A Linear Programming Method Based on Proximal-Point Iterations with Applications to Multi-Parametric Programming

Daniel Arnström, Alberto Bemporad and Daniel Axehill

***Abstract*— We propose a linear programming method that is based on active-set changes and proximal-point iterations. The method solves a sequence of least-distance problems using a warm-started quadratic programming solver that can reuse internal matrix factorizations from the previously solved least-distance problem. We show that the proposed method terminates in a finite number of iterations and that it outperforms state-of-the-art LP solvers in scenarios where an extensive number of small/medium scale LPs need to be solved rapidly, occurring in, for example, multi-parametric programming algorithms. In particular, we show how the proposed method can accelerate operations such as redundancy removal, computation of Chebyshev centers and solving linear feasibility problems.**

***Index Terms*— Optimization algorithms, predictive control for linear systems.**

I. Introduction

In this letter we are interested in the classical problem of finding solutions x^* to linear programs (LPs) in the form

$$\begin{aligned} x^* = \underset{x}{\operatorname{argmin}} \quad & f^T x \\ \text{s.t.} \quad & Ax \leq b, \end{aligned} \tag{1}$$

where $x, f \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. The need for solving such problems in various engineering applications is ubiquitous, and herein we are in particular interested in scenarios where an extensive number of small/medium scale LPs must be solved rapidly. Such scenarios arise in numerous control applications, for example, in real-time model predictive control (MPC) when the 1- or ∞-norm is used in the objective function [1][2, Ch.9] and in multi-parametric programming algorithms [3]–[10], used to, for example, compute explicit control laws [11] and control invariant sets [12]. In such scenarios, state-of-the-art LP solvers such as CPLEX and Gurobi can yield, relatively, extensive execution times since they are developed with challenging, large-scale, LPs in mind.

The method that we propose solves (1) by performing proximal-point iterations [13], [14], where each such iteration corresponds to solving a perturbed least-distance problem (LDP). We show how these LDPs can be solved efficiently with the dual active-set quadratic programming (QP) method proposed in [15], which has proven to be efficient for solving small/medium size QPs arising in real-time MPC. Moreover, the warm-starting capabilities (i.e., possibility of reusing previous solutions as good initializations) of the QP method in [15] makes the proposed LP method competitive with state-of-the-art implementations of the simplex method [16] and interior-point methods [17].

The proposed LP method is similar to the simplex method in that it is an active-set method. There are, however, three important differences between the simplex method and the proposed method: (i) The linear system of equations to be solved at each iteration of the proposed algorithm is symmetric, allowing an LDL^T factorization to be used and updated. In contrast, the revised simplex method does not solve *symmetric* linear systems and, hence, typically maintains an LU factorization [18] instead of an LDL^T factorization, which requires more computations [19]. (ii) In contrast to the simplex method, the proposed algorithm does not have to be started in a complementary basic solution. Moreover, the starting point does not have to be feasible (neither in the primal nor dual sense). (iii) Finally, the proposed method does not restrict its iterates to be vertices of the feasible set.

The main contribution of this letter is, hence, an LP method that is based on proximal-point iterations and the QP solver in [15]. Moreover, we show that an implementation of the proposed method can outperform state-of-the-art LP solvers for small/medium size LPs, which in turn can improve computational performance in control applications. In particular, we show how the computational burden in geometrical and combinatorial multi-parametric programming algorithms can be reduced by using the proposed solver.

The outline of the letter is as follows: In Section II we introduce how proximal-point iterations can be used to solve LPs and how such iterations can be performed cheaply, leading up to the proposed LP algorithm presented in Section III. We then compare an implementation of the proposed method with state-of-the-art LP solvers on small/medium size LPs in Section IV, and show that the proposed method can lead to over an order of magnitude speedup when solving LPs encountered in explicit MPC applications.

This work was partly supported by the Swedish Research Council (VR) under contract number 2017-04710.

