Les Cahiers du GERAD ISSN: 0711–2440

Tulip: An open-source interior-point linear optimization solver with abstract linear algebra

M. F. Anjos, A. Lodi,
M. Tanneau

G–2019–36

June 2019

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2019
– Bibliothèque et Archives Canada, 2019

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2019
– Library and Archives Canada, 2019

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Tulip: An open-source interior-point linear optimization solver with abstract linear algebra

Miguel F. Anjos [a,b,c]

Andrea Lodi [a,c,d]

Mathieu Tanneau [a,c,d]

[a] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[b] *School of Mathematics, University of Edinburgh, Edinburgh, EH9 3FD, United Kingdom*

[c] *Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

[d] *Canada Excellence Research Chair in Data Science for Real-time Decision-making, Montréal (Québec) Canada, H3C 3A7*

anjos@stanfordalumni.org
andrea.lodi@polymtl.ca
mathieu.tanneau@polymtl.ca

June 2019
Les Cahiers du GERAD
G–2019–36

Abstract: This paper introduces the algorithmic design and implementation of Tulip, an open-source interior-point solver for linear optimization. It implements the homogeneous interior-point algorithm with multiple centrality corrections, and therefore handles unbounded and infeasible problems. Tulip's most remarkable feature is that its algorithmic framework is fully disentangled from linear algebra implementations. This allows to seamlessly integrate specialized routines for structured problems, which we illustrate in the case of structured problems found in Dantzig-Wolfe decomposition. Extensive computational results are reported. We find that, using general-purpose sparse linear algebra, Tulip is competitive with open-source interior-point solvers on the Netlib LP testset. Furthermore, armed with specialized linear algebra, Tulip outperforms state-of-the-art commercial solvers on a set of large-scale, decomposable instances that arise in power systems operation.

Keywords: Linear programming, interior-point methods

Acknowledgments: Mathieu Tanneau was supported by an excellence doctoral scholarship from FQRNT.

1 Introduction

Linear programming (LP) algorithms have been around for over 70 years, and LP remains a fundamental paradigm in optimization. Indeed, although nowadays most real-life applications involve discrete decisions or non-linearities, the methods employed to solve them often rely on LP as their workhorse. Besides algorithms for mixed-integer linear programming (MILP), these include cutting-plane and outer-approximation algorithms that substitute a non-linear problem with a sequence of iteratively refined LPs [28, 31, 37]. Furthermore, LP is at the heart of classical decomposition methods such as Dantzig-Wolfe and Benders decompositions [7, 13]. Therefore, efficient and robust LP technology is instrumental to our ability to solve more involved optimization problems.

Over the past few decades, interior-point methods (IPMs) have become a standard and efficient tool for solving LPs [19, 38]. While IPMs tend to overcome Dantzig's simplex algorithm on large-scale problems, the latter is well-suited for solving sequences of closely related LPs, by taking advantage of an advanced basis. Nevertheless, beyond sheer performance, it is now well recognized that a number of LP-based algorithms can further benefit from IPMs, despite their limited ability to warm start. In cutting plane algorithms, stronger cuts are often obtained by cutting off an interior point rather than an extreme vertex [10, 31, 32]. Similarly, in the context of decomposition methods, well-centered interior solutions typically provide a stabilization effect [20, 22, 34]. This, in turn, reduces tailing-off and improves convergence.

The remarkable performance of IPMs stems from both strong algorithmic foundations and efficient linear algebra. Indeed, the main computational effort of IPMs resides in the resolution, at each iteration, of a system of linear equations, commonly known as the Newton system. Therefore, the efficiency of the underlying linear algebra has a direct impact of the method's overall performance.

While most IPM solvers employ general-purpose sparse linear algebra routines, substantial speedups can be obtained by exploiting a problem's specific structure. For instance, block-angular matrices typically arise in stochastic programming when using scenario decomposition. In [9] and later in [27], the authors thus design specialized factorization techniques that outperform generic implementations. Gondzio [24] observed that the master problem in Dantzig-Wolfe decomposition possesses a block-angular structure. Similar approaches have been explored for network flow problems [12], and for solving facility location problems with Benders decomposition [11].

The aforementioned works focus on devising specialized linear algebra for a particular structure or application. On the other hand, a handful of IPM codes that accommodate various linear algebra implementations have been developed. The OOQP software, developed by Gertz and Wright [17], uses object-oriented design so that data structures and linear algebra routines can be tailored to specific applications. Motivated by large-scale stochastic programming, PIPS [29] incorporates a large share of OOQP's codebase, alongside specialized linear solvers for block-angular matrices. In a similar fashion, OOPS [21, 23] implements custom linear algebra that can exploit arbitrary block matrix structures. Nevertheless, to the best of the authors' knowledge, OOQP is no longer actively maintained, while current development on PIPS focuses on non-linear programming.[1] We also note that both PIPS and OOPS are primarily intended for massive parallelism on high-performance computing infrastructure, and that OOPS is distributed under a closed-source proprietary license.

In this paper, we describe Tulip, an open-source structure-independent interior-point solver written in Julia [8]. Tulip leverages Julia's multiple dispatch and built-in support for linear algebra, thus allowing to disentangle the IPM algorithmic framework from linear algebra implementations.

1.1 Contributions and outline

The remainder of the paper is structured as follows. In Section 2, we introduce some notations and relevant definitions.

[1]Personal communication with PIPS developers.

In Section 3, we describe the homogeneous self-dual embedding, and Tulip's homogeneous interior-point algorithm. This feature contrasts with most IPM LP codes, namely, those that implement the almost-ubiquitous infeasible primal-dual interior-point algorithm [30]. The main advantage of the homogeneous algorithm is its ability to return certificates of primal or dual infeasibility. It is therefore better suited for use within cutting-plane algorithms or decomposition methods, wherein one may encounter infeasible or unbounded LPs.

In Section 4, we provide further implementation details of Tulip, such as the treatment of variable bounds, default values of parameters, and how the algorithm is decoupled from the linear algebra. Tulip is publicly available [35] under an open-source license. It can be used as a stand-alone package in Julia, and through the solver-independent `MathProgBase` interface [3].

In Section 5, we present an example of specialized linear algebra, in the context of Dantzig-Wolfe decomposition. Specifically, we highlight the structure of the master problem, and implement a specialized Cholesky factorization for solving the resulting normal equations system. This extends earlier work by Gondzio [24] by considering structurally dense columns.

In Section 6, we report on two sets of computational experiments. First, we compare Tulip to several open-source and commercial IPM solvers on the classical Netlib LP testset. We observe that, using generic sparse linear algebra, Tulip is competitive with open-source IPM solvers. Second, we benchmark Tulip against commercial IPM solvers in the context of large-scale column generation, and obtain state-of-the-art performance.

Finally, Section 7 concludes the paper and highlights future research directions.

2 Notations

We consider LPs in primal-dual standard form

$$(P) \quad \begin{array}{rrl} \displaystyle\min_{x} & c^T x & \\ s.t. & Ax & = b, \\ & x & \geq 0, \end{array} \qquad (D) \quad \begin{array}{rrl} \displaystyle\max_{y,s} & b^T y & \\ s.t. & A^T y + s & = c, \\ & s & \geq 0, \end{array} \tag{1}$$

where $c, x, s \in \mathbb{R}^n$, $b, y \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$ is assumed to have full row rank. We follow the usual notations from interior-point literature, and write X (resp. S) the diagonal matrix whose diagonal is given by x (resp. s), i.e., $X := Diag(x)$ and $S := Diag(s)$.

We denote I the identity matrix and e the vector with all coordinates equal to one; their respective dimensions are always obvious from context. The norm of a vector is written $|\cdot|$ and, unless specified otherwise, it denotes the ℓ_∞ norm.

A primal solution x is feasible if $Ax = b$ and $x \geq 0$. A strictly feasible (or interior) solution is a primal feasible solution with $x > 0$. Similarly, a dual solution (y, s) is feasible if $A^T y + s = c$ and $s \geq 0$, and strictly feasible if, additionally, $s > 0$. Finally, a primal-dual solution (x, y, s) is optimal for (1) if x is primal-feasible, (y, s) is dual-feasible, and their objective values are equal, i.e., $c^T x = b^T y$.

A solution (x, y, s) with $x, s \geq 0$ is strictly complementary if

$$\forall i \in \{1, \dots, n\}, \big(x_i s_i = 0 \text{ and } x_i + s_i > 0\big). \tag{2}$$

The complementary gap is defined as $x^T s$. When (x, y, s) is primal-dual feasible, the complementary gap equals the classical optimality gap, i.e., we have $x^T s = c^T x - b^T y$.

For ease of reading, we assume, without loss of generality, that all primal variables are required to be non-negative. The handling of free variables and of variables with finite upper bound will be detailed in Section 4.

3 Homogeneous self-dual algorithm

In this section, we describe the homogeneous self-dual formulation and algorithm. Our implementation largely follows the algorithmic framework of [39] and [4]. Consequently, we focus on the algorithm's main components, and refer to [4, 39] for convergence proofs and theoretical results. Specific implementation details will be further discussed in Section 4.

