Newton Algorithms for Large-Scale Strictly Convex Separable Network Optimization

Aleksandar Donev* Phillip Duxbury

January 2001

Abstract

In this work we summarize the basic elements of primal and dual Newton algorithms for network optimization with continuously differentiable (strictly) convex arc cost functions. Both the basic mathematics and implementation are discussed, and hints to important tuning details are made. The exposition assumes that the reader posseses a significant level of prior knowledge in the field. The algorithms have been drawn from a very large pool of literature spanning over 20 years of research in the area.

Contents

*E-mail *donev@pa.msu.edu*

1 Problem Formulation

We consider solving a *strictly convex separable min-cost network problem*, which consists of minimizing a scalar fully-separable convex cost function of the flow $\mathbf{x}$ through the arcs[1] $e_{(i,j)} \in E$ of the simple connected directed graph $G = (E,V)$, $f(\mathbf{x})$, subject to flow conservation at the nodes $i \in V$,

$$\left\{ \begin{array}{c} \min f(\mathbf{x}), \text{ where } f(\mathbf{x}) = \sum_{e_{(i,j)}} f_{(i,j)}(x_{(i,j)}) \\ \text{subject to } \mathbf{Ax} = \mathbf{b} \end{array} \right\} \tag{1}$$

where $f_{(i,j)}$ is the *cost-flow function* for arc $e_{(i,j)}$ and is strictly convex and at least twice continuously differentiable[2] on $[-\infty, \infty]$, $\mathbf{A}$ is the *node-arc incidence matrix* of G, and $\mathbf{b}$ is the *nodal supply-demand vector*[3].

Of interest to physics are separable problems which are just convex[4], which will be discussed in this formulation because of their relation to ill-conditioned strictly convex problems. Also of interest are problems in which the feasible range of flows is limited, in particular, there is a capacity constraint on the flows, $l \leq x \leq u$, which can be incorporated into the formulation of problem 1 by setting f to $+\infty$ for all infeasible flows. We will not discuss this case, and recommend reference [14] for logarithmic barrier methods aimed at this problem. *All algorithms will be geared toward very sparse networks*[5], that is, the number of arcs, m, is assumed to be of the same order as the number of nodes in the network, $m = O(n)$.

[1] We will call these arcs instead of edges and nodes instead of vertices.
[2] This means that $f''(x) > \mathbf{0}$ for all x.
[3] That is, b_i is the excess in(out)flow at node i.
[4] That is, $f''(x) \geq \mathbf{0}$ for all x.
[5] For example, networks with nodal degrees bounded from above by $d = O(1)$, such as d-dimensional randomly diluted grids.

Vectors and matrices will be denoted by bold letters in this write-up, and vectors will be either of length m, called *arc vectors*, or of length n, called *nodal vectors*. Most vectors will be denoted with lowercase letters, while matrices will be uppercase. We tried to follow the original notation in most cases, which will often cause notation conflicts, but we hope the context will enable the reader to understand the intended meaning. Particular difficulties arise because of the notion of directedness of the arcs. In physics applications of interest to us, arcs are not directed. In a strict sense, directed arcs do not support negative flows in most network formulations, but we allow negative flows to avoid replacing each arc between nodes i and j with two directed arcs[6], $e_{(i,j)}$ and $e_{(j,i)}$. We call an undirected arc between i and j a *link* and denote it with $\{i,j\}$, and we assume each arc has a chosen natural direction (i,j) that determines the positive direction of the flow.

1.1 Important Considerations

There are several properties of problem 1 that should be kept in mind while reading this summary of methods to solve it. First and most important, the strict convexity of the cost functions makes the task easy, in the sense that *problem* 1 *has a unique solution*, which is a unique local and global constrained minimum of $f(\mathbf{x})$. We will see that the strict convexity makes it easy to derive globally convergent algorithms for solving 1.

However, the problems of practical interest, such as the one discussed next, are usually very large-scale, so that what we really want are *CPU– and especially memory–efficient algorithms*[7] for solving 1. We will discuss the algorithms at a certain high level and any pseudocodes will be serial codes, however, it should be stressed that the algorithms presented here were chosen because of their potential for *parallelization* on both shared and distributed memory architectures. Implementation details will be discussed in a later report.

Finally, we wish to stress that although the primary motivation for this work is the importance of 1 to computational materials science, the problem is very interesting from an applied/computational mathematics point of view, since it is a synthesis of *continuous* and *combinatorial* optimization. The continuous aspects stem from the convex functions $f_{(i,j)}$, while the discrete aspects stem from the special graph structure of the constraint matrix $\mathbf{A}$. We will explain how advances in both fields can be used to create a robust and efficient algorithm.

It should be mentioned that the problem as formulated in more detail in the sections to come bears great resemblance to Finite Element formulations for non-elliptic PDE's, and also to the *linear min-cost network flow problem*[8], so that there is significant potential for sharing experiences with other very active areas of research.

[6] In this case G is no longer a simple graph.

[7] All algorithms presented here are $O(n)$ in storage and each step of the algorithms is at most $O(n \log n)$ in time complexity, albeit usually just $O(n)$.

[8] Such as max-flow or shortest-path problems.

1.2 Case Study: Non-Linear Resistive Networks

The object of focus throughout the paper will be non-linear networks of the type appearing frequently in computational materials science. These networks are usually very large and very sparse (nearest-neighbor lattices, for example). In these applications the cost functions are usually symmetric $f_{(i,j)}(-x) = f_{(i,j)}(x)$ and smooth, with simple asymptotic behavior, but with a characteristic highly non-linear transition region. Two particular cases include networks where the arcs are made from a superconductor-like or a dielectric-like material, which are both kinds of non-linear resistive materials. These are discussed in detail in Appendices B_1 and B_2. A special case also of interest is a network of linear resistive elements, discussed later on.

Consider a network of non-linear resistive elements, such as superconducting sticks. Following physics principles, we know that in this case the solution of problem 1 corresponds to finding the electric *potentials* (voltages) of each node U_i, given the current supply-demand at each node b_i and possibly pre-specified potentials at certain nodes. Given the nodal vector $\mathbf{U}$, we can find the potential drop across an arc $e_{(i,j)}$, $V_{(i,j)} = U_i - U_j$, in the form of an arc vector $\mathbf{V}$. The physics of the problem is contained in the functional relation between this voltage drop and the current through the arc $I_{(i,j)}$—the voltage-current characteristic curve, from now on called the *potential-flow arc characteristic curve*, $V_{(i,j)} = V_{(i,j)}\left(I_{(i,j)}\right)$.

For simplicity, in certain sections we drop the indexing to write $V = V(I)$, where it is assumed that certain parameters in $V(I)$ may depend on the particular arc $e_{(i,j)}$ one is referring to. For such a resistive arc, the cost-function is a power-like integral, $f(x) = \int_{I=0}^{x} V(I)dI$. Therefore, $f^{'}(x) = V(x)$. The strict convexity requirement on $f(x)$, $f^{''}(x) > 0$ for all x, now translates to $V'(I) > 0$ for all I, which means that the voltage must be an increasing function of the current. This is true in almost all physics-related problems.

2 The Dual Approach

The dual approach to solving the convex linearly constrained problem is a very attractive and well-studied approach based on the method of Lagrange multipliers. First we formulate the dual problem for a general separable strictly convex objective function subject to linear constraints,

$$\left\{ \begin{array}{c} \min f(\mathbf{x}), \text{ where } f(\mathbf{x}) = \sum_{i=1}^{n} f_i(x_i) \\ \text{subject to } \mathbf{Ax} = \mathbf{b} \end{array} \right\} \tag{2}$$

and then focus on the special case of the network optimization problem.

2.1 The Dual Problem

This section draws mostly from the summaries and work in references [10], [11] and [7] and references given therein.

We adopt a dual Lagrangian approach to solving 2. We form the Lagrangian function $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$ and the dual equivalent problem:

$$\left\{ \begin{array}{c} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) \\ \max_{\boldsymbol{\lambda}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \end{array} \right\} \tag{3}$$

where $\boldsymbol{\lambda}$ is a vector of Lagrange multipliers. It can be shown that in the special case of linear constraints and strictly convex objective function there is no duality gap between the primal and dual problem, i.e. we could solve either one to obtain the same answer. This global constraint minimum satisfies the Kuhn-Tucker optimality conditions, which in this simple case essentially state that the solution pair $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ must be an extremum of the Lagrangian:

$$\boldsymbol{\nabla}_{\mathbf{x}} \mathcal{L}(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \boldsymbol{\nabla}_{\boldsymbol{\lambda}} \mathcal{L}(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \mathbf{0} \tag{4}$$

Now we do the following transformations on the dual problem, using the linearity and separability of the primal problem:

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) &= \max_{\boldsymbol{\lambda}} \min_{\mathbf{x}} \left[f(\mathbf{x}) - \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) \right] = \\ -\min_{\boldsymbol{\lambda}} \left\{ \max_{\mathbf{x}} \left[\mathbf{x}^T \left(\mathbf{A}^T \boldsymbol{\lambda} \right) - f(\mathbf{x}) \right] - \boldsymbol{\lambda}^T \mathbf{b} \right\} &= -\min_{\boldsymbol{\lambda}} \left[g(\boldsymbol{\lambda}) - \boldsymbol{\lambda}^T \mathbf{b} \right] = -\min_{\boldsymbol{\lambda}} \mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) \end{aligned}$$

where now $\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = g(\boldsymbol{\lambda}) - \boldsymbol{\lambda}^T \mathbf{b}$ is the so-called *dual function*[9]. It is a fortunate characteristic of separable convex problems that this function exists and can be found easily and has some very nice properties:

$$g(\boldsymbol{\lambda}) = \max_{\mathbf{x}} \left[\mathbf{x}^T \left(\mathbf{A}^T \boldsymbol{\lambda} \right) - f(\mathbf{x}) \right] = \sum_i \max_{x_i} \left[x_i t_i - f_i(x_i) \right] = \sum_i g_i(t_i)$$

where now $\mathbf{t} = \mathbf{A}^T \boldsymbol{\lambda}$ and $t_i = \lambda_{\text{from}} - \lambda_{\text{to}}$ is the *tension of arc* i, and λ_{from} and λ_{to} are the Lagrangian costs of the node from which the arc originates and the node in which it terminates. The dual function is thus also separable, and each of its components can be found through a one dimensional extremization $g_i(t_i) = \max_{x_i} \left[x_i t_i - f_i(x_i) \right] = f^*(x_i)$. We look at this maximization more closely next.