D. Arnström and D. Axehill are with the Division of Automatic Control, Linköping University, Sweden daniel.{arnstrom,axehill}@liu.se

A. Bemporad is with the Department of Computer Science and Engineering, IMT School for Advanced Studies Lucca, Lucca, Italy alberto.bemporad@imtlucca.it

A. Notation

The operator $[\cdot]_i$ extracts the ith row of a matrix or vector. Similarly, $[\cdot]_{\mathcal{I}}$ extracts all rows of a matrix or vector given by the index set $\mathcal{I} \subset \mathbb{N}_{1:m}$, where $\mathbb{N}_{1:m} \triangleq \{1, 2, \ldots, m\}$. Moreover, we denote the complement of an index set with an overline, e.g., $\overline{\mathcal{I}} \equiv \mathbb{N}_{1:m} \backslash \mathcal{I}$. The active set at a point $x \in \mathbb{R}^n$ is denoted $\mathcal{A}(x)$ and is defined as $\mathcal{A}(x) \triangleq \{i \in \mathbb{N}_{1:m} : [A]_i x = [b]_i\}$, i.e., all inequalities of (1) that holds with equality (are *active*) at x. Specifically, we let $\mathcal{A}^*$ denote $\mathcal{A}(x^*)$.

II. Preliminaries

A. Proximal-point iterations

By performing so-called proximal-point iterations [13], [14], the LP in (1) can be solved by solving a sequence of QPs (more specifically perturbed LDPs) in the form

$$\begin{aligned} x^{j+1} = &\underset{x}{\text{argmin}} \quad f^T x + \frac{\epsilon}{2}\|x - x^j\|_2^2 \\ &\text{s.t.} \quad Ax \leq b, \end{aligned} \tag{2}$$

where $x^j \to x^*$ when $j \to \infty$ (see, e.g., Theorem 10.28 in [14]) and where $\epsilon > 0$ is a regularization parameter. Not only will $x^j \to x^*$, but the iterates produced by (2) are decreasing in terms of the objective function of (1):

Lemma 1 (Descent of proximal-point iterations): If $Ax^j \leq b$ and $x^{j+1} \neq x^j$ in (2), $f^T x^{j+1} < f^T x^j$.

Proof: Let the objective function in (2) be denoted $J^j(x) \triangleq f^T x + \frac{\epsilon}{2}\|x - x^j\|$. We then have that $J^j(x^j) = f^T x^j$ and that $J^j(x^{j+1}) = f^T x^{j+1} + \frac{\epsilon}{2}\|x^{j+1} - x^j\|_2^2$. Moreover, since x^{j+1} is the minimizer of (2), we get

$$J^j(x^{j+1}) \leq J^j(x^j) \Leftrightarrow f^T x^{j+1} + \frac{\epsilon}{2}\|x^{j+1} - x^j\|_2^2 \leq f^T x^j,$$

which in combination with $x^{j+1} \neq x^j$ yields the desired result $f^T(x^{j+1} - x^{j+1}) \leq -\frac{\epsilon}{2}\|x^{j+1} - x^j\|_2^2 < 0$. ■

By scaling the objective function with $\frac{1}{\epsilon}$ and removing terms not containing x, we rewrite (2) as

$$\begin{aligned} x^{j+1} = &\underset{x}{\text{argmin}} \quad \frac{1}{2}\|x\|_2^2 + \left(\frac{1}{\epsilon}f - x^j\right)^T x \\ &\text{s.t.} \quad Ax \leq b, \end{aligned} \tag{3}$$

with the corresponding dual problem

$$\underset{\lambda \geq 0}{\text{minimize}} \ \frac{1}{2}\lambda A A^T \lambda + d^T \lambda, \tag{4}$$

where $d \triangleq b + A(\frac{1}{\epsilon}f - x^j)$. Note that different values of ϵ corresponds to a direct scaling of f.

Completing the squares of the objective function in (3) results in $\frac{1}{2}\|x - (x^j - \frac{1}{\epsilon}f)\|_2^2$ and gives a geometrical interpretation of an iteration according to (3) (illustrated in Figure 1): First a step is taken along the negative gradient $-f$ from the current iterate x^j, where $\frac{1}{\epsilon}$ is the step-length taken in this direction. The resulting point $x^j - \frac{1}{\epsilon}f$ is then projected onto the feasible set $\{x \in \mathbb{R}^n : Ax \leq b\}$, yielding the next iterate x^{j+1}.