3.1 Homogeneous self-dual embedding

The simplified homogeneous self-dual form was introduced in [39]. It consists in reformulating the primal-dual pair (1) as a single, self-dual linear program, which writes

$$
\begin{array}{llllllll}
(HSD) & \displaystyle\min_{x,y,s,\tau,\kappa} & 0 & & & & & \\
& s.t. & Ax & & & -b\tau & & = 0, \\
& & & A^T y & +s & -c\tau & & = 0, \\
& & -c^T x & +b^T y & & & -\kappa & = 0, \\
& & x, & & s, & \tau, & \kappa & \geq 0,
\end{array}
\tag{3}
$$

where τ and κ are two scalar variables. A solution (x, y, s, τ, κ) is strictly complementary if

$$x_i s_i = 0, x_i + s_i > 0, \text{ and } \tau\kappa = 0, \tau + \kappa > 0.$$

Problem (HSD) is always feasible, has empty interior and, under mild assumptions, possesses a strictly complementary feasible solution [39].

Let $(x^*, y^*, s^*, \tau^*, \kappa^*)$ be a strictly complementary feasible solution for (HSD). If $\tau^* > 0$, then $(\frac{x^*}{\tau^*}, \frac{y^*}{\tau^*}, \frac{s^*}{\tau^*})$ is an optimal solution for the original problem (1). Otherwise, we have $\kappa^* > 0$ and thus $c^T x^* - b^T y^* < 0$. In that case, the original problem (P) is infeasible or unbounded. If $c^T x^* < 0$, then (P) is unbounded and x^* is an unbounded ray. If $-b^T y^* < 0$, then (P) is infeasible and y^* is an unbounded dual ray. The latter is also referred to as a Farkas proof of infeasibility. Finally, if both $c^T x^* < 0$ and $-b^T y^* < 0$, then both (P) and (D) are infeasible.

3.2 Homogeneous interior-point algorithm

We now describe the homogeneous interior-point algorithm. Let (x, y, s, τ, κ) denote the current primal-dual iterate, with $(x, s, \tau, \kappa) > 0$. First, define the residuals

$$r_p = \tau b - Ax, \tag{4}$$
$$r_d = \tau c - A^T y - s, \tag{5}$$
$$r_g = c^T x - b^T y + \kappa, \tag{6}$$

and the barrier parameter

$$\mu = \frac{x^T s + \tau\kappa}{n+1}.$$

A search direction $(\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa)$ is computed by solving a Newton system of the form

$$A\delta_x - b\delta_\tau = \eta r_p, \tag{7}$$
$$A^T \delta_y + \delta_s - c\delta_\tau = \eta r_d, \tag{8}$$
$$-c^T \delta_x + b^T \delta_y - \delta_\kappa = \eta r_g, \tag{9}$$
$$S\delta_x + X\delta_s = -XSe + \gamma\mu e, \tag{10}$$
$$\kappa\delta_\tau + \tau\delta_\kappa = -\tau\kappa + \gamma\mu, \tag{11}$$

where γ and η are non-negative scalars. The new iterate is then

$$(x^+, y^+, s^+, \tau^+, \kappa^+) = (x, y, s, \tau, \kappa) + \alpha(\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa) \tag{12}$$

for some step-size $\alpha > 0$.

It then follows [4] that

$$(r_p^+, r_d^+, r_g^+) = (1 - \alpha\eta)(r_p, r_d, r_g), \tag{13}$$

and

$$(x^+)^T s^+ + \tau^+\kappa^+ = \left(1 - \alpha(1-\gamma) + \alpha^2\eta(1-\gamma-\eta)\right)(x^T s + \tau\kappa). \tag{14}$$

Consequently, if $\eta = 1 - \gamma$, then infeasibility and optimality gap are decreased by the same factor $(1 - \alpha\eta)$. This remarkable property contrasts with classical infeasible primal-dual algorithms such as [30], in which feasibility is often reached earlier than optimality.

3.2.1 Starting point

We choose the following default starting point

$$(x^0, y^0, s^0, \tau^0, \kappa^0) = (e, 0, e, 1, 1).$$

This initial point was proposed in [39]. Besides its simplicity, it has well-balanced complimentary products, which are all equal to one.

3.2.2 Search direction

At each iteration, a search direction is computed using Mehrotra's predictor-corrector technique [30], combined with Gondzio's multiple centrality corrections [18]. Following [4], we adapt the original formulas of [18, 30] to account for the homogeneous embedding.

First, the affine-scaling direction $(\delta_x^{\text{aff}}, \delta_y^{\text{aff}}, \delta_s^{\text{aff}}, \delta_\tau^{\text{aff}}, \delta_\kappa^{\text{aff}})$ is obtained by solving the Newton system

$$A\delta_x^{\text{aff}} - b\delta_\tau^{\text{aff}} = r_p, \tag{15}$$

$$A^T\delta_y^{\text{aff}} + \delta_s^{\text{aff}} - c\delta_\tau^{\text{aff}} = r_d, \tag{16}$$

$$-c^T\delta_x^{\text{aff}} + b^T\delta_y^{\text{aff}} - \delta_\kappa^{\text{aff}} = r_g, \tag{17}$$

$$S\delta_x^{\text{aff}} + X\delta_s^{\text{aff}} = -XSe, \tag{18}$$

$$\kappa\delta_\tau^{\text{aff}} + \tau\delta_\kappa^{\text{aff}} = -\tau\kappa, \tag{19}$$

which corresponds to (7)–(11) for $\eta = 1$ and $\gamma = 0$. Taking a full step ($\alpha = 1$) would thus reduce both infeasibility and complementary gap to zero. However, doing so is generally not possible, due to the non-negativity requirement on (x, s, τ, κ).

Consequently, a corrected search direction is computed, as proposed in [30]. The corrected direction hopefully enables one to make longer steps, thus reducing the total number of IPM iterations. Let $\eta = 1 - \gamma$, where

$$\gamma = (1 - \alpha^{\text{aff}})^2 \min\left(\beta_1, (1 - \alpha^{\text{aff}})\right) \tag{20}$$

for some $\beta_1 > 0$, and

$$\alpha^{\text{aff}} = \max\left\{0 \leq \alpha \leq 1 \mid (x, s, \tau, \kappa) + \alpha(\delta_x^{\text{aff}}, \delta_s^{\text{aff}}, \delta_\tau^{\text{aff}}, \delta_\kappa^{\text{aff}}) \geq 0\right\}. \tag{21}$$

The corrected search direction is then given by

$$\begin{aligned} A\delta_x - b\delta_\tau &= \eta r_p, && (22)\\ A^T\delta_y + \delta_s - c\delta_\tau &= \eta r_d, && (23)\\ -c^T\delta_x + b^T\delta_y - \delta_\kappa &= \eta r_g, && (24)\\ S\delta_x + X\delta_s &= -XSe + \gamma\mu e - \Delta_x^{\text{aff}}\Delta_s^{\text{aff}} e, && (25)\\ \kappa\delta_\tau + \tau\delta_\kappa &= -\tau\kappa + \gamma\mu - \delta_\tau^{\text{aff}}\delta_\kappa^{\text{aff}}, && (26) \end{aligned}$$

where $\Delta_x^{\text{aff}} = Diag(\delta_x^{\text{aff}})$ and $\Delta_s^{\text{aff}} = Diag(\delta_s^{\text{aff}})$.

Finally, additional corrections are computed. These corrections aim at improving the centrality of the new iterate, i.e., to keep the complementary products well-balanced. Doing so generally allows to make longer steps, thus reducing the total number of IPM iterations. We implement Gondzio's original technique [18], with some modifications introduced in [4].

Let $\delta = (\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa)$ be the current search direction, α^{max} the corresponding maximum step size, and define

$$(\bar{x}, \bar{y}, \bar{s}, \bar{\tau}, \bar{\kappa}) := (x, y, s, \tau, \kappa) + \bar{\alpha}(\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa), \tag{27}$$

where $\bar{\alpha} := \min(1, 2\alpha^{max})$ is a tentative step size.