2.2 Convex-Conjugate Functions

In this section we drop the subscript i from the previous section, and define the *convex-conjugate function* $f^*(x)$ *of a convex function of one variable* $f(x)$:

$$f^*(t) = \max_x \left[xt - f(x) \right] \tag{5}$$

[9] Actually, in the literature the part $\boldsymbol{\lambda}^T \mathbf{b}$ is also included in $g(\boldsymbol{\lambda})$; we omit it at this point to simplify writing the remainder of this document.

Owing to the strict convexity of the function f, there is a unique such maximum which can be found by setting the derivative of the minimizing argument to zero[10]. Let the value that reaches this optimum value be x^*:

$$t - f'(x^*) = 0, \text{ and thus } x^* = \left(f'\right)^{-1}(t), \text{ and } f^*(t) = x^* t - f(x^*) \tag{6}$$

For strictly convex cases, it turns out that the convex-conjugate function $f^*(t)$ is also convex and differentiable in an almost trivial way:

$$\left(f^*\right)'(t) = x^* \tag{7}$$

Also, the function f^* is twice differentiable (with a positive second derivative):

$$\left(f^*\right)''(t) = \frac{1}{f''(x^*)} \tag{8}$$

provided that the second derivative of the function f exists and is non-zero at the uniquely defined x^*.

Equation 6 for x^* is in general a univariate non-linear equation and may be prohibitively expensive to solve numerically. Some special non-quadratic cases are discussed in Appendix B; we discuss quadratic cost functions $f(x)$ in what follows.

2.3 Derivatives of the Dual

To use standard techniques for differentiable unconstrained optimization, we need the derivatives of $\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda})$. The dual function $\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda})$ is convex and differentiable, and in certain cases twice differentiable. Therefore the dual problem, $\min_{\boldsymbol{\lambda}} \mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda})$, becomes *convex differentiable unconstrained optimization* which is well known to be an easy problem for which a high-accuracy solution can be found quickly and efficiently. There are many unconstrained optimization tools that one can use, and all require evaluating the gradient:

$$\mathbf{g} = \boldsymbol{\nabla}\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = \mathbf{A}\mathbf{x}^* - \mathbf{b}, \text{ where } x_i^* = \left(f_i'\right)^{-1}(t) \text{ and } \mathbf{t} = \mathbf{A}^T\boldsymbol{\lambda} \tag{9}$$

Many techniques, and notably Newton methods, discussed at length later on, also need a positive-definite approximation for the Hessian[11], which can be:

$$\mathbf{H} = \boldsymbol{\nabla}^2\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = \mathbf{A}\mathbf{H}^*\mathbf{A}^T \tag{10}$$

where $\mathbf{H}^* \approx \boldsymbol{\nabla}^2 f^*$ is the diagonal conjugate Hessian,

[10] We assume these derivatives exist, although this is by no means neccessary (see reference [7]).

[11] It should be evident that this Hessian is indeed positive-definite because $\mathbf{x}^T\mathbf{H}\mathbf{x} = \left(\mathbf{A}^T\mathbf{x}\right)^T \mathbf{H}^* \left(\mathbf{A}^T\mathbf{x}\right) > \mathbf{0}$ so long as $\mathbf{H}^*$ is positive-definite.

$$\mathbf{H}^* = \operatorname{diag}\left\{ f^{*''}(t_i) \right\} = \operatorname{diag}\left\{ \frac{1}{f''(x_i^*)} \right\} \tag{11}$$

An important thing to notice about the above equations is that they only require the evaluation of $(f_i')^{-1}(t)$ and $f_i''(x)$, and possibly $f_i'(x)$, for given t and x.

2.4 Quadratic Networks

The full dual formulation for quadratic networks with bounds on the flows[12] is given in reference [18], and also discussed in a different setting in [12].

The case when the objective function is quadratic is of special interest, because of the Sequential Quadratic Programming methods discussed later on. In this case all of the above manipulations can be done exactly analytically and easily implemented numerically. Then, we have that $f(x) = \frac{\alpha}{2}x^2 + \beta x$ and so equation 6 becomes:

$$t = \alpha x^* + \beta, \text{ so that } x^* = \frac{t-\beta}{\alpha} \tag{12}$$

and from here all the derivatives of the conjugate-function can easily be derived:

$$(f^*)'(t) = \frac{t-\beta}{\alpha}, \text{ and } (f^*)''(t) = \frac{1}{\alpha} \tag{13}$$

Thus we conclude that when,

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\bar{\mathbf{H}}\mathbf{x} + \bar{\mathbf{g}}^T\mathbf{x} \tag{14}$$

and $\bar{\mathbf{H}}$ is diagonal positive definite, the gradient and Hessian of the dual function, which is also convex quadratic, become,

$$\mathbf{g} = \mathbf{A}\bar{\mathbf{H}}^{-1}\left(\mathbf{A}^T\boldsymbol{\lambda} - \bar{\mathbf{g}}\right) - \mathbf{b}, \text{ and } \mathbf{H} = \mathbf{A}\bar{\mathbf{H}}^{-1}\mathbf{A}^T \tag{15}$$

since $\mathbf{x}^* = \bar{\mathbf{H}}^{-1}\left(\mathbf{A}^T\boldsymbol{\lambda} - \bar{\mathbf{g}}\right)$. We will use these in subsequent sections.

2.5 Solving the Dual Problem

We again stress that the dual problem, $\min_{\boldsymbol{\lambda}} \mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda})$ is a *convex differentiable unconstrained optimization problem* which is well known to be an easy problem for which a high-accuracy solution can be found quickly and efficiently. We also gave formulae for the gradient and Hessian of $\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda})$. The optimization process would in a very general setting follow the algorithm,

1. Choose initial guess for $\boldsymbol{\lambda}$, $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}_0$

[12] Bounds make the solution significantly more complex, and are usually a neccessary part of the formulation.

2. *Major iteration*: Iterate until $\|\mathbf{g}\| < \varepsilon \|\mathbf{b}\|$

   (a) Obtain a descent search direction $\Delta\boldsymbol{\lambda}$

   (b) Perform a line search along $\Delta\boldsymbol{\lambda}$, $\alpha = \arg\min_\alpha \mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}+\alpha\Delta\boldsymbol{\lambda})$

   (c) Take a step $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}+\alpha\Delta\boldsymbol{\lambda}$ and update $\mathbf{t} = \mathbf{A}^T\boldsymbol{\lambda}$ and $x_i^* = f_i^*(t_i)$

where ε is the desired relative precision. Different types of norms could of course be used in the convergence test, which can also take different forms[13]. In particular, tests for convergence of both the multipliers and the flows should probably be included. The main component of the algorithm is step 2a, where different procedures for determining the descent direction, such as non-linear conjugate gradients or Newton's method can be used. This is discussed in great detail next.

It is important to understand that at each point in this algorithm one of the Kuhn-Tucker conditions is satisfied, $\boldsymbol{\nabla}_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*,\boldsymbol{\lambda}) = \mathbf{0}$, even though the iterates for the flow $\mathbf{x}^*$ are infeasible. When the minimum of the dual is reached, $\mathbf{g} = \mathbf{A}\mathbf{x}^* - \mathbf{b} = \mathbf{0}$, and the flow becomes feasible, at which point both Kuhn-Tucker conditions are satisfied. This process gives great freedom to the implementor. In fact, the algorithm can start from any initial point, and can take an arbitrary path to the optimum. Feasibility is not required at either step of the algorithm. The other two algorithms reviewed in this work follow a feasible or semi-feasible path to the optimum and are thus more constrained in their search space. However, the above algorithm requires strict convexity[14] and strict separability of the cost functions, and also requires that the conjugate functions be derived and coded, which hinders simple plug-and-play library implementations. The other two algorithms will relax these requirements. Therefore the different algorithms should be tried and maybe even combined.

Now we discuss two algorithms for obtaining a descent search direction and also discuss the line search step in the above algorithm.

2.5.1 Non-Linear Conjugate-Gradient Algorithm

Since the gradient in 9 is easy to compute and requires only a single matrix-vector product with $\mathbf{A}$, *non-linear conjugate gradient algorithms* for determining the descent direction $\Delta\boldsymbol{\lambda}$ will be efficient. Computational experiments documented in detail in reference [11] have suggested that the *Polak-Ribiere update formula with a gradient restart procedure* is an efficient choice. These are all standard optimization tools. The main disadvantage of conjugate-gradient methods is that they have linear near-optimum convergence, so that they are not very suitable if a high accuracy for the solution is requested. In fact, it is the authors' belief that Truncated Newton techniques offer greater flexibility and hold more promise, so we will discuss these in greater detail next.

[13] There seems to be little concensus on the best termination criteria, simply because this is very problem-dependent.

[14] This will be discussed in greater detail when discussing preconditioning.

2.5.2 The Truncated Dual Newton Algorithm

If a high accuracy is wanted for the solution, then superlinearly convergent algorithms are useful. The best known example is *Newton's method*, which determines the descent direction as a solution to the Newton linear system of equations:

$$\langle \mathbf{H} \rangle \, \Delta \boldsymbol{\lambda} = -\mathbf{g} \tag{16}$$

where $\langle \mathbf{H} \rangle$ is a numerically stable positive definite approximation for the Hessian. For large-scale optimization, system 16 can be efficiently solved using a *Preconditioned Conjugate-Gradient (PCG) algorithm*[15]. This only requires multiplication of a vector with the Hessian matrix $\langle \mathbf{H} \rangle = \mathbf{A} \langle \mathbf{H}^* \rangle \mathbf{A}^T$, and possibly a preconditioner. We will assume that the reader is well-familiar with PCG and will discuss PCG in the context of 16 in network optimization later on. We refer to the PCG iterations as the *minor iterations*.

For a quadratic minimization problem, Newton's method is equivalent to the non-linear conjugate gradient algorithm discussed above, and reaches the minimum in a single step. For a general case it is usually wasteful to solve 16 exactly at each global iteration. In fact, the quadratic convergence benefits of Newton's method only become worth-while when the search reaches the nearly-quadratic basin around the optimum. An efficient modification is to use a *Truncated Conjugate-Gradient Newton algorithm (TCGN)*, in which we solve 16 only approximately, refining the accuracy of the solution as we approach the optimum, according to well established theoretical and empirical formulas found in reference [16].

