-1)).' 0  p0(i:end).' ].');
```

C. GAMS model for TNEP

The GAMS model used for the tests is listed here. It was written using GAMS IDE from [gam], Build 23.7.3 WIN 27723.27726 VS8 x86/MS together with the default free (demo) license. The parameter and variable names follow the notation in [RMGH02] rather than the one used in this work.

```
SETS
         i nodes,
         b branches;

Alias (i,j);

* include first data file
$include ib.gms

SET      bmap(b,i,j) branch to nodes incidence matrix
;


PARAMETERS
         load(i)         demand of each node
         genLvl(i)       initial supply of each node
         susc(b)         susceptance per edge in branch k
         nMin(b)         minimum (initial) number of edges for branch b
         nMax(b)         maximum number of edges for branch b
         capacity(b)     capacity for each edge of branch b
```

```
          costs(b)         cost per edge added
          pScale           voltage scale /100/
;

*second data file
$include parameters.gms

VARIABLES
          z        total cost
          n(b)     total number of edges in branch b (most important variable)
          p(i)     voltage angle at node i (determines flows)
          f(b)     total flow on branch b (aux. var)
;
POSITIVE VARIABLE p;
INTEGER VARIABLE n;
          n.lo(b) = nMin(b);
          n.up(b) = nMax(b);
          f.lo(b) = -1000;
          f.up(b) = 1000;
          p.up(i) = 1000;

EQUATIONS
          cost             added edges times cost per edge
          kpl(b,i,j)       Kirchhoff's potential law (c*u=i)
          kcl(i)           Kirchhoff's current law (outflow + load = inflow +
              supply)
          flowMin(b,i,j)   minimum flow
          flowMax(b,i,j)   maximum flow
;
          cost ..          z =e= sum(b, (n(b)-nMin(b))*costs(b));

          kpl(b,i,j)$bmap(b,i,j) ..  f(b) =e= n(b)*susc(b)*(p(i)-p(j))*pScale ;
* assume fixed generation for each node with sum(supply) = sum(demand)
* in this model without redispatch
          kcl(i).. sum((b,j)$bmap(b,i,j), f(b)) + load(i)
                   =e= sum((b,j)$bmap(b,j,i), f(b)) + genLvl(i);
          flowMin(b,i,j)$bmap(b,i,j)..  f(b) =g= -n(b)*capacity(b);
          flowMax(b,i,j)$bmap(b,i,j)..  f(b) =l= n(b)*capacity(b);

          MODEL tnep /ALL/ ;
          OPTION minlp = baron;
* include option file
        tnep.optfile = 1;
* set runtime limit in seconds (max 8h)
        tnep.reslim = 18000;
          SOLVE tnep USING minlp MINIMIZING z ;
```

As an example, the input files used for the Garver system are as following

```
SET i /1*6/;
SET b /1*15/;
```

```
SET bmap(b,i,j) branch to nodes incidence matrix
/  1 .1 .2, 2 .1 .4, 3 .1 .5, 4 .2 .3, 5 .2 .4,
 6 .3 .5, 7 .1 .3, 8 .1 .6, 9 .2 .5, 10 .2 .6,
 11 .3 .4, 12 .3 .6, 13 .4 .5, 14 .4 .6, 15 .5 .6/
 ;
 PARAMETERS
susc(b) susceptance per edge
/ 1 0.5000000000, 2 0.3333333333, 3 1.0000000000, 4 1.0000000000, 5
    0.5000000000, 6 1.0000000000, 7 0.5263157895, 8 0.2941176471, 9
    0.6451612903, 10 0.6666666667, 11 0.3389830508, 12 0.4166666667, 13
    0.3174603175, 14 0.6666666667, 15 0.3278688525
```

```
/
costs(b) cost per edge added
/ 1 40.00, 2 60.00, 3 20.00, 4 20.00, 5 40.00,
 6 20.00, 7 38.00, 8 68.00, 9 31.00, 10 30.00,
 11 59.00, 12 48.00, 13 63.00, 14 30.00, 15 61.00
/
capacity(b) capacity for each edge
/ 1 100.0000000000, 2 80.0000000000, 3 100.0000000000, 4 100.0000000000, 5
    100.0000000000, 6 100.0000000000, 7 100.0000000000, 8 70.0000000000, 9
    100.0000000000, 10 100.0000000000, 11 82.0000000000, 12 100.0000000000, 13
    75.0000000000, 14 100.0000000000, 15 78.0000000000
/
nMin(b) minimum (initial) number of edges for branch b
/ 1 1, 2 1, 3 1, 4 1, 5 1,
 6 1, 7 0, 8 0, 9 0, 10 0,
 11 0, 12 0, 13 0, 14 0, 15 0
/
nMax(b) maximum number of edges for branch b
/ 1 4, 2 4, 3 4, 4 4, 5 4,
 6 4, 7 3, 8 3, 9 3, 10 3,
 11 3, 12 3, 13 3, 14 3, 15 3
/
genLvl(i) / 1 50.0000000000, 2 0.0000000000, 3 165.0000000000, 4 0.0000000000,
    5 0.0000000000, 6 545.0000000000/
load(i) / 1 80.0000000000, 2 240.0000000000, 3 40.0000000000, 4 160.0000000000,
     5 240.0000000000, 6 0.0000000000/
;
```

For use with NEOS, the input files need to be merged into the model or put into a GDX file (see GAMS documentation).

A typical BARON option file that was used is listed here:

```
n.prior = 10
f.prior = 1
p.prior = 1
EpsR = 1e-2
```