First, a soft target in the space of complimentary products is computed as

$$t_j = \begin{cases} \mu_l - \bar{x}_j\bar{s}_j & \text{if } \bar{x}_j\bar{s}_j < \mu_l \\ 0 & \text{if } \bar{x}_j\bar{s}_j \in [\mu_l, \mu_u] \\ \mu_u - \bar{x}_j\bar{s}_j & \text{if } \bar{x}_j\bar{s}_j > \mu_u \end{cases}, \quad j = 1, \dots, n, \tag{28}$$

$$t_0 = \begin{cases} \mu_l - \bar{\tau}\bar{\kappa} & \text{if } \bar{\tau}\bar{\kappa} < \mu_l \\ 0 & \text{if } \bar{\tau}\bar{\kappa} \in [\mu_l, \mu_u] \\ \mu_u - \bar{\tau}\bar{\kappa} & \text{if } \bar{\tau}\bar{\kappa} > \mu_u \end{cases}, \tag{29}$$

where $\mu_l = \gamma\mu\beta_4$ and $\mu_u = \gamma\mu\beta_4^{-1}$, for a fixed $0 < \beta_4 \leq 1$. Then, define

$$v = t - \frac{e^T t + t_0}{n+1} e, \tag{30}$$

$$v_0 = t_0 - \frac{e^T t + t_0}{n+1}. \tag{31}$$

A correction is obtained by solving the linear system

$$\begin{aligned} A\delta_x^c - b\delta_\tau^c &= 0, && (32)\\ A^T\delta_y^c + \delta_s^c - c\delta_\tau^c &= 0, && (33)\\ -c^T\delta_x^c + b^T\delta_y^c - \delta_\kappa^c &= 0, && (34)\\ S\delta_x^c + X\delta_s^c &= v, && (35)\\ \kappa\delta_\tau^c + \tau\delta_\kappa^c &= v_0, && (36) \end{aligned}$$

which yields a corrected search direction

$$(\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa) + (\delta_x^c, \delta_y^c, \delta_s^c, \delta_\tau^c, \delta_\kappa^c).$$

The corrected direction is accepted if it results in an increased step size.

Finally, additional centrality corrections are computed only if a sufficient increase in the step size is observed. Specifically, as suggested in [4], an additional correction is computed only if the new step size α satisfies

$$\alpha \geq 1.10 \times \alpha^{\max}. \tag{37}$$

3.2.3 Resolution of the Newton system

Search directions and centrality corrections are obtained by solving several Newton systems such as (7)–(11), all with identical left-hand side matrix but different right-hand side. Each Newton system has the form

$$\begin{bmatrix} A & & & -b & \\ & A^T & I & -c & \\ -c^T & b^T & & & -1 \\ S & & X & & \\ & & & \kappa & \tau \end{bmatrix} \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_s \\ \delta_\tau \\ \delta_\kappa \end{bmatrix} = \begin{bmatrix} \xi_p \\ \xi_d \\ \xi_g \\ \xi_{xs} \\ \xi_{\tau\kappa} \end{bmatrix}, \tag{38}$$

where $\xi_p, \xi_d, \xi_g, \xi_{xs}, \xi_{\tau\kappa}$ are given right-hand side vectors.

First, we eliminate δ_s and δ_κ as follows:

$$\delta_s = X^{-1}(\xi_{xs} - S\delta_x), \tag{39}$$
$$\delta_\kappa = \tau^{-1}(\xi_{\tau\kappa} - \kappa\delta_\tau), \tag{40}$$

which yields

$$\begin{bmatrix} -\Theta^{-1} & A^T & -c \\ A & & -b \\ -c^T & b^T & \tau^{-1}\kappa \end{bmatrix} \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_\tau \end{bmatrix} = \begin{bmatrix} \xi_d - X^{-1}\xi_{xs} \\ \xi_p \\ \xi_g + \tau^{-1}\xi_{\tau\kappa} \end{bmatrix}, \tag{41}$$

where $\Theta = XS^{-1}$.

A solution to the reduced system (41) is obtained by first solving the two augmented systems

$$\begin{bmatrix} -\Theta^{-1} & A^T \\ A & \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} c \\ b \end{bmatrix}, \tag{42}$$

and

$$\begin{bmatrix} -\Theta^{-1} & A^T \\ A & \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \xi_d - X^{-1}\xi_{xs} \\ \xi_p \end{bmatrix}. \tag{43}$$

Then, $\delta_x, \delta_y, \delta_\tau$ are computed as follows:

$$\delta_\tau = \frac{\xi_g + \tau^{-1}\xi_{\tau\kappa} + c^T u + b^T v}{\tau^{-1}\kappa - c^T p + b^T q}, \tag{44}$$
$$\delta_x = u + \delta_\tau p, \tag{45}$$
$$\delta_y = v + \delta_\tau q. \tag{46}$$

Finally, the augmented systems (42) and (43) are first reduced to the normal equations system, by pivoting out the diagonal block $-\Theta^{-1}$. In the case of (43), the normal equations write

$$(A\Theta A^T)v = \xi_p + A\Theta^{-1}(\xi_d - X^{-1}\xi_{xs}). \tag{47}$$

One then recovers $u = \Theta^{-1}(A^T v - \xi_d + X^{-1}\xi_{xs})$.

The normal equations are typically solved by a direct method, namely by computing a Cholesky factorization of the positive definite matrix $A\Theta A^T$. This factorization is computed only once per iteration, and is re-used in subsequent solves. Further details on the practical computation of the Cholesky factors will be given in Section 4 and Section 5.

3.2.4 Step size

Once the final search direction has been computed, the step size α is given by

$$\alpha = 0.99995 \times \alpha^{max}, \tag{48}$$

where

$$\alpha^{max} = \max \left\{ 0 \leq \alpha \leq 1 \mid (x, s, \tau, \kappa) + \alpha(\delta_x, \delta_s, \delta_\tau, \delta_\kappa) \geq 0 \right\}.$$

3.2.5 Stopping criteria

The algorithm stops when, up to numerical tolerances, one of the following three cases holds: the current iterate is optimal, the primal problem is proven infeasible, the dual problem is proven infeasible (unbounded primal).

The problem is declared solved to optimality if

$$\frac{|r_p|}{\tau(1+|b|)} < \varepsilon_p, \tag{49}$$

$$\frac{|r_d|}{\tau(1+|c|)} < \varepsilon_d, \tag{50}$$

$$\frac{|c^T x - b^T y|}{\tau + |b^T y|} < \varepsilon_g, \tag{51}$$

where $\varepsilon_p, \varepsilon_d, \varepsilon_g$ are positive parameters. The above criteria are independent of the magnitude of τ, and correspond to primal feasibility, dual feasibility and optimality, respectively.

Primal or dual infeasibility is detected if

$$\mu < \varepsilon_i, \tag{52}$$

$$\frac{\tau}{\kappa} < \varepsilon_i, \tag{53}$$

where ε_i is a positive parameter. When this is the case, a complimentary solution with small τ has been found. If $c^T x < -\varepsilon_i$, the problem is declared dual infeasible (primal unbounded), and x is an unbounded ray. If $-b^T y < -\varepsilon_i$, the problem is declared primal infeasible (dual unbounded), and y is a Farkas dual ray.

Finally, premature termination criteria such as numerical instability, time limit or iteration limit are discussed in Section 4.

4 Implementation details

Our interior-point solver, Tulip, is written in Julia 1.0 [8], and is publicly available[2] under an open-source license. We did not implement any presolve nor crossover procedure. The code is single-threaded, however external linear algebra libraries may exploit mutliple threads.

4.1 Bounds on variables

Free variables are an outstanding issue for interior-point methods, see, e.g. [5,38], and are not supported explicitly in Tulip. Instead, free variables are automatically split into the difference of two non-negative variables, with the knowledge that this reformulation may introduce some numerical instability.

[2]`https://github.com/ds4dm/Tulip.jl`

Although finite upper bounds may be treated as arbitrary constraints, it is more efficient to handle them separately. Let $\mathcal{I}$ denote the set of indices of upper-bounded variables. Upper-bound constraints then write

$$x_i \leq u_i, \quad \forall i \in \mathcal{I}, \tag{54}$$

which we write in compact form $Ux \leq u$, where $U \in \mathbb{R}^{|\mathcal{I}| \times n}$ and

$$U_{i,j} = \left\{ \begin{array}{ll} 1 & \text{if } i = j \in \mathcal{I} \\ 0 & \text{otherwise} \end{array} \right. .$$

Therefore, Tulip considers linear programs of the form

$$\begin{array}{llll} (P) & \displaystyle\min_{x,w} & c^T x \\ & s.t. & Ax = b, \\ & & Ux + w = u \\ & & x, w \geq 0, \end{array} \qquad \begin{array}{llll} (D) & \displaystyle\max_{y,s,z} & b^T y - u^T z \\ & s.t. & A^T y + s - U^T z = c, \\ & & s, z \geq 0. \\ & & \end{array} \tag{55}$$

Let us emphasize that handling upper bounds separately only affects the underlying linear algebra operations, not the interior-point algorithm.

The Newton system (38) thus writes

$$\left[\begin{array}{ccccccc} A & & & & & -b & \\ U & I & & & & -u & \\ & & A^T & I & -U^T & -c & \\ -c^T & & b^T & & -u^T & & -1 \\ S & & & X & & & \\ & Z & & & W & & \\ & & & & & \kappa & \tau \end{array} \right] \left[\begin{array}{c} \delta_x \\ \delta_w \\ \delta_y \\ \delta_s \\ \delta_z \\ \delta_\tau \\ \delta_\kappa \end{array} \right] = \left[\begin{array}{c} \xi_p \\ \xi_u \\ \xi_d \\ \xi_g \\ \xi_{xs} \\ \xi_{wz} \\ \xi_{\tau\kappa} \end{array} \right], \tag{56}$$

and it reduces, after performing diagonal substitutions, to solving two augmented systems of the form

$$\left[\begin{array}{cc} -\tilde{\Theta}^{-1} & A^T \\ A & \end{array} \right] \left[\begin{array}{c} p \\ q \end{array} \right] = \left[\begin{array}{c} \tilde{\xi}_d \\ \tilde{\xi}_p \end{array} \right], \tag{57}$$

where $\tilde{\Theta} = \left(X^{-1}S + U^T(W^{-1}Z)U \right)^{-1}$. Note that $\tilde{\Theta}$ is a diagonal matrix with positive diagonal. Therefore, system (57) has the same size and structure as (42). Furthermore, $\tilde{\Theta}$ can be computed efficiently using only vector operations, i.e., without any matrix-matrix nor matrix-vector product.