In this approach, we terminate PCG while solving 16 as soon as the residual $\mathbf{r}$ becomes,

$$\|\mathbf{r}\| = \|\langle \mathbf{H} \rangle \, \Delta \boldsymbol{\lambda} + \mathbf{g}\| \approx \eta \, \|\mathbf{g}\| \tag{17}$$

where η is the *forcing sequence*. It is clear that η should be large in the early stages of the optimization and reduced close to the optimum. A general adaptive form which guarantees quadratic convergence near the minimum, but which may not be the best choice for a given problem is,

$$\eta = \min \left\{ \frac{1}{k}, \|\mathbf{g}\| \right\} \tag{18}$$

where k is the number of the current major iteration and is used to insure stability and avoid hang-ups in the algorithm, and of course the term $\frac{1}{k}$ can be replaced with other forcing sequences more suitable to a given problem. Near the optimum the gradient will become very small and so $\eta \approx \|\mathbf{g}\|$. The theory here is that convergence is superlinear if the residual in TCGN is $\|\mathbf{r}\| = o(\|\mathbf{g}\|)$, and quadratic if $\|\mathbf{r}\| = o(\|\mathbf{g}\|^2)$.

It is important to stress that a suitable choice for the initial guess for $\Delta \boldsymbol{\lambda}$ when solving the system 16 with PCG is $(\Delta \boldsymbol{\lambda})_0 = \mathbf{0}$. A simple run through

[15] The diagonal of $\langle \mathbf{H} \rangle$ can be used as a preconditioner.

PCG shows that with this choice, if PCG terminates only after 1 iteration, the result $\Delta\boldsymbol{\lambda} = -\frac{\mathbf{g}^T\mathbf{g}}{\mathbf{g}^T\langle\mathbf{H}\rangle\mathbf{g}}\mathbf{g}$ is returned, which is a steepest descent direction[16]. If preconditioning with $\mathbf{M}^{-1}$ is used, $\mathbf{g}$ should be replaced with $\mathbf{M}^{-1}\mathbf{g}$. In this sense, TCGN will act as a bridge between gradient descent, which is known to be about the best universal choice far from an optimum, and Newton methods, which is known to be optimal near the minimum, thus imitating the fast descent of the former far from the optimum, and approximating the superlinear convergence of the later near the optimum.

2.5.3 Line Search

In the general non-linear case both the conjugate gradient and Newton algorithms require some form of a line search. The former requires a higher accuracy in the search than the later because the algorithm is based on the fact that a minimum has been reached along the previous search direction. In fact, the line search in Newton's algorithm can be avoided by using some elegant step-size heuristics. It is premature to discuss the best line search method at this time, although this is an important decision. The line search that needs to be performed at each major iteration is:

$$\min_{\alpha} \mathcal{L}_{\boldsymbol{\lambda}}\left(\boldsymbol{\lambda} + \alpha\Delta\boldsymbol{\lambda}\right) \tag{19}$$

Here, $\mathcal{L}_{\boldsymbol{\lambda}}\left(\boldsymbol{\lambda} + \alpha\Delta\boldsymbol{\lambda}\right)$ is a continuously differentiable strictly convex function, so that we can solve 19 uniquely by simply setting the derivative to zero, to get:

$$\frac{\partial\mathcal{L}_{\boldsymbol{\lambda}}\left(\boldsymbol{\lambda} + \alpha\Delta\boldsymbol{\lambda}\right)}{\partial\alpha} = \left(\left[\mathbf{g}\right]_{\boldsymbol{\lambda}+\alpha\Delta\boldsymbol{\lambda}}\right)^T\Delta\boldsymbol{\lambda} = \left(\mathbf{A}\left[\mathbf{x}^*\right]_{\boldsymbol{\lambda}+\alpha\Delta\boldsymbol{\lambda}} - \mathbf{b}\right)^T\Delta\boldsymbol{\lambda} = h(\alpha) = 0 \tag{20}$$

It can be shown that the function $h(\alpha)$ is continuous nonincreasing, so that it has a unique and fairly easy to find zero. In fact, in certain special occasions, it may be possible to solve 20 analytically (such as for quadratic optimization with box bounds on $\mathbf{x}$, as in [18]). The derivative,

$$\frac{\partial h(\alpha)}{\partial\alpha} = \Delta\boldsymbol{\lambda}^T\left(\left[\mathbf{H}\right]_{\boldsymbol{\lambda}+\alpha\Delta\boldsymbol{\lambda}}\right)\Delta\boldsymbol{\lambda} \approx \Delta\boldsymbol{\lambda}^T\mathbf{A}\left(\langle\mathbf{H}^*\rangle_{\boldsymbol{\lambda}+\alpha\Delta\boldsymbol{\lambda}}\right)\mathbf{A}^T\Delta\boldsymbol{\lambda} \tag{21}$$

where $\langle\mathbf{H}^*\rangle$ is again some stable positive definite approximation for the conjugate Hessian, can also be evaluated and used, for example, in Newton's algorithm for solving the non-linear equation 20. The specifics of this process do however depend on the specific type of cost functions at hand.

Appendix A contains an illustration of a step in the Dual TCGN Method. Implementation details, such as the matrix-vector product in PCG, as well as some specific preconditioning strategies tuned for network optimization will be discussed after another related approach to solving problem 2 has been described.

[16] In this case a step-size $\alpha = 1$ is such that the minimum of the quadratic model to the objective function is reached along the gradient descent direction.

2.6 Proximal Point Algorithm for Convex Programming

In this section we describe an algorithm which can be almost trivially included in the dual Newton algorithm described above and that can be used to solve (non necessarily strictly) convex programs, but that can also be used to improve numerical conditioning for ill-conditioned strictly convex programs, as discussed later on. For details, consult reference [9]. The algorithm is based on the *proximal point algorithm* for minimizing a convex function[17] $f(\mathbf{x})$ over a convex subspace $\mathcal{S}$:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}\in\mathcal{S}} f(\mathbf{x}) \tag{22}$$

It can be shown that under very mild conditions for convex f the sequence $\mathbf{x}_k$ generated by the following algorithm converges to $\hat{\mathbf{x}}$,

$$\mathbf{x}_{k+1} \leftarrow \arg\min_{\mathbf{x}\in\mathcal{S}} \left[f(\mathbf{x}) + \varepsilon_k D_g(\mathbf{x},\mathbf{x}_k)\right] = \arg\min_{\mathbf{x}\in\mathcal{S}} \tilde{f}_k(\mathbf{x}) \tag{23}$$

where $D_g(\mathbf{x},\mathbf{y})$ is a *distance function* defined via another strictly convex differentiable function $g(\mathbf{x})$,

$$D_g(\mathbf{x},\mathbf{y}) = [g(\mathbf{x}) - g(\mathbf{y})] - [\boldsymbol{\nabla} g(\mathbf{y})]^T (\mathbf{x}-\mathbf{y}) \tag{24}$$

and ε_k is a decreasing sequence of positive numbers.

In essence, the proximal point algorithm simply augments the cost function with a strictly convex distance-like proximal term $D_g(\mathbf{x},\mathbf{x}_k)$ so that each minimization subproblem $\arg\min_{\mathbf{x}\in\mathcal{S}} \tilde{f}_k(\mathbf{x})$ in 23 becomes easy to solve using the truncated Newton algorithm described above. Of course, each of the subproblems does not have to be solved exactly, and provable globally convergent truncation in this case is achieved if each of the minimization subproblems is solved up to an accuracy of,

$$\left\| \boldsymbol{\nabla} \tilde{f}_k(\mathbf{x}_{k+1}) \right\| \leq \varepsilon_k \delta_k \tag{25}$$

where δ_k is a decreasing sequence whose sum converges (see [9] and references to Rockafellar's work therein for exact expressions).

The proximal point algorithm is easily generalized to constrained convex programming, such as problem 2. One approach, the *Primal Dual Proximal Point algorithm*, described in [9], is to add proximal terms in both the primal and dual (multipliers) variables to the Lagrangian in the dual problem 3 and execute the algorithm[18]:

$$\left\{ \begin{array}{c} (\mathbf{x}_{k+1},\boldsymbol{\lambda}_{k+1}) \leftarrow \arg\max_{\boldsymbol{\lambda}} \min_{\mathbf{x}} \overset{\heartsuit}{\mathcal{L}}_k(\mathbf{x},\boldsymbol{\lambda}) \\ \text{where } \overset{\heartsuit}{\mathcal{L}}_k(\mathbf{x},\boldsymbol{\lambda}) = f(\mathbf{x}) + D_g(\mathbf{x},\mathbf{x}_k) - \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x}-\mathbf{b}) - D_g(\boldsymbol{\lambda},\boldsymbol{\lambda}_k) \end{array} \right\} \tag{26}$$

[17] The global minimum in this case may be degenerate, but every local minimum is still guaranteed to be a global one for convex cost functions.

[18] Notice the negative sign in the proximal term for the multipliers.

In this scheme, the proximal point iteration becomes an outer iteration wrapping up problem 2, which we can solve using the dual Newton algorithm described above. We will call this a *super iteration*. In particular, if we choose a simple function $g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{M}\mathbf{x}$, where $\mathbf{M}$ is a positive definite diagonal matrix[19], then the distance function becomes a simple scaled Euclidean distance,

$$D_2(\mathbf{x}, \mathbf{y}) = \frac{1}{2}(\mathbf{x} - \mathbf{y})^T \mathbf{M}(\mathbf{x} - \mathbf{y}) \tag{27}$$

so that during each super iteration the cost function in problem 2 gets changed to,

$$f_i(x_i) \leftarrow \left(\tilde{f}_k\right)_i (x_i) = f_i(x_i) + \frac{M_{ii}\varepsilon_k}{2}(x_i - \bar{x}_i)^2 \tag{28}$$

where the bar in $\bar{\mathbf{x}}$ denotes the value of $\mathbf{x}$ at the previous super iteration. In essence, the only difference is that the cost function gets an added quadratic-linear term, which needs to be included when calculating the convex-conjugate functions and their derivatives[20]. Notice that the quadratic term guarantees strict convexity. Also, the added dual proximal term introduces an additional term in the gradient and Hessian of the dual function in 9 and 10:

$$\mathbf{g} \leftarrow \bar{\mathbf{g}}_k = \mathbf{A}\mathbf{x}^* - \mathbf{b} + \varepsilon_k\mathbf{M}(\boldsymbol{\lambda} - \boldsymbol{\lambda}_k) \tag{29}$$

$$\mathbf{H} \leftarrow \bar{\mathbf{H}}_k = \mathbf{A}\bar{\mathbf{H}}^*_{\mathbf{k}}\mathbf{A}^T + \varepsilon_k\mathbf{M} \tag{30}$$

Notice that the diagonal correction $\varepsilon_k\mathbf{M}$ conditions the Hessian numerically. With these expressions all of the above algorithms apply to each super iteration without any significant changes. It is not clear how many super iterations will be required to reach convergence in this approach and what the additional incurred cost will be. We will discuss this in some more detail later.