Remark 1 (Implicit step): The step $-\frac{1}{\epsilon}f$ described above is implicit in the sense that this step only forms the least-distance problem in (3) to be solved. The method that will be used to solve the least-distance problem, described next, will be warm-started, which implies that it is not started in the unconstrained minimizer of (3).

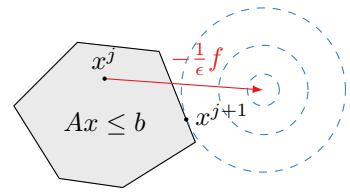


Fig. 1. Illustration of a proximal-point iteration according to (3). The dashed circles are level curves of the objective function in (3). The gray region is the feasible set $\boldsymbol{Ax} \leq \boldsymbol{b}$.

B. Solving least-distance problems

For the iterations in (3) to be practical, the LDPs must be solved efficiently. Since the problem in an iteration is similar to the problem solved in the previous (only the linear term in the objective function is perturbed), warm-starting the solver is advantageous. This makes active-set methods clear candidates for effective inner solvers because of their warm-starting capabilities. Indeed, a QP solver that uses an active-set solver in conjunction with proximal-point iterations was proposed in [20], resulting in a numerically robust and computationally efficient QP solver.

The ideas in [20] were improved upon in [15], where the inner active-set solver was refined to reduce the computational burden. Here we use this dual active-set QP solver (DAQP), and tailor it for LDPs, to solve (3). In fact, the initial proximal-point method that we propose herein can be seen as Algorithm 2 in [15] for the special case when the Hessian $H = 0$. This special case is, nevertheless, very important and we show how the additional structure of $H = 0$ leads to favorable properties.

A brief description, sufficient for our purpose, of how DAQP solves (3) is given below and the reader is referred to [15] for a complete description. DAQP operates on the dual problem in (4) and, as any other active-set algorithm, updates a working set $\mathcal{W} \subseteq \mathbb{N}_{1:m}$ until $\mathcal{W} = \mathcal{W}^{j+1}$, where $\mathcal{W}^{j+1}$ is the active set at x^{j+1}, i.e., $\mathcal{W}^{j+1} \triangleq \mathcal{A}(x^{j+1})$. Changes to $\mathcal{W}$ are decided by the solution to a system of symmetric linear equations (defined by the current working set) in the form $[A]_{\mathcal{W}}[A]_{\mathcal{W}}^T y = -[d]_{\mathcal{W}}$. To solve these symmetric linear system of equations efficiently, an LDL^T factorization is maintained so that $[A]_{\mathcal{W}}[A]_{\mathcal{W}}^T = LDL^T$, i.e., L and D are updated every time the working set $\mathcal{W}$ changes.

III. A proximal-point LP method

Next, we combine the ideas of the proximal-point iterations described in Section II-A and the dual-active solver DAQP to propose a proximal-point method for solving the LP in (1). This proximal-point method is given in Algorithm 1 and consists of three steps:

i) form the perturbed LDP (Step 2)
ii) solve the LDP (Step 3)
iii) check if a fixed point has been reached (Step 4)

These steps are repeated until a fixed point is reached, i.e., until $x^{j+1} \approx x^j$. As mentioned before, the LDPs in Step 3 are solved using DAQP, which takes as inputs A, d, defining the dual of the LDP in (4), a starting iterate λ, and a starting working set $\mathcal{W}$. Importantly, DAQP is warm-started In Step 3 of Algorithm 1 with the working set from the previous outer iteration, reducing the computational burden significantly. Even more advantageous, the matrix factors L and D used in DAQP can be reused directly between outer iterations, i.e., DAQP can be *hot*-started in Step 3.