4.2 Solver parameters

The default values for numerical tolerances of Section 3.2.5 are

$$\begin{aligned} \varepsilon_p &= 10^{-8}, \\ \varepsilon_d &= 10^{-8}, \\ \varepsilon_g &= 10^{-8}, \\ \varepsilon_i &= 10^{-8}. \end{aligned}$$

When computing additional centrality corrections, we use the following default values:

$$\begin{aligned} \beta_1 &= 10^{-1}, \\ \beta_2 &= 10^{-1}. \end{aligned}$$

The default maximum number of centrality corrections is set to 5.

Finally, the maximum number of IPM iterations is set to a default of 100. A time limit may be imposed by the user, in which case it is checked at the beginning of each IPM iteration.

4.3 Linear algebra

Our implementation leverages Julia's multiple dispatch feature and built-in support for linear algebra, thus allowing to disentangle the algorithmic framework from the linear algebra implementation.

First, the interior-point algorithm is defined over abstract linear algebra structures. Namely, the constraint matrix A is always declared as an `AbstractMatrix`, whose concrete type is only known once the model is instantiated. Julia's standard library includes extensive support for linear algebra, thus removing the need for a custom abstract linear algebra layer.

Then, specialized methods are automatically dispatched based on the (dynamic) type of A. These include matrix-vector and matrix-matrix product, as well as matrix factorization routines. We emphasize that the dispatch feature is a core component of the Julia programming language, and is therefore entirely transparent to the user. Consequently, one can easily define custom routines that exploit certain properties of A, so as to speed-up computation or reduce memory overheads. Such an example is given in Section 5. Furthermore, this customization is entirely independent of the interior-point algorithm, thus allowing to properly assess the impact of different linear algebra implementations.

Finally, for general dense and sparse matrices, the following libraries are used by default. Dense linear algebra operations, such as dense matrix-matrix and matrix-vector products, use BLAS (OpenBlas) routines. Dense factorizations use LAPACK routines. Sparse linear algebra operations, such as matrix-matrix and matrix-vector products, use Julia's standard library `SparseArrays`. Sparse factorizations use the CHOLMOD module of SuiteSparse [14].

4.4 Numerical stabilization

Numerical stability is a well-known source of computational issues in interior-point methods. As the iterates converge towards an optimal solution, the left-hand matrix of the normal equations typically becomes ill-conditioned. Combined with the limited accuracy of floating-point arithmetic, one may therefore encounter small or slightly negative pivots during the factorization phase. Small of negative pivots may be handled directly by the linear algebra library, in which case it is generally transparent to the user. Alternatively, one may introduce a regularization term that improves the numerical behavior, however this typically requires a tight integration with the factorization routine.

Consequently, we implemented the simple following regularization procedure. If the factorization of $A\Theta A^T$ fails, a static diagonal term of $10^{-6} \times I$ is added. A second factorization of this regularized matrix is then attempted. If this second factorization fails, an error is thrown, leading to the premature termination of the algorithm.

We did not implement any automatic scaling of the problem data, nor any presolve procedure. These possibilities are left for future work.

5 Specialized linear algebra

In this section, we present a specialized linear algebra implementation that exploits the intrinsic structure of the master problem in a Dantzig-Wolfe decomposition. In Section 5.1, we introduce the Dantzig-Wolfe decomposition principle, together with the column-generation method. For a thorough overview of column-generation methods and their use within decomposition frameworks, we refer to [15]. A specialized Cholesky factorization is described in Section 5.2. Computational results will be presented in Section 6.2.

5.1 Dantzig-Wolfe decomposition and column generation

We consider a mixed-integer linear program of the form

$$(MILP) \quad \min_{x_1,\ldots,x_R} \quad \sum_{r=1}^{R} c_r^T x_r \tag{58}$$

$$s.t. \quad \sum_{r=1}^{R} A_r x_r = b_0, \tag{59}$$

$$x_r \in \mathcal{X}_r \quad \forall r, \tag{60}$$

where $R > 0$, $c_r, x_r \in \mathbb{R}^{n_r}$, $b_0 \in \mathbb{R}^{m_0}$, $A_r \in \mathbb{R}^{m_0 \times n_r}$ and, for each r, $\mathcal{X}_r \subset \mathbb{R}^{n_r}$ is defined by a finite number of linear constraints plus integrality requirements on some coordinates of x_r. For ease of reading, we assume that $\mathcal{X}_r$ is bounded, and denote Ω_r the (finite) set of its extreme points. Therefore, we have

$$conv(\mathcal{X}_r) = conv(\Omega_r) = \left\{ \sum_{\omega \in \Omega_r} \lambda_\omega \omega \;\middle|\; e^T \lambda = 1, \lambda \geq 0 \right\}. \tag{61}$$

The Dantzig-Wolfe decomposition principle [13] then consists in replacing x_r by a convex combination of extreme vertices $\{\omega\}_{\omega \in \Omega_r}$. This change of variable yields the extended *Master Problem*

$$(MP) \quad \min_{\lambda} \quad \sum_{r, \omega \in \Omega_r} c_{r,\omega} \lambda_{r,\omega} \tag{62}$$

$$s.t. \quad \sum_{\omega \in \Omega_r} \lambda_{r,\omega} = 1, \quad \forall r \tag{63}$$

$$\sum_{r, \omega \in \Omega_r} a_{r,\omega} \lambda_{r,\omega} = b_0, \tag{64}$$

$$\lambda \geq 0, \tag{65}$$

where $c_{r,\omega} = c_r^T \omega$ and $a_{r,\omega} = A_r \omega$. The Master Problem (62)–(65) contains $m_0 + R$ constraints and $N = \sum_r |\Omega_r|$ variables, with N exponentially large in general. Therefore, (MP) is typically solved by column generation, wherein only a small subset of the variables are considered. Additional variables are generated iteratively by solving an auxiliary pricing problem. For the case at hand, we focus on the resolution of (MP), i.e., we only consider the root node in a branch-and-price algorithm. Furthermore, we consider the case where R is large compared to the number of linking constraints (64), and the vectors $a_{r,\omega}$ are dense. This latter assumption will be discussed in Section 6.2.

For each r, let $\bar{\Omega}_r$ be a (small) subset of Ω_r. The *Restricted Master Problem* is then defined as

$$(RMP) \quad \min_{\lambda} \quad \sum_{r, \omega \in \bar{\Omega}_r} c_{r,\omega} \lambda_{r,\omega} \tag{66}$$

$$s.t. \quad \sum_{\omega \in \bar{\Omega}_r} \lambda_{r,\omega} = 1, \quad \forall r \qquad [\sigma_r] \tag{67}$$

$$\sum_{r, \omega \in \bar{\Omega}_r} a_{r,\omega} \lambda_{r,\omega} = b_0, \qquad [\pi] \tag{68}$$

$$\lambda \geq 0. \tag{69}$$

Dual variables are indicated in brackets for each constraint. In all that follows, we assume that (RMP) is feasible. If necessary, one can enforce feasibility by adding artificial variables [15].

At the beginning of each iteration, (RMP) is solved to optimality, yielding a dual solution (π, σ). Then, for each r, a variable $\lambda_{r,\omega}$ with most negative reduced cost can be obtained by solving the pricing subproblem

$$(SP_r) \quad \min_{x_r} \quad (c_r^T - \pi^T A)x_r - \sigma_r \tag{70}$$

$$s.t. \quad x_r \in \mathcal{X}_r. \tag{71}$$

Note that (SP_r) is a mixed-integer linear program, which we assume can be solved efficiently. Let z^* and ω^* denote the optimal value and an optimal solution of (SP_r), respectively. If $z^* < 0$, then the corresponding variable λ_{r,ω^*} has negative reduced cost and it is added to the restricted master problem. Otherwise, all variables $\{\lambda_{r,\omega}\}_{\omega \in \Omega_r}$ have non-negative reduced cost. The column-generation algorithm continues until no variable with negative reduced cost can be identified among all R subproblems. When this is the case, the current solution of (RMP) is also optimal for (MP), and the algorithm stops. A basic column-generation procedure is formally stated in Algorithm 1.