3 A Sequential Quadratic Programming Dual Approach

Through the possibly confusing previous section we demonstrated that problem 2 can be solved by considering a dual approach in which an infeasible path is taken to the optimum. The main disadvantage of this approach is that it required evaluation of the derivatives of the conjugate convex function $f^*(\mathbf{x})$, which requires coding and extensive optimization of complicated non-linear solvers in some cases. Here we discuss one approach that only requires evaluation of the first and second order derivatives of the primal function $f(\mathbf{x})$, the *sequential quadratic programming approach*.

[19] For simplicity it is usually taken that $\mathbf{M} = \mathbf{I}$.

[20] We have maintained the D-function formulation because it may be simpler to derive these conjugate functions with another non-quadratic distance function.

3.1 Sequential Quadratic Programming (SQP)

The SQP algorithm for constrained optimization is rather simple in its nature—it finds the optimum by successively solving a *sequence of constrained quadratic problems* that are based on the local nature of the objective function. In network optimization (albeit not under this name) it was introduced in [12], but it is also commonly used in general non-linear programming, as in [17]. In this section barred quantities will denote the current estimates[21] of certain variables.

We start by making a local quadratic approximation to the objective function around the current point $\bar{\mathbf{x}}$, $\mathbf{x} = \bar{\mathbf{x}} + \Delta\mathbf{x}$ and include the constraint $\mathbf{Ax} = \mathbf{A}(\bar{\mathbf{x}} + \Delta\mathbf{x}) = \mathbf{b}$, to get the linearly constrained quadratic problem,

$$\left\{ \begin{array}{c} \min \Delta f(\Delta\mathbf{x}), \text{ where } \Delta f(\Delta\mathbf{x}) = \frac{1}{2}\Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} + \mathbf{g}^T\Delta\mathbf{x} \\ \text{subject to } \mathbf{A}\left(\Delta\mathbf{x}\right) = \mathbf{b} - \mathbf{A}\bar{\mathbf{x}} \end{array} \right\} \tag{31}$$

where now $\mathbf{H} = \boldsymbol{\nabla}^2_{\mathbf{x}} f(\bar{\mathbf{x}})$ and $\mathbf{g} = \boldsymbol{\nabla}_{\mathbf{x}} f(\bar{\mathbf{x}})$ are the Hessian and gradient of the primal objective function. It should be noted than in the general SQP method $\mathbf{H}$ is taken to be Hessian of the Lagrangian function $\mathcal{L}(\mathbf{x}, \bar{\boldsymbol{\lambda}}) = f(\mathbf{x}) - \bar{\boldsymbol{\lambda}}^T(\mathbf{Ax} - \mathbf{b})$, where $\bar{\boldsymbol{\lambda}}$ is a current estimate of the optimal Lagrange multipliers, but in this case of linear equality constraints there is no difference. Also, the gradient $\mathbf{g}$ could be taken to be the gradient of the Lagrangian $\nabla_{\mathbf{x}} f(\mathbf{x}) - \mathbf{A}^T\bar{\boldsymbol{\lambda}}$, but the only difference with the above formulation in this case is that the Lagrange multipliers of 31 should be interpreted as the change in the Lagrange estimates, whereas in our formulation the multipliers of 31 are the new estimates of the Lagrange multipliers of problem 2.

We thus arrive at a semi-feasible approach to solving problem 2:

1. Choose initial guess for $\bar{\mathbf{x}}$, $\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}}_0$, $\bar{\boldsymbol{\lambda}} \leftarrow \mathbf{0}$
2. *Major iteration*: Iterate until $\left\|\mathbf{g} - \mathbf{A}^T\bar{\boldsymbol{\lambda}}\right\| < \varepsilon$
   (a) Obtain a descent search direction $\Delta\mathbf{x}$ as an approximate solution to 31 and obtain estimate for $\bar{\boldsymbol{\lambda}}$.
   (b) Perform a line search along $\Delta\mathbf{x}$, $\alpha = \arg\min_\alpha f(\bar{\mathbf{x}} + \alpha\Delta\mathbf{x})$
   (c) Take a step $\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \alpha\Delta\mathbf{x}$

where ε is the desired precision. Of course, in reality, more sophisticated termination criteria should be used, that check for convergence in both $\bar{\mathbf{x}}$ and $\bar{\boldsymbol{\lambda}}$.

3.1.1 Dual Newton Method for Solving Quadratic Problems

The reason we discussed quadratic objective functions in the Dual Newton approach should now be evident. As shown there, the solution of problem 31 can be obtained in one step by solving the Newton system for the dual function. In

[21] Usually the values of these variables at the previous iteration.

fact, using our previous derivations, equation 15, it can easily be shown that the solution to problem 31 is the solution to the linear system:

$$\begin{aligned} \mathbf{A}\mathbf{H}^{-1}\mathbf{A}^T\boldsymbol{\lambda} &= \mathbf{A}\mathbf{H}^{-1}\mathbf{g} + \mathbf{b} - \mathbf{A}\bar{\mathbf{x}} \\ \Delta\mathbf{x} &= \mathbf{H}^{-1}\left(\mathbf{A}^T\boldsymbol{\lambda} - \mathbf{g}\right) \end{aligned} \tag{32}$$

Full separability of the objective function is not needed in this case even in the dual formulation. In fact, 32 gives the solution of 31 for any quadratic objective function.

We however prefer to still think of problem 31 as an instance of the general problem 2, but carrying in mind that only a single Newton step is needed to reach the optimum in this case, and that very simple closed formulae exist for the conjugate functions and their derivatives. This is simply because by doing so most of the code, and especially the linear solver and system assembly, can be reused for the two algorithms. An important thing to mention is that the sequential quadratic programming approach is also *guaranteed asymptotic quadratic convergence* in the case of strictly convex separable networks under certain relatively mild conditions, just as the dual Newton approach.

3.1.2 Truncated SQP Method

An important embellishment to the dual Newton method was the fact that CG could be easily truncated far from the optimum. The same is possible for the SQP approach, yielding the *Sequential Truncated Quadratic Programming (STQP) method*, as described in [6].

The literature on this truncation is somewhat too oriented toward non-linearly constrained optimization, in which truncation serves purposes other than saving CPU time, which leads to great complexities. For the truncated Newton algorithm for unconstrained optimization, the results were simple: Convergence is superlinear if the residual in CG is $\|\mathbf{r}\| = o(\|\mathbf{g}\|)$, and quadratic if $\|\mathbf{r}\| = o(\|\mathbf{g}\|^2)$. This lead to the choice for the forcing sequence $\eta = \min\left\{\frac{1}{k}, \|\mathbf{g}\|\right\}$ given earlier. The only rigorous result known to the authors, given in [6], is that STQP will be superlinearly convergent if the quadratic program 31 is solved using CG with a final residual $\|\mathbf{r}\| = o(\|\Delta\mathbf{x}\|) \approx o(\left\|\mathbf{H}^{-1}(\mathbf{A}^T\bar{\boldsymbol{\lambda}} - \mathbf{g})\right\|)$, where $\Delta\mathbf{x}$ can most likely be approximated with the previous search direction, $\Delta\bar{\mathbf{x}}$. This leads to the forcing sequence,

$$\eta = \frac{\|\mathbf{r}\|}{\|\Delta\bar{\mathbf{x}}\|} = \frac{1}{k} \tag{33}$$

which guarantees superlinear convergence. We are not aware of a result discussing whether a more strict forcing sequence guarantees faster convergence. For example, it may be that a more suitable sequence is,

$$\eta = \min\left\{\frac{1}{k}, \|\Delta\bar{\mathbf{x}}\|\right\} \tag{34}$$

giving quadratic convergence. We are currently exploring more literature on the issue. As previously, taking the gradient as the first guess in PCG is a good idea. Notice that truncation destroys exact feasibility, so that in a sense STQP follows a semi-feasible path to the optimum. We have therefore explicitly included the residual in the formulation, $\mathbf{A}(\Delta\mathbf{x}) = \mathbf{b} - \mathbf{A}\bar{\mathbf{x}}$, instead of using $\mathbf{A}(\Delta\mathbf{x}) = \mathbf{0}$ as in [12], which assumes a start from a feasible point and no truncation.

4 Relation of Dual Approaches to Network Optimization and Physics

In the previous two sections we discussed the TCGN and STQP methods for solving problem 2, but did not relate this specifically to network optimization, problem 1, as encountered in physics applications. Here we discuss some specific implementation details, as well as relate the previous mathematics to the physics of the problems. In this section, variables indexed as $\mathbf{H}_{ij}$ denote usual matrix indexing, while indexes such as $\mathbf{H}_{(i,j)}$ or $\mathbf{H}_{\{i,j\}}$ denote the elements of the matrix corresponding to a given arc $e_{(i,j)}$ or link $e_{\{i,j\}}$. For nodal quantities there is no difference in the two notations.