Algorithm 1 A proximal-point method for solving (1)

Input: $\epsilon > 0, f, A, b, \mathcal{W}^0, \lambda^0, x^0, \eta > 0, j \leftarrow 0$
Output: $x^*, \mathcal{A}^*$

1: **while** true **do**
2: $\quad v^j \leftarrow \frac{1}{\epsilon} f - x^j; \quad d^j \leftarrow b + Av^j$
3: $\quad [x^{j+1}, \lambda^{j+1}, \mathcal{W}^{j+1}] \leftarrow$ **DAQP**$(A, d^j, \lambda^j, \mathcal{W}^j)$
4: $\quad$ **if** $\|x^{j+1} - x^j\| < \eta$ **then**
5: $\quad\quad$ **return** $x^{j+1}, \mathcal{W}^{j+1}$
6: $\quad j \leftarrow j + 1$

The constant $\eta > 0$ in Algorithm 1 is a tolerance used to determine when a fixed point has approximately been reached.

Remark 2 (Complexity certification): If Algorithm 1 is used to solve LPs in the context of linear MPC, the certification method presented in [21] can be used to determine a worst-case complexity bound for Algorithm 1 for a given MPC problem.

A. Unboundedness and superfluous iterations

Even though the iterates of Algorithm 1 will converge to an optimal solution, the algorithm has some shortcomings. First off, the algorithm does not directly detect whether the LP is unbounded; one can observe that the iterates decrease the objective function indefinitely, but a more direct way of detecting unboundedness is desirable.

Another shortcoming is that the algorithm might take short steps in certain situations, leading to superfluous iterations. Such superfluous iterations, exemplified in Figure 2, occur when ϵ is selected too large (resulting in $-\frac{1}{\epsilon} f$ becoming small) or if the normal of the linear manifold defined by $\mathcal{W}^j$ is approximately perpendicular to f.

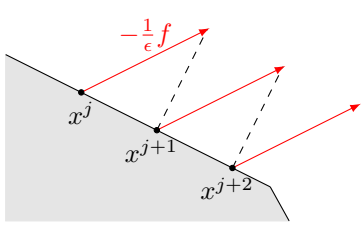


Fig. 2. Example of redundant iterations performed by Algorithm 1 that emerge when $\mathcal{W}^{j+1} = \mathcal{W}^j$.

Both of these shortcomings arise when $\mathcal{W}^{j+1} = \mathcal{W}^j$ in Step 3 of Algorithm 1. We will now show how both shortcomings can be ameliorated by extending Algorithm 1 with an additional step if $\mathcal{W}^{j+1} = \mathcal{W}^j$. In particular, the extension is based on $x^{j+1} - x^j$ being a descent direction (shown in Lemma 1):

Instead of performing another proximal-point iteration when $\mathcal{W}^{j+1} = \mathcal{W}^j$, the iterate x is updated by moving in the descent direction $x^{j+1} - x^j$ until primal feasibility is lost. The first blocking constraint (i.e., the first constraint that becomes violated) is then added to $\mathcal{W}^j$, ensuring that $\mathcal{W}^j$ changes in every outer iteration.

Concretely, we let $\Delta x \triangleq x^{j+1} - x^j$ and determine the first blocking constraint l as

$$l = \text{argmin}_{i \in \mathcal{B}} \frac{[b]_i - [A]_i x^j}{[A]_i \Delta x}, \tag{5}$$

where the set $\mathcal{B} \triangleq \{i : [A]_i \Delta x > 0\}$ contains all possible blocking constraints. Finally, the working set and iterate are updated as

$$\mathcal{W} \leftarrow \mathcal{W} \cup \{l\}, \quad x^{j+1} \leftarrow x^j + \left(\frac{[b]_l - [A]_l x^j}{[A]_l \Delta x} \right) \Delta x. \tag{6}$$

Remark 3: The extension is similar to an iteration of the active-set LP algorithm presented in [22] and [23].

This additional step can also be used to detect an unbounded LP through the following lemma:

Lemma 2: Let $Ax^j \leq b$ and $\Delta x = x^{j+1} - x^j$. If the set $\{i : [A]_i \Delta x > 0\} = \emptyset$, the LP in (1) is unbounded.