Algorithm 1 Column-generation procedure

```
Input: Initial $(RMP)$
 1: while stopping criterion not met do
 2:     Solve $(RMP)$ and obtain optimal dual variables $(\pi, \sigma)$
 3:     // Pricing step
 4:     for all $r \in \{1, \dots, R\}$ do
 5:         Solve $SP_r$ with the query point $(\pi, \sigma_r)$; obtain $\omega^*$
 6:         if $\bar{c}_{r,\omega^*} < 0$ then
 7:             Add corresponding column to $(RMP)$
 8:         end if
 9:     end for
10:     // Stopping criterion
11:     if no column added to $(RMP)$ then
12:         STOP
13:     end if
14: end while
```

5.2 Specialized Cholesky factorization

We now describe a specialized Cholesky factorization that exploits the block structure of the master problem. For ease of reading, and noting that (MP) and (RMP) have identical block structures, in what follows, we consider the case of (MP).

First, the constraint matrix of (MP) is unit block-angular, i.e., it has the form

$$A = \begin{bmatrix} e^T & & \\ & \ddots & \\ & & e^T \\ \tilde{A}_1 & \cdots & \tilde{A}_R \end{bmatrix}, \tag{72}$$

where each dense lower block $\tilde{A}_r$ is given by

$$\tilde{A}_r = \begin{pmatrix} | & & | \\ a_{r,\omega_1} & \dots & a_{r,\omega_{|\Omega_r|}} \\ | & & | \end{pmatrix} \in \mathbb{R}^{m_0 \times |\Omega_r|}. \tag{73}$$

Let us recall that the normal equations system writes

$$(A\Theta A^T)v = \xi,$$

where $v \in \mathbb{R}^{m_0}$, and $\Theta \in \mathbb{R}^{N\times N}$ is a diagonal matrix with positive diagonal. In particular, Θ can be partitioned in blocks corresponding to the lower blocks of A, i.e.,

$$\Theta = \begin{pmatrix} \Theta_1 & & \\ & \ddots & \\ & & \Theta_R \end{pmatrix}, \tag{74}$$

with $\Theta_r = Diag(\theta_r)$ and $\theta_r > 0$. Consequently, the normal equations system has the form

$$\begin{bmatrix} e^T\theta_1 & & & (\tilde{A}_1\theta_1)^T \\ & \ddots & & \vdots \\ & & e^T\theta_R & (\tilde{A}_R\theta_R)^T \\ \tilde{A}_1\theta_1 & \cdots & \tilde{A}_R\theta_R & \Phi \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_R \\ v_0 \end{bmatrix} = \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_R \\ \xi_0 \end{bmatrix}, \tag{75}$$

where $v_1, \dots, v_R \in \mathbb{R}$, $v_0 \in \mathbb{R}^{m_0}$, and $\Phi = \sum_r \tilde{A}_r \Theta_r \tilde{A}_r^T \in \mathbb{R}^{m_0\times m_0}$.

Then, define

$$l_r = \frac{1}{e^T\theta_r}\tilde{A}_r\theta_r \in \mathbb{R}^{m_0} \tag{76}$$

and

$$C = \Phi - \sum_{r=1}^{R} \frac{1}{e^T\theta_r}(\tilde{A}_r\theta_r)(\tilde{A}_r\theta_r)^T \in \mathbb{R}^{m_0\times m_0}. \tag{77}$$

Given that both $A\Theta A^T$ and its upper-left block are positive definite, so is the Schur complement C. Therefore, its Cholesky factorization exists, which we denote $C = L_C D_C L_C^T$. It then follows that a Cholesky factorization of $A\Theta A^T$ is given by

$$A\Theta A^T = \underbrace{\begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ l_1 & \cdots & l_R & L_C \end{bmatrix}}_{L} \times \underbrace{\begin{bmatrix} e^T\theta_1 & & & \\ & \ddots & & \\ & & e^T\theta_R & \\ & & & D_C \end{bmatrix}}_{D} \times \underbrace{\begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ l_1 & \cdots & l_R & L_C \end{bmatrix}^T}_{L^T}. \tag{78}$$

Finally, once the Cholesky factors L and D are computed, the normal equations (75) are solved as follows:

$$v_0 = (L_C D_C L_C^T)^{-1}\left(\xi_0 - \sum_{r=1}^{R} \xi_r l_r\right), \tag{79}$$

$$\forall r,\ v_r = \frac{1}{e^T\theta_r}\xi_r - l_r^T v_0. \tag{80}$$

Exploiting the structure of A yields several computational advantages. First, the factors L and D can be computed directly from A and Θ, i.e., the matrix $A\Theta A^T$ does not need to be explicitly formed nor stored, thus saving both time and memory. Second, the sparsity structure of L is known beforehand. Specifically, the lower blocks $l_1, \dots, l_R$ are all dense column vectors, and the Schur complement C is a dense $m_0 \times m_0$ matrix. Therefore, one does not need a preprocessing phase wherein a sparsity-preserving ordering is computed, thus saving time and making memory allocation fully known in advance. Third, since most heavy operations are performed on dense matrices, efficient cache-exploiting kernels for dense linear algebra can be used, further speeding-up the computations. Finally, note that most operations such as forming the Cholesky factors and performing the backward substitutions, are amenable to parallelization.

6 Computational results

The purpose of this section is twofold. First, in Section 6.1, we compare Tulip (with general-purpose sparse linear algebra) to several open-source and commercial interior-point solvers on the Netlib LP testset. Then, in Section 6.2, we demonstrate the effectiveness of using specialized linear algebra by benchmarking Tulip against state-of-the-art solvers, on a set of large-scale column-generation instances from [6].

6.1 Netlib LP testset

The performance of Tulip, with general-purpose sparse linear algebra, is compared to that of several open-source and commercial interior-point codes on the Netlib LP testset. We emphasize that our goal is not to perform a comprehensive benchmark of interior-point LP solvers. Rather, our purpose is to show that 1) Tulip with basic linear algebra can be competitive with open-source solvers on small LP instances, and 2) that commercial solvers significantly outperform open-source alternatives. The former indicates that Tulip has the potential to be used as a general-purpose LP solver, while the latter underlines the benefits of using specialized linear algebra.

Experiments are carried out on an Intel i5 6600-3.30GHz, 48GB RAM desktop computer running Linux. We select the following open-source and commercial solvers: Clp 1.16 [1], GLPK 4.64 [2], ECOS 2.0 [16], Ipopt 3.12 [36], CPLEX 12.9 (CPX) [26], Gurobi 8.0 (GRB) [25] and Mosek 8.1 (MSK) [33]. All are accessed through their respective Julia interface. We also include OOQP 0.99 [17]. Since OOQP does not have a Julia interface, it is called via its command line executable, namely `qpgen-sparse-gondzio`. This implements a primal-dual interior-point algorithm with Gondzio's multiple corrections, and uses general-purpose sparse linear algebra.

We run the interior-point algorithm of each solver on a single thread, with presolve and crossover deactivated, and a 300 s time limit. All other parameters are left to their default values, except Ipopt for which we set the parameter `mehrotra_algorithm` to `yes`, which is the recommended setting for solving LPs.[3] Scripts for running these experiments are available online.[4] OOQP's command-line executable does not allow to control presolve nor the number of threads. To the best of our knowledge, OOQP does not perform any presolve, and we observed that OOQP did use several threads (up to four, the maximum available on our machine). Finally, we note that stopping criteria and numerical tolerances vary among solvers, though not significantly enough to impact our conclusions.

We consider 114 instances from the Netlib LP testset: all 95 feasible Netlib instances, the three `qap8`, `qap12` and `qap15` instances, plus the 16 Kennington instances. This full testset is denoted N-114. In addition, we denote N-solved the set of instances that were reportedly solved to optimality by all solvers. Finally, we split N-solved into three sets of small, medium and large instances, respectively denoted N-small, N-medium and N-large. A given instance i is in N-small if

$$m_i + n_i < 10^3,$$

where m_i (resp. n_i) is the number of rows (resp. columns) of instance i. Similarly, instance i is in N-medium if

$$10^3 \leq m_i + n_i < 10^4,$$

and in N-large if

$$10^4 \leq m_i + n_i.$$

N-solved, N-small, N-medium and N-large comprise 72, 31, 33 and 8 instances, respectively.

[3]Ipopt documentation: https://www.coin-or.org/Ipopt/documentation/node46.html
[4]https://github.com/mtanneau/Netlib_experiments

Computational results are reported in Table 1 and Table 2 for open-source and commercial solvers, respectively. For each solver, we report the total number of instances solved, as well as average runtimes (in seconds) for each set of instances N-114, N-solved, N-small, N-medium and N-large. Average runtimes are computed using the geometric mean with a shift of 1. Failures are counted as timeouts, i.e., 300 s.