4.1 The Structure of the Hessian

4.1.1 Products of the Hessian with a Vector

The most computationally expensive part of both dual algorithms presented so far is the solution of a linear system with a matrix of the form $\mathbf{ADA}^T$ as a coefficient matrix, where $\mathbf{D}$ is a diagonal matrix. Recall that in network optimization the constraint matrix $\mathbf{A}$ is the node-arc incidence matrix of the network and has very special sparse structure—it has one $+1$ and one -1 in each column:

$$\mathbf{A}\left(i, e_{(i,j)}\right) = 1,\ \mathbf{A}\left(j, e_{(i,j)}\right) = -1 \tag{35}$$

This means that matrix-vector products of the form $\mathbf{A}^T\boldsymbol{\lambda}$ and $\mathbf{Ax}$ can very easily be computed using only a few vectors. In particular, it is easy to show that,

$$\begin{aligned} \left(\mathbf{A}^T\lambda\right)_{e_{(i,j)}} &= \lambda_i - \lambda_j \\ (\mathbf{Ax})_i &= \sum_{e_{(i,j)} \in E} x_j - \sum_{e_{(j,i)} \in E} x_j \end{aligned} \tag{36}$$

and also:

$$\left(\mathbf{A}\mathbf{D}\mathbf{A}^T\right)_{ii} = \sum_{e_{\{i,j\}} \in E} D_{\{i,j\}} \tag{37}$$

$$\left(\mathbf{A}\mathbf{D}\mathbf{A}^T\right)_{ij} = \left\{ \begin{array}{c} -D_{\{i,j\}} \text{ if } e_{\{i,j\}} \in E \\ 0 \text{ otherwise} \end{array} \right\}$$

This simply means that the Hessian matrix has the same sparsity structure as the *adjacency matrix* of the graph, and is therefore sparse, which opens possibilities for sparse incomplete factorizations. This gives a very simple way of calculating the product of the Hessian with a vector, as needed in CG, first discussed in [12]:

$$\left[\left(\mathbf{A}\mathbf{D}\mathbf{A}^T\right)\boldsymbol{\lambda}\right]_i = \sum_{e_{\{i,j\}} \in E} D_{\{i,j\}}(\lambda_i - \lambda_j) \tag{38}$$

Various node and arc numbering optimization techniques should be used in an actual computer implementation when computing 38 to optimize cache and memory bandwidth performance, as discussed in [19]. In fact, since the matrix-vector product is the core of the computation in PCG, special optimization techniques may need to be used.

4.1.2 Singularity of the Hessian

Notice that the sum of each row in the matrix $\mathbf{A}\mathbf{D}\mathbf{A}^T$ is zero[22]. Thus the vector of all ones $\hat{\boldsymbol{\lambda}}$, $\hat{\boldsymbol{\lambda}}_i = 1$, is in the null-space of the Hessian, $\mathbf{H}\hat{\boldsymbol{\lambda}} = \mathbf{0}$. This means that the Hessian is a *semi-definite matrix* with exactly one zero eigenvalue, and is therefore non-invertible. This is very understandable from a physics point of view because the potentials are determined physically up to a constant, and adding a multiple of $\hat{\boldsymbol{\lambda}}$ to the solution $\bar{\boldsymbol{\lambda}}$ does not change the nature of the solution. There is also no need to panic about this in relation to non-invertibility of the coefficient matrix, since CG can solve semidefinite problems: PCG simply leaves the component of the initial guess for the solution in the null-space of the coefficient matrix unchanged. In our case, this means that the mean value of $\boldsymbol{\lambda}$ will not change during the course of the optimization.

There are of course formal ways of making the Hessian matrix non-singular, and the two main techniques, which amount to the same effect, are to either add a dummy node not included in $\boldsymbol{\lambda}$ but connected to other (possibly all) nodes via very high cost arcs, or removing one of the nodes of the network from $\boldsymbol{\lambda}$ (this is equivalent to fixing its potential to 0), but leaving the arcs that end in this *rooting node* $\mathcal{R}$ in the diagonal elements of the Hessian. Formally, both approaches amount to making the Hessian matrix a matrix of the form $\bar{\mathbf{A}}\mathbf{D}\bar{\mathbf{A}}^T + \bar{\mathbf{D}}$, where $\bar{\mathbf{D}}$ is a diagonal matrix containing the elements of the Hessian that belong to the non-empty set of arcs originating or ending in the root node, and $\bar{\mathbf{A}}$ is the node-arc incidence matrix of the network without the root node

[22] The diagonal equals the negative of the sum of the off-diagonal entries in each row.

and its arcs. This form of the matrix is invertible, and there may be more than one roots.

Of particular interest, especially to the primal Newton method and network preconditioning documented later on is the case when the incidence matrix $\bar{\mathbf{A}}$ corresponds to a *spanning tree in the network rooted at the root* $\mathcal{R}$, called a *basis* $\mathbf{B}$ from now on. In this case the matrix $\bar{\mathbf{A}}\mathbf{D}\bar{\mathbf{A}}^T + \bar{\mathbf{D}}$ can be inverted easily by traversing the tree from the leafs up and/or from the root down, as used in the complicated algorithms of the network simplex algorithm, discussed in [8]. In fact, we will say that the node-arc incidence matrix $\mathbf{B}$ is invertible, but carrying in mind that this is really $\bar{\mathbf{A}}$ with the added arcs to the root, and write:

$$\left(\bar{\mathbf{A}}\mathbf{D}\bar{\mathbf{A}}^T + \bar{\mathbf{D}}\right)^{-1} = \left(\mathbf{B}^T\right)^{-1}\mathbf{D}^{-1}\mathbf{B}^{-1} \tag{39}$$

4.1.3 Numerical Conditioning

Although we assumed that problem 1 was strictly convex and thus guaranteed a unique global minimum, this minimum may not be easy to find with the above methods numerically due to numerical stability issues. In particular, the sizes of the diagonal entries in the conjugate Hessian in the dual Newton method may be very large of very small. If the second derivative of the cost functions $f_i''(x)$ are very small for a certain range of flows or zero at a certain point $\tilde{x}$ (as is the case in several important physics applications), the conjugate Hessians may overflow or not be defined.

This may be mended in various ways, from more formal and complex (and possibly more efficient), to intuitive and simple. For example, we may simply choose to bound the sizes of the entries in the conjugate Hessian from above and below, as done in [4], $\mathbf{H}_{ij} \leftarrow \min\{\max\left(\mathbf{H}_{ij}, \underline{\varepsilon}\right), \frac{1}{\bar{\varepsilon}}\}$ for some small constants $\bar{\varepsilon}$ and $\underline{\varepsilon}$. A more formal and universal approach is the previously described Primal-Dual Proximal Point algorithm, which is intended for the solution of non-strictly convex programs, but can aid significantly in the solution of ill-conditioned strictly convex problems. To review, this method simply augments the cost function , i.e. the Lagrangian, with proximal distance terms, for example quadratic terms, that insure strict convexity,

$$\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = \sum_i \max_{x_i}\left[x_i t_i - f_i(x_i) - \frac{\varepsilon\left(\mathbf{M}_x\right)_{ii}}{2}(x_i - \bar{x}_i)^2\right] - \boldsymbol{\lambda}^T\mathbf{b} + \frac{\varepsilon}{2}\left(\boldsymbol{\lambda} - \bar{\boldsymbol{\lambda}}\right)^T\mathbf{M}_\lambda\left(\boldsymbol{\lambda} - \bar{\boldsymbol{\lambda}}\right) \tag{40}$$

where ε is a forcing and stabilizing parameter that is reduced as the optimization proceeds toward the optimum, and bared quantities denote the values at the previous iteration (the current estimates). The effect of the augmented terms is not only to make all functions strictly convex, but also to limit the size of the step that can be taken away from the current point. We also allowed for the possibility of using different diagonal matrices, $\mathbf{M}_x$ and $\mathbf{M}_\lambda$, in the distance functions for the primal and dual variables.

This kind of numerical conditioning may be very important in physics applications, where second derivatives may nearly vanish in certain regions (see Appendix B). For example, assume that $f'(x) = 1 + \tanh \frac{|x| - x_0}{\xi}$. Although still strictly monotonically increasing, this gives a function $f(x)$ which is nearly linear in asymptotic regions $||x| - x_0| \gg \xi$. Although the proximal point algorithm in 40 is not the best algorithm for dealing with not strictly convex problems, it is a way to incorporate this possibility within the framework of the dual algorithms discussed in this paper. It does however require including a quadratic term $\frac{\varepsilon}{2}(x_i - \bar{x}_i)^2$ in the primal functions, which may complicate finding the conjugate functions numerically.

In the proximal-point algorithm, the proximal parameter ε is kept constant during a super iteration, and then reduced for the next super iteration. However, each super iteration requires solving (albeit inexactly) a problem such as 2, so this seems like a very costly algorithm. Another approach would be to decrease ε at each major iteration, and dispense with super iterations altogether. Since here we are mostly concerned with using the proximal regularization as a numerical conditioner, this is probably allowed. However, convergence results, including the ones for the Truncated Newton algorithm, cease to be strictly valid[23]. Obviously, there is much experimenting to be done before more can be said.

Even after such numerical stabilization techniques are applied to problem 1, the Newton system of equations 16 may still be ill-conditioned, and suitable preconditioning strategies need to be used. We discuss this next.

4.2 Preconditioning the Newton System

Preconditioning is a very important issue, but it is immature to discuss it in great detail before a practical implementation of the basic algorithm is implemented and the hang-ups/hotspots seen clearly. Of course, the diagonal of the Hessian, given in 37, can always be used as a preconditioner.

4.2.1 MST Preconditioner

There is at least one preconditioner aimed specifically at network optimization documented in literature on interior-point methods for linear network optimization, summarized in [3], and improved upon in [2]. It is interesting to point out that we considered this approach independently prior to finding it in the literature. As explained earlier, it is possible to invert the Hessian matrix explicitly if the node-arc incidence matrix corresponds to a *rooted spanning tree*[24], or a basis $\mathbf{B}$. This hints to partitioning E into those arcs that belong to the spanning tree—basic arcs, and those that don't—non-basic arcs, and partitioning all the

[23] In other words, we are on our own.