Table 1: Netlib testset - Open source solvers

		Clp	ECOS	GLPK	Ipopt	OOQP	Tulip
# Solved		114	106	92	108	103	94
Time (s)	N-114	0.490	1.155	2.924	1.778	1.819	2.420
	N-solved	0.143	0.143	0.352	0.430	0.306	0.134
	N-small	0.007	0.009	0.007	0.044	0.018	0.011
	N-medium	0.065	0.076	0.081	0.489	0.116	0.059
	N-large	1.505	1.366	9.628	3.088	5.560	1.352

Table 2: Netlib testset - Commercial solvers

		CPX	GRB	MSK
# Solved		114	113	113
Time (s)	N-114	0.115	0.229	0.262
	N-solved	0.029	0.039	0.045
	N-small	0.004	0.004	0.005
	N-medium	0.019	0.023	0.030
	N-large	0.173	0.268	0.288

First, observe that commercial solvers are overall more robust than open-source ones. Only one instance was reported as unsolved by Gurobi and Mosek, due to numerical stalling. Note that activating cross-over removes the failure for both solvers. Tulip solved 94 instances out of 114, and is thus less robust than other open-source solvers, to the exception of GLPK. The corresponding failures were caused either by small pivots encountered in the Cholesky factorization, or by a stalling behavior. Both are typically addressed via numerical stabilization, which is currently not implemented in Tulip.

Similarly, commercial solvers significantly outperform open-source ones in terms of computing times. Nevertheless, we find that Tulip is competitive, outperforming other open-source solvers on N-solved instances. Results for N-medium and N-large further indicate that the relative performance of Tulip improves on larger instances.

These results demonstrate that Tulip has the potential to be used as a general-purpose LP solver. Implementing the required numerical stabilization techniques is beyond the scope of this paper, and will be the subject of future work. Furthermore, Tulip is significantly outperformed by current commercial solvers when solving general LPs. Nevertheless, subsequent results will show that, combined with specialized linear algebra, Tulip achieves state-of-the-art performance on classes of structured problems despite its lack of numerical stabilization.

6.2 Column-generation instances

We now compare Tulip to state-of-the-art commercial solvers on a set of challenging problems that arise in power systems applications [6].

6.2.1 Column-generation algorithm

We implemented the column-generation algorithm of Section 5.1 in Julia. This implementation is publicly available under an open-source license.[5] Solvers' respective Julia APIs are accessed through

[5] https://github.com/ds4dm/Linda.jl

`MathProgBase` [3], a solver-agnostic interface for mathematical programming. We emphasize that implementing a state-of-the-art column-generation solver is beyond the scope of this paper. Therefore, we did not implement any stabilization technique, cleaning procedure, nor branching framework.

Both CPLEX and Gurobi are unable to return a Farkas ray for infeasible models when using their barrier algorithm with crossover turned off. Consequently, we follow the usual practice of adding artificial slack and/or surplus variables to each linking constraint in the master problem, thus ensuring it remains feasible. These artificial variables have large cost and effectively implement an ℓ_1 penalty on the violation of the linking constraints. Therefore, they take value zero as soon as a feasible solution for the Master Problem is found, which typically happens after a few iterations. Note that the presence of these artificial variables also enforces the full rank assumption on the constraint matrix of (RMP).

For large instances with numerous subproblems, full pricing, wherein all subproblems are solved at each iteration, is often not the most efficient approach. Therefore, we implemented a partial pricing strategy, in which subproblems are solved in a random order until either all subproblems have been solved, or a user-specified number of columns with negative reduced cost have been generated.

Finally, subproblems are solved as generic MILPs, and each subproblem is instantiated and solved from scratch every time it is called. The latter is to avoid storing the corresponding MILP instances (up to tens of thousands) in memory, thus keeping memory requirements acceptable for large instances. subproblems are solved to optimality, and at most one column per subproblem is generated at each pricing step.

6.2.2 Experimental setup

Each instance consists in coordinating the operation of R distributed energy resources over a time horizon of length T, see [6]. In the corresponding Dantzig-Wolfe decomposition, the master problem thus contains T linking constraints plus R convexity constraints, and there are R subproblems. Columns correspond to a resource's electricity consumption over the T time periods, and are structurally dense.

Our benchmark set comprises 180 instances of varying sizes. The number of subproblems, R, ranges from 2^{10} to 2^{15}. The number of linking constraints, T, ranges from 24 to 96. In order to mitigate performance variations, for each tuple (R, T), we generated 10 instances of the corresponding size. For a given tuple (R, T), performance metrics are therefore averaged over the corresponding 10 instances, using a geometric mean with a shift of 1. All experiments are performed on an Intel Xeon 6142-2.6GHz, 512GB RAM machine running Linux. Data and scripts for running these experiments are available online.[6]

Each instance is solved using the column-generation framework described in Section 6.2.1, with a time limit of $10,000$ seconds. The master problem is initialized with one column per resource, each obtained by solving the associated subproblem with $\pi = (0, \dots, 0)^T$. We compare the following six solvers: Tulip (TLP) with the specialized linear algebra of Section 5.2, the interior-point of Mosek 8.1 (MSK), the barrier (CPX) and primal simplex (SCPX) of CPLEX 12.8, and the barrier (GRB) and primal simplex (SGRB) of Gurobi 8.0. To ensure fair comparison, all solvers run on a single thread, with presolve deactivated and no crossover. Note that IPM solvers do not use any warm-start, i.e., (RMP) is solved from scratch at each column-generation iteration. Finally, all subproblems are solved with Gurobi 8.0, with a single thread and default parameters. In all that follows, we use the term "solver" to denote which solver (and algorithm) is used for solving the master problem.

6.2.3 Results

Computational results are presented in Table 3. We use the terms *inner iteration* and *outer iteration* to denote an iteration of an IPM algorithm (when solving (RMP)) and of the column-generation procedure, respectively. For each problem size and solver, we report the proportion of instances

[6] `https://github.com/mtanneau/DER_experiments`

solved to proven optimality within the time limit (% solved), the average number of outer and (when applicable) inner iterations, and average total computing times (Pricing, Master and Total). Since the time limit is checked at the beginning of each outer iteration, actual total times may slightly exceed the $10,000$ s limit.

Table 3: Column-generation statistics

Instance				Iterations		Time (s)		
T	R	Solver	% solved	Outer	Inner	Pricing	Master	Total
24	1,024							
		TLP	100	36.7	803.3	8.8	2.4	11.3
		MSK	100	35.7	682.9	9.1	3.7	12.8
		CPX	100	37.2	860.1	9.2	10.0	19.2
		GRB	100	37.0	645.1	9.4	4.8	14.3
		SCPX	100	36.0	-	9.2	1.2	10.4
		SGRB	100	35.8	-	9.2	1.2	10.5
24	2,048							
		TLP	100	36.6	868.0	18.3	5.1	23.7
		MSK	100	36.8	765.8	18.5	9.1	27.6
		CPX	100	36.9	968.8	18.8	9.3	28.1
		GRB	100	36.1	673.7	18.4	11.3	29.8
		SCPX	100	37.3	-	19.8	5.0	24.8
		SGRB	100	37.4	-	19.1	4.3	23.5
24	4,096							
		TLP	100	37.4	984.4	38.6	12.8	52.3
		MSK	100	37.8	820.4	39.3	22.5	62.0
		CPX	100	36.8	1,103.9	39.0	17.7	56.9
		GRB	100	37.2	751.6	38.8	28.0	67.0
		SCPX	100	37.8	-	39.9	18.3	58.4
		SGRB	100	37.4	-	39.2	16.0	55.4
24	8,192							
		TLP	100	37.4	1,068.3	78.4	31.8	113.0
		MSK	100	37.8	909.5	78.9	58.1	137.6
		CPX	100	36.8	1,216.8	79.0	46.3	125.9
		GRB	100	36.6	798.3	77.9	64.1	142.3
		SCPX	100	36.0	-	74.9	67.2	142.6
		SGRB	100	36.4	-	78.0	65.4	143.7
24	16,384							
		TLP	100	36.9	1,151.1	156.1	68.8	234.4
		MSK	100	38.6	985.1	174.0	136.5	311.7
		CPX	100	37.4	1,470.5	168.8	127.8	297.6
		GRB	100	37.8	951.5	168.6	169.7	339.1
		SCPX	100	37.6	-	163.3	249.1	413.8
		SGRB	100	37.4	-	165.7	274.9	441.5
24	32,768							
		TLP	100	37.6	1,243.6	340.7	159.0	535.0
		MSK	100	38.6	1,036.4	361.4	306.2	669.3
		CPX	100	38.6	1,738.9	354.3	332.1	688.5
		GRB	100	39.0	1,065.9	363.3	406.8	772.2
		SCPX	100	38.0	-	332.6	968.4	1,304.3
		SGRB	100	38.0	-	342.2	1,709.3	2,057.1
48	1,024							
		TLP	100	60.6	1,614.5	24.2	13.2	37.7
		MSK	100	60.8	1,320.9	24.1	20.9	45.1
		CPX	100	61.0	1,853.6	24.5	22.0	47.2
		GRB	100	61.0	1,200.4	24.9	38.2	63.3
		SCPX	100	59.2	-	24.3	6.5	31.0
		SGRB	100	60.4	-	24.6	5.7	30.4
48	2,048							
		TLP	100	57.3	1,710.9	45.9	28.5	74.9
		MSK	100	58.0	1,372.0	46.3	46.2	92.7
		CPX	100	58.7	2,106.2	48.6	110.8	159.7
		GRB	100	58.5	1,242.5	48.0	82.0	130.2
		SCPX	100	58.5	-	47.6	22.1	69.9
		SGRB	100	58.6	-	49.6	16.6	66.4

Table 3: (continued)

Instance		Solver	Solved	Iterations		Time (s)		
T	R		(%)	Outer	Inner	Pricing	Master	Total
48	4,096							
		TLP	100	57.6	1,994.3	97.0	75.4	174.0
		MSK	100	57.7	1,507.6	95.0	114.4	209.9
		CPX	100	57.8	2,486.3	96.1	130.0	226.5
		GRB	100	58.4	1,444.6	97.5	195.9	293.7
		SCPX	100	58.4	-	101.2	66.8	168.4
		SGRB	100	59.0	-	98.5	57.8	157.0
48	8,192							
		TLP	100	59.6	2,252.3	203.6	176.0	384.3
		MSK	100	57.9	1,631.7	200.2	262.4	463.3
		CPX	100	59.2	2,952.8	215.8	376.1	592.8
		GRB	100	58.1	1,582.1	206.1	446.5	653.3
		SCPX	100	57.8	-	203.1	188.4	392.1
		SGRB	100	59.1	-	210.5	225.5	439.9
48	16,384							
		TLP	100	58.9	2,411.2	424.1	387.9	828.1
		MSK	100	58.1	1,748.8	416.0	579.4	997.3
		CPX	100	56.9	3,143.6	416.1	792.4	1,210.8
		GRB	100	57.6	1,786.9	413.6	1,050.9	1,466.1
		SCPX	100	58.8	-	397.5	585.1	984.9
		SGRB	100	59.7	-	410.8	1,601.5	2,019.5
48	32,768							
		TLP	100	58.2	2,610.5	862.9	814.4	1,737.2
		MSK	100	58.2	1,823.1	876.6	1,258.0	2,139.2
		CPX	100	58.4	3,710.2	798.1	1,890.0	2,691.2
		GRB	100	58.0	1,977.1	826.4	2,411.7	3,241.0
		SCPX	100	58.6	-	797.6	2,389.7	3,191.5
		SGRB	0	55.6	-	654.6	9,563.3	10,220.3
96	1,024							
		TLP	100	102.8	3,140.9	77.9	98.7	177.1
		MSK	100	102.2	2,291.1	76.8	134.2	211.2
		CPX	100	102.6	3,848.4	78.9	202.5	281.9
		GRB	100	101.4	2,209.1	77.4	324.9	402.5
		SCPX	100	103.2	-	78.7	44.3	123.3
		SGRB	100	103.4	-	78.6	35.9	116.0
96	2,048							
		TLP	100	103.5	3,689.6	161.1	243.6	406.0
		MSK	100	102.8	2,602.7	158.1	312.4	471.0
		CPX	100	103.8	4,715.3	161.2	1,092.2	1,254.7
		GRB	100	103.1	2,598.3	159.5	831.4	991.3
		SCPX	100	103.2	-	158.4	108.5	267.4
		SGRB	100	106.8	-	165.1	148.1	323.1
96	4,096							
		TLP	100	104.4	4,291.8	331.1	555.4	890.1
		MSK	100	100.8	2,812.1	317.5	690.5	1,009.8
		CPX	100	101.8	5,515.2	312.1	2,046.1	2,360.9
		GRB	100	100.1	2,931.6	319.9	1,986.2	2,307.9
		SCPX	100	103.0	-	314.6	260.2	576.0
		SGRB	100	105.3	-	318.9	656.1	1,017.7
96	8,192							
		TLP	100	102.2	4,635.7	647.4	1,171.5	1,828.7
		MSK	100	103.0	3,162.2	652.1	1,628.3	2,283.3
		CPX	100	101.0	6,760.3	618.4	4,188.4	4,809.4
		GRB	100	100.6	3,319.6	624.4	4,455.1	5,081.2
		SCPX	100	101.7	-	616.6	707.0	1,326.0
		SGRB	100	104.0	-	641.4	3,012.1	3,666.9
96	16,384							
		TLP	100	105.0	5,430.7	1,348.3	2,780.9	4,162.8
		MSK	100	101.8	3,391.0	1,268.5	3,427.8	4,700.6
		CPX	20	98.9	7,322.1	1,121.1	8,975.7	10,101.5
		GRB	0	96.7	3,447.8	1,078.3	9,145.3	10,225.7
		SCPX	100	102.4	-	1,237.9	2,062.0	3,305.0
		SGRB	0	73.6	-	760.1	10,106.2	10,869.2

Table 3: (continued)

Instance		Solver	Solved	Iterations		Time (s)		
T	R		(%)	Outer	Inner	Pricing	Master	Total
96	32,768							
		TLP	80	102.4	5,738.6	2,697.6	6,091.5	8,909.4
		MSK	80	101.6	3,537.1	2,496.7	7,171.6	9,677.2
		CPX	0	70.5	5,010.0	1,461.4	8,657.2	10,123.6
		GRB	0	67.4	2,378.0	1,381.0	8,744.3	10,128.2
		SCPX	80	101.2	-	2,410.5	7,329.8	9,752.4
		SGRB	0	35.4	-	627.4	9,622.1	10,253.2

The results in Table 3 motivate the following remarks. First, using the simplex or an interior-point algorithm does not seem to affect the column-generation process much. Indeed, the number of outer iterations, as well as computing times for the pricing subproblems, are very similar between all solvers. This behavior most likely indicates that the instances at hand display little degeneracy. Then, the differences in total computing times are explained by the differences in solving time for the master problem, which directly reflects the performance of each solver. Simplex solvers (SCPX and SGRB) are typically faster on smaller instances, however interior-point solvers exhibit a more scalable behavior. This latter observation is in line with existing computational experience, see e.g. [31]. Since the performance of simplex solvers is not the primary scope of this work, we now focus on IPM solvers.

Tulip solves all but two of the largest instances within the time limit. This overall performance is on par with that of Mosek, while CPLEX's and Gurobi's barrier algorithms fail to solve large instances ($R = 96, T \geq 16{,}384$). Furthermore, Tulip consistently outperforms other interior-point solvers in terms of computing time for the master problem. This performance difference is quite significant, with typical speedups of a factor 1.5x to 4x. Nevertheless, Gurobi and Mosek typically require 30% to 50% fewer iterations than Tulip and CPLEX, respectively.

In Table 4, we report the average time per inner iteration, for each solver and problem size. Recall that the cost of an individual IPM iteration depends not only on problem size and the efficiency of the underlying linear algebra, but also on algorithmic features such as the number of corrections, which we cannot measured directly. Therefore, while we cannot assert that per-iteration times are solely explained by linear algebra implementations, this metric is a good proxy to assess the performance of a solver's internal linear algebra. We thus observe that, on a per-iteration basis, Tulip is 2x faster than Mosek, 3x to 4x faster than Gurobi, and up to 4x faster than CPLEX. These results strongly indicate that Tulip's linear algebra indeed outperforms that of commercial solvers, and demonstrate the benefits of using specialized linear algebra.

Overall, we observed that Tulip's behavior is as robust as that of commercial solvers, in part thanks to well-behaved numerics for the testset at hand. There exists significant room for further improving Tulip's performance, most notably through algorithmic enhancements that would reduce the number of IPM iterations. Finally, linear algebra is one of the lowest-level components of interior-point methods. Therefore, any algorithmic variation or improvement would benefit from highly efficient, specialized linear algebra.

7 Conclusion

In this paper, we have described the homogeneous self-dual interior-point algorithm and its implementation in Tulip, an open-source linear optimization solver. Most notably, the algorithmic framework in Tulip is fully disentangled from linear algebra implementations. The performance of the code has been evaluated on the Netlib LP testset, and on a set of structured instances for which we developed specialized linear algebra routines. Numerical results indicate that Tulip is competitive with open-source solvers on unstructured LP instances, and significantly outperforms state-of-the-art commercial codes

Table 4: Performance statistics IPM solvers

		Tulip		Mosek		CPLEX		Gurobi	
T	R	Iter.	Time / iter (s)	Iter.	Time / iter (s)	Iter.	Time / iter (s)	Iter.	Time / iter (s)
24	1,024	803	0.003	683	0.005	860	0.012	645	0.007
	2,048	868	0.006	766	0.012	969	0.010	674	0.017
	4,096	984	0.013	820	0.028	1,104	0.016	752	0.037
	8,192	1,068	0.030	910	0.065	1,217	0.039	798	0.082
	16,384	1,151	0.060	985	0.140	1,470	0.088	952	0.180
	32,768	1,244	0.129	1,036	0.297	1,739	0.192	1,066	0.382
48	1,024	1,614	0.008	1,321	0.016	1,854	0.012	1,200	0.032
	2,048	1,711	0.017	1,372	0.034	2,106	0.053	1,243	0.066
	4,096	1,994	0.038	1,508	0.077	2,486	0.052	1,445	0.136
	8,192	2,252	0.078	1,632	0.162	2,953	0.128	1,582	0.283
	16,384	2,411	0.162	1,749	0.333	3,144	0.253	1,787	0.589
	32,768	2,611	0.314	1,823	0.694	3,710	0.510	1,977	1.222
96	1,024	3,141	0.032	2,291	0.059	3,848	0.053	2,209	0.148
	2,048	3,690	0.066	2,603	0.120	4,715	0.235	2,598	0.321
	4,096	4,292	0.130	2,812	0.247	5,515	0.374	2,932	0.680
	8,192	4,636	0.253	3,162	0.518	6,760	0.621	3,320	1.343
	16,384	5,431	0.514	3,391	1.012	*7,322	*1.228	*3,448	*2.654
	32,768	5,739	1.064	3,537	2.030	*5,010	*1.729	*2,378	*3.678

* Premature termination due to time limit.

on structured problems. These results demonstrate the benefits of being able to seamlessly integrate specialized linear algebra within an interior-point algorithm.