[24] The inverse depends on the choice of the root, since an unrooted tree gives a non-invertible Hessian.

arrays accordingly[25]:

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{N} \end{bmatrix} \text{ and also } \mathbf{H}^* = \begin{bmatrix} \mathbf{H}_B^* & \\ & \mathbf{H}_N^* \end{bmatrix} \tag{41}$$

Then we have that,

$$\mathbf{H} = \mathbf{A}\mathbf{H}^*\mathbf{A}^T = \mathbf{B}\mathbf{H}_B^*\mathbf{B}^T + \mathbf{N}\mathbf{H}_N^*\mathbf{N}^T = \mathbf{H}_B + \mathbf{H}_N \tag{42}$$

where now $\mathbf{H}_B$ is invertible[26], $\mathbf{H}_B^{-1} = \left(\mathbf{B}^T\right)^{-1}\left(\mathbf{H}_B^*\right)^{-1}\mathbf{B}^{-1}$, while $\mathbf{H}_N$ is likely not. $\mathbf{H}$ can formally be inverted using a well-known linear algebra formula,

$$\begin{aligned} \mathbf{H}^{-1} &= \left(\mathbf{H}_B + \mathbf{N}\mathbf{H}_N^*\mathbf{N}^T\right)^{-1} = \mathbf{H}_B^{-1} - \mathbf{H}_B^{-1}\mathbf{N}\left[\left(\mathbf{H}_N^*\right)^{-1} + \mathbf{N}^T\mathbf{H}_B^{-1}\mathbf{N}\right]^{-1}\mathbf{N}^T\mathbf{H}_B^{-1} \\ &= \mathbf{H}_B^{-1} - \mathbf{H}_B^{-1}\mathbf{N}\mathcal{H}^{-1}\mathbf{N}^T\mathbf{H}_B^{-1} \end{aligned} \tag{43}$$

where now $\mathcal{H} = \left(\mathbf{H}_N^*\right)^{-1} + \mathbf{N}^T\mathbf{H}_B^{-1}\mathbf{N}$ is a matrix of the same form and structure as the Hessian in the primal Newton method discussed later on, and is not really trivially invertible either. However, notice that if we choose the partition so that $\|\mathbf{H}_B^*\| \gg \|\mathbf{H}_N^*\|$, that is, if we choose as our basis the *maximal weight spanning tree* (MST) with the conjugate Hessians as weights, than the term $\mathbf{H}_B^{-1}$ will dominate the inverse an can be used as the preconditioning matrix:

$$\mathbf{M}_1^{-1} = \mathbf{H}_B^{-1} \tag{44}$$

If this does not produce good enough preconditioning, then we can include additional correction terms. Such techniques based on QR and incomplete Cholesky factorizations are discussed in [2], but here we present some novel ideas which can be implemented trivially once the basic preconditioner 44 is implemented, and that we believe may work very well. For example, notice that the term $\left(\mathbf{H}_N^*\right)^{-1}$ dominates $\mathcal{H}$. If we ignore the rest and write $\mathcal{H}^{-1} \approx \left(\mathbf{H}_N^*\right)^{-1}$ then we get a new approximation to the inverse of the Hessian that can be used as a preconditioner,

$$\mathbf{M}_2^{-1} = \mathbf{H}_B^{-1} - \mathbf{H}_B^{-1}\mathbf{H}_N\mathbf{H}_B^{-1} \tag{45}$$

which is in fact just the first two terms of the formal power series,

$$\mathbf{H}^{-1} = \left(\mathbf{H}_B + \mathbf{H}_N\right)^{-1} = \left[\sum_{k=0}^{\infty}(-1)^k\left(\mathbf{H}_B^{-1}\mathbf{H}_N\right)^k\right]\mathbf{H}_B^{-1} \tag{46}$$

[25] The notation N stands for *non-basic*, however, in the context of primal algorithms with active sets, a more suitable notation is probably S, for *super-basic*, since in simplex-like algorithms superbasic variables are free while non-basic are fixed at the bounds, which are absent in our formulation. We chose this notation in honor of the work in [14].

[26] In writing this inverse we assume that the rooting node $\mathcal{R}$ is also the root for the whole node-arc incidence matrix, i.e. that the row of $\mathbf{A}$ that corresponds to the root has been deleted.

which will converge (rapidly) if the spectral radius $\rho\left(\mathbf{H}_B^{-1}\mathbf{H}_N\right) \ll 1$. This requires that the matrix $\mathbf{H}_B - \mathbf{H}_N$ also be (strongly) positive definite, which will most likely be true when $\|\mathbf{H}_B^*\| \gg \|\mathbf{H}_N^*\|$. It should be noted that a truncated series as the one in 46 is used in subsidiary-iteration preconditioning for PCG used in certain parallel implementations of PCG.

However, we believe that a better solution is to use the fact that $\mathcal{H}$ is a nearly diagonal matrix, so that a good approximation is to say[27] $\mathcal{H} \approx \mathcal{D}(\mathcal{H})$, and use the preconditioner,

$$\mathbf{M}_3^{-1} = \mathbf{H}_B^{-1} - \mathbf{H}_B^{-1}\mathbf{N}\left[\mathcal{D}(\mathcal{H})\right]^{-1}\mathbf{N}^T\mathbf{H}_B^{-1} \tag{47}$$

where the diagonal of the term $\mathbf{N}^T\mathbf{H}_B^{-1}\mathbf{N}$ in $\mathcal{D}(\mathcal{H})$ can easily be calculated using the tree structure and the basic equivalent cycle $\mathcal{S}_{(i,j)}$ formed in the tree when non-tree arc $e_{(i,j)}$ is added, as documented in reference [14],

$$\left[\mathcal{D}(\mathcal{H})\right]_{(i,j)} = \sum_{e_{\{k,l\}} \in \mathcal{S}_{(i,j)}} \left(\mathbf{H}^*_{\{k,l\}}\right)^{-1} \tag{48}$$

where presumably the non-basic contribution $\left(\mathbf{H}^*_{\{i,j\}}\right)^{-1}$ will dominate. The data structures needed to finding the cycle $\mathcal{S}_{(i,j)}$ are also needed in the process of updating the spanning tree, so this poses no additional burden on the implementation. Formally, one could use PCG to invert $\mathcal{H}$, using its diagonal given in 48 as a preconditioner, which should give very fast convergence. But this will require repeated multiplication with $\mathbf{H}_B^{-1}$, which, although still $O(m)$ in time complexity, entails complicated tree operations which are not easily optimized for cache performance or for parallelization/vectorization[28].

Notice that there is no guarantee that the preconditioning matrices M_2 and M_3 in equations 45 and 47 are positive definite, as PCG requires. Experimenting with the above three preconditioners at different stages of the optimization is highly recommended. Also notice that we are not aware of previous investigations of 45 and 47, so this is really only a proposal at this stage.

At this point it is important to emphasize that the most difficult step in implementing these preconditioners, and probably a hinderance to their instant success, is the process of creating and updating the maximal weight spanning tree for $\mathbf{B}$, $\mathcal{B}$. As the optimization algorithms proceeds, the weights on the arcs (Hessian values) will change, which requires updating the tree $\mathcal{B}$. This can be done efficiently using an update procedure based on basic equivalent cycles, as discussed in [14]. But data structures become complicated and storage demanding, and the process of updating these data structures, and in particular the ones needed for the inversion $\left(\mathbf{B}^T\right)^{-1}\boldsymbol{\lambda}$ and $\mathbf{B}^{-1}\mathbf{x}$, becomes a challenge[29]. Ideas for parallelizing the above preconditioners are even more entangled and

[27] This is used in the primal Newton method, as discussed in reference [14].

[28] This will be necessary in the Primal Newton method, discussed next, if high accuracy for the solution is required.

[29] To put it midly, [8] is no bed-time reading!

will be discussed in later reports only if/after they are found to be successful in a serial setting.

4.2.2 Other Preconditioners

Of course, there are numerous other preconditioners that can be used, starting from general ones such as incomplete Cholesky factorization, to more specialized ones. In particular, finite element applications have stimulated research in preconditioners aimed at problems of similar structure.

Most applicable FEM preconditioners which we know of are based on partitioning the system 16 into p smaller problems, which are then solved independently and the results combined using the coupling between the subproblems. This offers saving in both time and space if the coupling is small, so that most of the computation is expended on solving the subsystems. If the partition into subproblems is based on partitioning the nodes $V = \{V_i \mid 1 \leq i \leq p\}$, then this is a standard domain-decomposition preconditioning, as described for example in [5]. If instead we partition the edges $E = \{E_i \mid 1 \leq i \leq p\}$, then a recent element-by-element preconditioning technique for partially separable problems, discussed in [4], becomes applicable.

Both kinds of partitioning would be done so as to minimize the arc/node overlap between the different partitions, and could be based on different types of greedy maximal matchings. However, both algorithms produce subsystems with coefficient matrices that are of the form $\mathbf{A}_i\mathbf{D}_i\mathbf{A}_i^T + \bar{\mathbf{D}}_i$, where $\bar{\mathbf{D}}_i$ is a coupling diagonal matrix with a sparsely populated diagonal. Solving this type of system for medium to large partitions is a challenge we have not yet thought of a good solution for. In most likelihood, it will require algorithms different from the ones discussed in this paper, and will therefore place additional burden on the implementor. We therefore think it is advisable to test the network preconditioner first.

4.3 Summary of Relation to Physics

Most readers are probably aware that for a resistive network the Lagrange multipliers introduced above are the (electric) potentials at the nodes, $\lambda_i = U_i$. Also, the tensions are in fact the voltages across the arcs, $t_{(i,j)} = V_{(i,j)}$. Therefore the statement $x_i^* = (f_i')^{-1}(t)$ simply assigns a current to each arc according to the voltage-current characteristic, $I_{(i,j)} = V_{(i,j)}^{-1}\left(V_{(i,j)}\right)$. In fact, the dual Newton algorithm simply optimizes in potential space, while at each step obtaining the currents from the potential-flow characteristics of the arcs. At the optimum, the flow obtained this way is also feasible. The SQP method on the other hand obtains a feasible flow at each step from an estimate of the voltages, so that the potential-flow characteristics are satisfied exactly only at the optimum.

In the case of non-linear resistive networks, the maximum weight spanning tree $\mathcal{B}$ in the network preconditioning is made up of arcs of low resistance, which determine the current to a first approximation. In a sense, the tree is the high conductivity backbone of the network. In some cases, there are theoretical

results about the form of the solution at the optimum. For linear network problems, it is known that the solution has a spanning tree structure. This is also true for non-convex arcs with power law voltage-current characteristics, $V(I) \sim I^\gamma$, $\gamma \leq 1$, but is not true for $\gamma \geq 1$. This shows that the nature of the solution and the success of certain preconditioners depends to a large degree on the type of problem. The same is in particular true for forming the initial guess for the flow[30]. In some cases, such as resistive networks made of superconducting or dielectric-like arcs, discussed in Appendix B, there are good approximations to the solution, such as maximum-flow or minimal-path flow patterns[31]. These can be found efficiently using combinatorial techniques and should also be used if an initial feasible flow is needed, such as in the Primal Newton method discussed next.

At this point we will emphasize that the formulation in 1 is not sufficient for physics related purposes. In particular, the supply-demands are not usually known at the boundary nodes in physical systems, since this is a micro detail, while in experiments only a few microscopic values, such as the potential of an electrode or the current through a wire can be set and measured.

Fixed-Potential Boundary Conditions For example, in many cases the potentials are specified at certain boundary nodes $\mathcal{N}_B$, i.e. it is known that $\lambda_{i\in\mathcal{N}_B} = V_i^B$, while the supply-demands are not known at these points (and are thus also a part of the space over which the optimization is carried out). This is reminiscent of finite element formulations for elliptic PDE's, in which one can either specify natural (von Neumann) , or essential (Dirichlet) boundary conditions. In finite elements, the formulation of the stiffness equations is not changed when boundary conditions are changed and a simple solution[32] for satisfying fixed-potential (Dirichlet) boundary conditions is to add large entries to the diagonal elements in the stiffness matrix and to the elements in the load vector corresponding to the boundary nodes, which ensures that at the solution the displacements at the boundary do indeed have the desired values. It seems that such an approach should work in our case as well, although we have not found strict formal derivations in literature[33].

So, we propose to continue to solve the Newton system $\langle \mathbf{H} \rangle \, \Delta\boldsymbol{\lambda} = -\mathbf{g}$ when fixed-potential boundary conditions are specified, only now we know that some nodes should not change the potential, $\Delta\lambda_{i\in\mathcal{N}_B} = 0$, which we ensure by modifying the Newton system so that $\langle \mathbf{H} \rangle_{ii} \leftarrow \langle \mathbf{H} \rangle_{ii} + \frac{1}{\varepsilon}$ for $i \in \mathcal{N}_B$, where ε is a small number, and setting $(\lambda_0)_{i\in\mathcal{N}_B} = (V_B)_i$. As explained earlier, this approximately corresponds to adding dummy root nodes with specified potentials

[30] Initial guesses are very important in non-linear optimization in general and have been somewhat neglected in this summary.

[31] In these cases the critical currents or breakdown potentials are assigned as costs for the arcs.

[32] This solution can very effectively be used with PCG if at least diagonal preconditioning is used.

[33] Note that the unknown supply-demands at the boundary nodes, $\mathbf{b}_B$, do enter in the linear constraints, but not in the cost function, so that the resulting problem is no longer strictly convex.

connected to the boundary nodes via very low-cost arcs (for example, linear resistors[34] of resistance ε). A more formal approach might consist in splitting the nodes into those with known supplies-demands, i.e. not on a boundary, $\mathcal{N}_{NB}$, and those at the boundary, $\mathcal{N}_B$, and splitting all related vectors and matrices accordingly. Then, $\mathbf{b}_B$ becomes a part of the unknowns, but does not enter the cost function, which we can amend with proximal point regularization. Then the arrays in the problem formulation 1 get redefined,

$$\mathbf{A} \leftarrow \begin{bmatrix} \mathbf{A}_{NB} & \\ \mathbf{A}_B & -\mathbf{I} \end{bmatrix}, \mathbf{b} \leftarrow \begin{bmatrix} \mathbf{b}_{NB} \\ \mathbf{0} \end{bmatrix} \text{ and } \mathbf{x} \leftarrow \begin{bmatrix} \mathbf{x} \\ \mathbf{b}_B \end{bmatrix} \tag{49}$$

giving a modified Hessian,

$$\mathbf{H} \leftarrow \mathbf{A}\mathbf{H}^*\mathbf{A}^T + \begin{bmatrix} \mathbf{0} & \\ & \frac{1}{\varepsilon}\mathbf{I} \end{bmatrix} \tag{50}$$

which simply adds large entries to the diagonal entries in the usual Hessian corresponding to the boundary nodes, as we have stipulated. The gradient also changes,

$$\mathbf{g} \leftarrow \mathbf{A}\mathbf{x}^* - \begin{bmatrix} \mathbf{b}_{NB} \\ \bar{\mathbf{b}}_B \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \frac{\boldsymbol{\lambda}_B - \boldsymbol{\lambda}_0}{\varepsilon} \end{bmatrix} \tag{51}$$

where we have added the specified boundary voltages $\boldsymbol{\lambda}_0 = \mathbf{V}_B$ by hand, and $\bar{\mathbf{b}}_B$ is the estimate of the outflow at the boundary nodes at the previous iteration. The assymptotic behaviour, with small ε is clear: The potentials are driven to their desired values and $\mathbf{b}_B$ gets set to the corresponding part[35] of $\mathbf{A}\mathbf{x}^*$, as it should be. Whether this more formal proximal formulation has any advantages over the simple approach described above is not clear.

Flow-Source Boundary Conditions Another interesting case in non-linear resistor network analysis is the case when a constant current source of strength J_B is connected to a set of the nodes $\mathcal{N}_B$ via an electrode, which means that $\sum_{i \in \mathcal{N}_B} b_i = J_B$ and $\lambda_{i \in \mathcal{N}_B} = V^B = const.$ This case can be (at least mentally) represented as a problem of the form 1, in which additional dummy source/sink nodes with specified supply-demands $\pm J_B$ are added and connected to the boundary nodes via very low cost arcs (for example, linear resistors of negligible resistance ε or superconductors with very high critical current). These kinds of ill-behaved elements will of course induce ill-conditioning in the Newton system, but since this is mostly reflected as addition of very large numbers to the diagonal entries in the Hessian, simple diagonal preconditioning should be sufficient to compensate for this. Of course, the proximal point algorithm can be used in this case too by assigning strictly zero cost to the added arcs.

[34] In the proximal point regularization approach described next, the added term to the cost function, $\frac{\varepsilon}{2}(\mathbf{b}_B - \bar{\mathbf{b}}_B)^T(\mathbf{b}_B - \bar{\mathbf{b}}_B)$ is not just a quadratic resistive term, but also adds a correction to the gradient.

[35] That is, the added dummy arcs take all the extra in/outflow at the boundary nodes.

5 The Primal Approach

This section gives a brief overview of the primal approach to solving problem 1, which according to our analysis of the literature is computationally inferior to the dual method, particularly in environments with contested memory resources. However, the only public domain library for convex networks, LSNNO[36], uses this approach and it is thus an interesting method to look at. Our primary reference is [14], but a good overview is also given in [13] and in the primer [15].

5.1 Generalized Elimination

The idea behind the primal approach is very simple and intuitive. It consists of eliminating n of the flow variables using the constraint equation and then substituting the remaining $m-n$ equations into the objective function, giving an easy to solve unconstrained optimization problem in $m-n$ variables.

More formally, the solution space of the full-rank equation $\mathbf{Ax} = \mathbf{b}$ is a vector space of dimension $m-n$, given via the *generalized elimination*:

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{Zy}, \text{ where } \mathbf{AZ} = \mathbf{0} \text{ and } \mathbf{Ax}_0 = \mathbf{b} \text{ and } \mathbf{y} \in \mathcal{R}^{m-n} \tag{52}$$

In other words, every solution of the constraint equation can be decomposed into a *particular* initial feasible solution $\mathbf{x}_0$ (which is obtained via some fairly complicated methods[37]), and a *homogeneous* solution $\mathbf{Zy}$, where the *elimination matrix* $\mathbf{Z}$ spans the null space of $\mathbf{A}$, $\mathbf{AZ} = \mathbf{0}$. The different elimination methods differ in the way one chooses $\mathbf{Z}$. One of the primary disadvantages of the primal method is the need for an initial feasible solution. However, in physics related applications, a feasible flow that is also a good guess for the actual solution of 1 can often be found using some combinatorial approaches, so this may not be such a disadvantage.

5.2 The Primal Problem

Now we can substitute 52 back into the objective function to get the primal unconstrained optimization problem:

$$\min_{\mathbf{y} \in \mathcal{R}^{m-n}} f(\mathbf{x}_0 + \mathbf{Zy}) = \min_{\mathbf{y} \in \mathcal{R}^{m-n}} \tilde{f}(\mathbf{y}) \tag{53}$$

We can use any of the unconstrained optimization methods discussed earlier using the *reduced gradient* and *Hessian* of $\tilde{f}$, $\tilde{\mathbf{g}}$ and $\tilde{\mathbf{H}}$, expressed in terms of the gradient and Hessian of f, $\mathbf{g}$ and $\mathbf{H}$:

[36] There is also another publcily available binary library, PPRN, which is primarily intended for multicommodity non-linear problems with linear side constraints, but can be used for solving problem 1 as well.

[37] Note that there was no need to obtain an initial feasible solution in the dual method, where we could start with an arbitrary random guess $\mathbf{x}_0$.

$$\begin{aligned} \bar{\mathbf{g}} &= \mathbf{Z}^T\mathbf{g} \\ \bar{\mathbf{H}} &= \mathbf{Z}^T\mathbf{H}\mathbf{Z} \end{aligned} \tag{54}$$

It is easy to see that the reduced Hessian $\bar{\mathbf{H}}$ is positive definite, so that the new problem 53 is easy to solve.

5.3 Network Elimination

The only unresolved question about the primal problem is how to find $\mathbf{Z}$. The apparent goal is to make $\mathbf{Z}$ as sparse as possible, so that multiplication of $\mathbf{Z}$ or $\mathbf{Z}^T$ with a vector would take as least operations as possible and so that we can use the truncated Newton method efficiently. The literature offers at least one method for choosing $\mathbf{Z}$ specifically tailored for network optimizations. The method has great similarities to the MST preconditioner for the dual approaches described in the previous sections, and in fact the same data structures and algorithms are needed for the basic implementation. Since the primal method described here, proposed in [14], relies strongly on the extremal property of the spanning tree $\mathcal{B}$, more sophisticated techniques for maintaining $\mathcal{B}$ may be needed for an efficient implementation of the primal method.