Future work will include algorithmic enhancements such as pre-solve, scaling and numerical stabilization, so as to further improve both performance and robustness. Finally, one strength of IPMs is their ability to exploit parallelism. For instance, specialized solvers like OOPS and PIPS have been designed to exploit massively parallel architectures, to form and solve large-scale structured problems efficiently. Although our code is currently single-threaded, it can already take advantage of multi-threaded linear algebra libraries, thus carrying out heavy operations such as matrix-matrix products and factorization in parallel.

References

[1] CLP. URL https://projects.coin-or.org/Clp

[2] GNU Linear Programming Kit. URL https://www.gnu.org/software/glpk/

[3] MathProgBase. URL https://github.com/JuliaOpt/MathProgBase.jl

[4] Andersen, E.D., Andersen, K.D.: The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm, pp. 197–232. Springer US, Boston, MA (2000). DOI 10.1007/978-1-4757-3216-0_8. URL https://doi.org/10.1007/978-1-4757-3216-0_8

[5] Anjos, M., Burer, S.: On handling free variables in interior-point methods for conic linear optimization. SIAM Journal on Optimization 18(4), 1310–1325 (2008). DOI 10.1137/06066847X

[6] Anjos, M.F., Lodi, A., Tanneau, M.: A decentralized framework for the optimal coordination of distributed energy resources. IEEE Transactions on Power Systems 34(1), 349–359 (2019). DOI 10.1109/TPWRS.2018.2867476

[7] Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik 4(1), 238–252 (1962). DOI 10.1007/BF01386316

[8] Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. SIAM Review 59(1), 65–98 (2017). DOI 10.1137/141000671

[9] Birge, J.R., Qi, L.: Computing block-angular karmarkar projections with applications to stochastic programming. Management Science 34(12), 1472–1479 (1988). URL http://www.jstor.org/stable/2632037

[10] Bixby, R.E., Gregory, J.W., Lustig, I.J., Marsten, R.E., Shanno, D.F.: Very large-scale linear programming: A case study in combining interior point and simplex methods. Operations Research 40(5), 885–897 (1992). DOI 10.1287/opre.40.5.885

[11] Castro, J., Nasini, S., Saldanha-da Gama, F.: A cutting-plane approach for large-scale capacitated multi-period facility location using a specialized interior-point method. Mathematical Programming 163(1), 411–444 (2017). DOI 10.1007/s10107-016-1067-6

[12] Choi, I.C., Goldfarb, D.: Exploiting special structure in a primal—dual path-following algorithm. Mathematical Programming 58(1), 33–52 (1993). DOI 10.1007/BF01581258. URL https://doi.org/10.1007/BF01581258

[13] Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. Operations Research 8(1), 101–111 (1960). DOI 10.1287/opre.8.1.101

[14] Davis, T.A.: SuiteSparse: A suite of sparse matrix software. URL http://faculty.cse.tamu.edu/davis/suitesparse.html

[15] Desaulniers, G., Desrosiers, J., Solomon, M.M.: Column generation, *GERAD 25th anniversary*, vol. 5, 1 edn. Springer Science & Business Media (2006)

[16] Domahidi, A., Chu, E., Boyd, S.: ECOS: An SOCP solver for embedded systems. In: European Control Conference (ECC), pp. 3071–3076 (2013)

[17] Gertz, E.M., Wright, S.J.: Object-oriented software for quadratic programming. ACM Trans. Math. Softw. 29(1), 58–81 (2003). DOI 10.1145/641876.641880

[18] Gondzio, J.: Multiple centrality corrections in a primal-dual method for linear programming. Computational Optimization and Applications 6(2), 137–156 (1996). DOI 10.1007/BF00249643

[19] Gondzio, J.: Interior point methods 25 years later. European Journal of Operational Research 218(3), 587–601 (2012). DOI 10.1016/j.ejor.2011.09.017

[20] Gondzio, J., Gonzalez-Brevis, P., Munari, P.: New developments in the primal-dual column generation technique. European Journal of Operational Research 224(1), 41–51 (2013). DOI 10.1016/j.ejor.2012.07.024

[21] Gondzio, J., Grothey, A.: Exploiting structure in parallel implementation of interior point methods for optimization. Computational Management Science 6(2), 135–160 (2009). DOI 10.1007/s10287-008-0090-3

[22] Gondzio, J., Sarkissian, R.: Column generation with a primal-dual method. Tech. rep., Technical report 96.6, Logilab (1996). URL http://www.maths.ed.ac.uk/~gondzio/reports/pdcgm.pdf

[23] Gondzio, J., Sarkissian, R.: Parallel interior-point solver for structured linear programs. Mathematical Programming 96(3), 561–584 (2003). DOI 10.1007/s10107-003-0379-5

[24] Gondzio, J., Sarkissian, R., Vial, J.P.: Using an interior point method for the master problem in a decomposition approach. European Journal of Operational Research 101(3), 577–587 (1997). DOI 10.1016/S0377-2217(96)00182-8

[25] Gurobi Optimization, L.: Gurobi optimizer reference manual (2018). URL http://www.gurobi.com

[26] IBM: IBM ILOG CPLEX Optimization Studio. URL https://www.ibm.com/products/ilog-cplex-optimization-studio

[27] Jessup, E.R., Yang, D., Zenios, S.A.: Parallel factorization of structured matrices arising in stochastic programming. SIAM Journal on Optimization 4(4), 833–846 (1994). DOI 10.1137/0804048. URL https://doi.org/10.1137/0804048

[28] Kelley Jr., J.: The cutting-plane method for solving convex programs. Journal of the Society for Industrial and Applied Mathematics 8(4), 703–712 (1960). DOI 10.1137/0108053

[29] Lubin, M., Petra, C.G., Anitescu, M., Zavala, V.: Scalable stochastic optimization of complex energy systems. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, pp. 64:1–64:64. ACM, New York, NY, USA (2011). DOI 10.1145/2063384.2063470

[30] Mehrotra, S.: On the implementation of a primal-dual interior point method. SIAM Journal on Optimization 2(4), 575–601 (1992). DOI 10.1137/0802028. URL https://doi.org/10.1137/0802028

[31] Mitchell, J.E.: Cutting Plane Methods and Subgradient Methods, chap. Chapter 2, pp. 34–61 (2009). DOI 10.1287/educ.1090.0064

[32] Mitchell, J.E., Borchers, B.: Solving Linear Ordering Problems with a Combined Interior Point/Simplex Cutting Plane Algorithm, pp. 349–366. Springer US, Boston, MA (2000). DOI 10.1007/978-1-4757-3216-0_14. URL https://doi.org/10.1007/978-1-4757-3216-0_14

[33] MOSEK ApS: The MOSEK Optimization Suite. URL https://www.mosek.com/

[34] Rousseau, L.M., Gendreau, M., Feillet, D.: Interior point stabilization for column generation. Operations Research Letters 35(5), 660–668 (2007). DOI https://doi.org/10.1016/j.orl.2006.11.004

[35] Tanneau, M.: Tulip.jl. URL https://github.com/ds4dm/Tulip.jl

[36] Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106(1), 25–57 (2006). DOI 10.1007/s10107-004-0559-y. URL https://doi.org/10.1007/s10107-004-0559-y

[37] Westerlund, T., Pettersson, F.: An extended cutting plane method for solving convex minlp problems. Computers & Chemical Engineering 19, 131–136 (1995). DOI https://doi.org/10.1016/0098-1354(95)87027-X. URL http://www.sciencedirect.com/science/article/pii/009813549587027X

[38] Wright, S.: Primal-Dual Interior-Point Methods. Society for Industrial and Applied Mathematics (1997). DOI 10.1137/1.9781611971453

[39] Xu, X., Hung, P.F., Ye, Y.: A simplified homogeneous and self-dual linear programming algorithm and its implementation. Annals of Operations Research 62(1), 151–171 (1996). DOI 10.1007/BF02206815. URL https://doi.org/10.1007/BF02206815