We start as in the MST preconditioner by partitioning the node-arc incidence matrix $\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{N} \end{bmatrix}$ into an invertible *basis* $\mathbf{B}$ corresponding to the edges of a rooted spanning tree $\mathcal{B}$ and $\mathbf{N}$ corresponding to the remaining *non-basic* edges. We also partition the flow vector $\mathbf{x} = \begin{bmatrix} \mathbf{x}_B & \mathbf{x}_N \end{bmatrix}$ and all the other arrays like the gradient and the Hessian accordingly, $\mathbf{g} = \begin{bmatrix} \mathbf{g}_B & \mathbf{g}_N \end{bmatrix}$ and $\mathbf{H} = \begin{bmatrix} \mathbf{H}_B & \\ & \mathbf{H}_N \end{bmatrix}$.

Then we get,

$$\mathbf{A}\mathbf{Z} = \begin{bmatrix} \mathbf{B} & \mathbf{N} \end{bmatrix} \begin{bmatrix} \mathbf{Z}_B \\ \mathbf{Z}_N \end{bmatrix} = 0, \text{ so that } \mathbf{B}\mathbf{Z}_B = -\mathbf{S}\mathbf{Z}_N \text{ and } \mathbf{Z}_B = -\left(\mathbf{B}^{-1}\mathbf{S}\right)\mathbf{Z}_N \tag{55}$$

and we are now at will to choose $\mathbf{Z}_S$, and the simplest and very sparse choice is to choose $\mathbf{Z}_S = \mathbf{I}$ to be the identity matrix, to finally get:

$$Z = \begin{bmatrix} -\left(\mathbf{B}^{-1}\mathbf{S}\right) \\ \mathbf{I} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{B}} \\ \mathbf{I} \end{bmatrix} \tag{56}$$

An attractive feature of this approach is that the matrix $\bar{\mathbf{B}} = -\left(\mathbf{B}^{-1}\mathbf{S}\right)$ is directly related to the spanning tree $\mathcal{B}$ and the cycles $\mathcal{S}_{(i,j)}$ introduced by adding non-basic arcs $e_{(i,j)}$ to $\mathcal{B}$, and earlier approaches, such as [15], stored $\bar{\mathbf{B}}$ in sparse format. This $O(n \log n)$ storage is not acceptable in very large-scale approaches and we do not recommend it.

If we now substitute equation 56 into 54 we get,

$$\begin{aligned} \bar{\mathbf{g}} &= \mathbf{g}_N - \left(\mathbf{B}^{-1}\mathbf{S}\right)\mathbf{g}_B \\ \bar{\mathbf{H}} &= \mathbf{H}_N + \mathbf{N}^T\left(\mathbf{B}^{-T}\mathbf{H}_B\mathbf{B}^{-1}\right)\mathbf{N} \end{aligned} \tag{57}$$

where the reduced gradient and the product of the reduced Hessian with a vector can be calculated explicitly in $O(n)$ time, so that TCGN can be used to solve problem 53. However, just as with the network preconditioner, if the spanning tree is chosen as a *minimal weight spanning tree* with the primal Hessians $H_{(i,j)}$ used as weights for the arcs, then $\bar{\mathbf{H}}$ will be a nearly diagonal matrix dominated by $\mathbf{H}_N$, and where the diagonal contribution due to the additional term can also be easily calculated. The work in [14] found that using a diagonal approximation for $\bar{\mathbf{H}} \approx \mathcal{D}\left(\bar{\mathbf{H}}\right)$ in Newton's method with an adaptive maintenance of $\mathcal{B}$ gave fast convergence to solutions of medium accuracy. If higher accuracy is needed, TCGN can be used with $\mathcal{D}\left(\bar{\mathbf{H}}\right)$ as a preconditioner. In fact, we still recommend using TCGN as the outer iteration, to accommodate different possibilities.

6 Summary

In this overview, we have described in detail three efficient algorithms for solving large-scale instances of problem 1 and related them to physics applications, in particular non-linear resistive networks.

The first algorithm, deemed "best", the[38] *Dual Truncated Preconditioned Conjugate Gradient Newton (DTPCGN) algorithm*, has the advantage of transforming the original problem into an equivalent strictly-convex unconstrained optimization problem, that it can start at an arbitrary point when no capacity constraints on the arcs are present, or it can incorporate bound-constraints easily via logarithmic barrier methods, and can also incorporate non-strictly convex programming via proximal point regularization. The method is also computationally very simple, at least in its basic variant, since the primary computational step, calculation of the product of the Hessian with a vector, can be well optimized even for parallel architectures. The primary disadvantage of the method is the possible large computational cost associated with repeatedly solving Newton's equations, which can be mitigated with preconditioning, but preconditioning is highly non-trivial for this case, although we proposed a preconditioning strategy based on maximal weight spanning trees that may prove effective. Implementing this preconditioning will also open an implementation path to the primal Newton method.

Another disadvantage of the Dual Newton method is the need for an implementation of the calculation of the conjugate convex functions and their derivatives, which hinders plug-and-play library implementations and may not

[38] We have invented these algorithm names and acronyms for our own use, they are not to be taken too seriously.

be possible to code efficiently for very peculiar cost functions. If this is the case, then the *Sequential Truncated Dual-Based Quadratic Programming (STDQP)* method may be a good alternative. The method is computationally almost identical to DTPCGN, but it follows a different, semi-feasible path to the optimum, so it is well worth including it in this work, although we still deem DTPCGN will outperform in most circumstances. This is because STDQP restricts the freedom of the optimization algorithm to a semi-feasible path of quadratic subproblems, makes truncation more complicated, and makes proximal regularizations more difficult.

Finally, if the MST preconditioning for DTPCGN is implemented, the *Primal Truncated MST-Based PCG Newton (PTMSTPCGN) algorithm* can also be implemented at little extra cost, giving a rather different, strictly feasible, approach to solving 1. The basic steps in the algorithm become tree operations, which may not be easy to optimize on current architectures, are difficult to parallelize, and have logarithmic cost $O(n \log n)$. Still, the fact that the Hessian matrix can be brought to a near diagonal form makes the CG solver very cost efficient and may mitigate some of the disadvantages of PTMSTPCGN.

It can never be overstressed that for maximal efficiency the algorithm should be tuned for the *particular* problem at hand, i.e. having a special network structure and cost function in mind.

References

[1] *Algorithms for Network Flow Problems with Convex Separable Costs*, **J.G. Klincewicz**, Ph.D. thesis, Yale University, 1974

[2] *Conjugate Gradient Based Implementation of Interior Point Methods for Network Flow Problems*, **S. Mehrotra** and **J.S. Wang**, Nortwestern University, TR 95-70.1

[3] *Interior Point Algorithms for Network Flow Problems*, **M.G.C. Resende** and **P.M. Pardalos**, Advances in linear and integer programming, Oxford Univ. Press, also avalable on-line (the best place to look is the ResearchIndex database at *http://citeseer.nj.nec.com*)

[4] *On the Use of Element-by-Element Preconditioners to Solve Large Scale Partially Separable Optimization Problems*, **M.J. Dayde**, **JY L'Excellent** and **N.I.M. Gould**, ENSEEIHT-IRIT, RT/APO/94/4, also on-line

[5] *A Domain Decomposition Preconditioner for a Parallel Finite Element Solver on Distributed Unstructured Grids*, **D.C. Hodgson** and **P.K. Jimack**, University of Leeds, TR 95.1, on-line

[6] *Sequential Truncated Quadratic Programming Methods*, **R.S. Dembo** and **U. Tulowitzki**, Numerical Optimization Proceedings, 1984

[7] *A Partitioned Epsilon-Relaxation Algorithm for Separable Convex Network Flow Problems*, **D.R. Leone**, **R.R. Meyer** and **A. Zakarian**, Computational Optimization and Applications, vol. 12, p.p. 107-126

[8] *Primal Simplex Network Codes: State-of-the-Art Implementation Technology*, **A.I. Ali**, **R.V. Helgason**, **J.L. Kennington** and **H.S. Lall**, Networks vol. 8, p.p. 315-339

[9] *Primal-Dual Proximal Point Algorithm for Linearly Constrained Convex Programming Problems*, **S. Ibaraki**, **M. Fukushima** and **T. Ibaraki**, Computational Optimization and Applications, vol.1, p.p. 207-226

[10] *Dual-Based Newton Methods for Nonlinear Minimum Cost Network Flow Problems*, **S. Ibaraki**, **M. Fukushima** and **T. Ibaraki**, Journal of Operations Research, vol. 34, p.p. 263-286

[11] *Computational Comparisons of Dual Conjugate Gradient Algorithms for Strictly Convex Networks*, **C.H. Wu**, **J.A. Ventura**, **S. Browning**, Computers & Operations Research, vol. 25, p.p. 333-349

[12] *A Newton Method for Convex Separable Network Flow Problems,* **J.G. Klincewicz**, Networks, vol. 13, p.p. 427-442.

[13] *On Large Scale Nonlinear Network Optimization*, **Ph.L. Toint** and **D. Tuyttens**, Mathematical Programming, vol. 48, p.p. 125-159

[14] *Interior Methods for Nonlinear Minimum Cost Network Flow Problems*, **R. Katsura**, **M. Fukushima** and **T. Ibaraki**, Journal of the Operations Research Society of Japan, vol. 32, p.p. 174-199

[15] *A Primal Truncated Netwton Algorithm with Application to Large-Scale Nonlinear Network Optimization*, **R.S. Dembo**, Mathematical Programming Study 31, p.p. 43-71

[16] *Truncated-Newton Algorithms for Large-Scale Unconstrained Optimization*, **R.S. Dembo** and **T. Steihaug**, Mathematical Programming, vol. 26, p.p. 190-212

[17] *A Sparse Sequential Quadratic Programming Algorithm*, **R.H. Nickel** and **J.W. Tolle**, Journal of Optimization Theory and Applications, vol. 60, p.p. 453-473

[18] *Computational Development of a Lagrangian Dual Approach for Quadratic Networks*, **J.A. Ventura**, Networks, vol. 21, p.p. 469-485

[19] *Improving Memory Hierarchy Performance for Irregular Applications*, **J.M. Crummey**, **D. Whalley**, **K. Kennedy**, also on-line at *http://www.cs.rice.edu/~johnmc/memhier/*