Proof: From Lemma 1 we have that Δx is a descent direction. Moving in the direction of Δx can, hence, make the objective of the LP in (1) arbitrarily small. Formally put: $f^T(x^j + \alpha \Delta x) \to -\infty$ when $\alpha \to \infty$. Moreover, if $\{i : [A]_i \Delta x > 0\} = \emptyset$ we have that $A\Delta x \leq 0$ and, hence,

$$A(x^j + \alpha \Delta x) = Ax^j + \alpha A \Delta x \leq b, \tag{7}$$

for any $\alpha > 0$, i.e., $x + \alpha \Delta x$ is primal feasible for any $\alpha > 0$. In conclusion, the LP is unbounded. ■

Consequently, Lemma 2 implies that an unbounded LP is detected whenever $\mathcal{B} = \emptyset$ in (5).

Algorithm 1 amended with the steps described above is summarized in Algorithm 2.

Algorithm 2 Extended version of Algorithm 1 that detects unboundedness and avoids redundant iterations.

Input: $\epsilon > 0, f, A, b, \mathcal{W}^0, \lambda^0, x^0, j \leftarrow 0$
Output: $x^*, \mathcal{A}^*$

1: **while** true **do**
2: $\quad v^j \leftarrow \frac{1}{\epsilon} f - x^j; \quad d^j \leftarrow b + Av^j$
3: $\quad [x^{j+1}, \lambda^{j+1}, \mathcal{W}^{j+1}] \leftarrow$ **DAQP**$(A, d^j, \lambda^j, \mathcal{W}^j)$
4: $\quad$ **if** $\mathcal{W}^{j+1} = \mathcal{W}^j$ **then**
5: $\quad\quad$ **if** $\|x^{j+1} - x^j\| < \eta$ **then return** $x^{j+1}, \mathcal{W}^{j+1}$
6: $\quad\quad \Delta x \leftarrow x^{j+1} - x^j$
7: $\quad\quad \mathcal{B} \leftarrow \{i \notin \mathcal{W}^j : [A]_i \Delta x > 0\}$
8: $\quad\quad$ **if** $\mathcal{B} = \emptyset$ **then return** unbounded
9: $\quad\quad$ **else**
10: $\quad\quad\quad l = \min_{i \in \mathcal{B}} \frac{[b]_i - [A]_i x^j}{[A]_i \Delta x}$
11: $\quad\quad\quad \mathcal{W}^{j+1} \leftarrow \mathcal{W}^j \cup \{l\}$
12: $\quad\quad\quad x^{j+1} \leftarrow x^j + \left(\frac{[b]_l - [A]_l x^j}{[A]_l \Delta x} \right) \Delta x$
13: $\quad j \leftarrow j + 1$

B. Finite termination

As mentioned in Section II-B, Algorithm 1 can be seen as a special case of Algorithm 2 in [15] for $H = 0$. This extra structure gives some additional, favorable, properties.

For instance, a common occurrence when proximal-point iterations are used to solve *QPs* is that multiple iterations have to be performed even once $\mathcal{W} = \mathcal{A}^*$ and $x^j \approx x^*$, that is, a tail of inefficient iterations are performed. This is not the case for LPs since the optimum is attained at the boundary of the feasible region:

Lemma 3 (Termination when $\mathcal{W}^j = \mathcal{A}^*$): If $j \neq 0$ and $\mathcal{W}^j = \mathcal{A}^*$ in Algorithm 1, the algorithm terminates in the subsequent iteration.

Proof: Directly follows from that the active set defines all optimizers in an LP ($\mathcal{A}(x^j) \in \mathcal{A}^* \implies x^j \in x^*$) and that any $x \in x^*$ is a fixed point to (2) (see, e.g., [13, Sec. 2.3]). ■

Next, we are interested in showing that Algorithm 2 terminates in a finite number of iterations for any given LP. First we show that the algorithm makes progress in every iteration for feasible LPs:

Lemma 4 (Progress of Algorithm 2): If the LP in (1) is feasible, $\mathcal{W}^j$ and x^j in Algorithm 2 have the following properties:

1) $f^T x^{j+1} < f^T x^j$ if $x^j \neq x^*$.
2) $\mathcal{W}^{j+1} \neq \mathcal{W}^j$ if $\mathcal{W}^j \neq \mathcal{A}^*$,

Proof: 1) Since the LP is assumed feasible, we have that x^{j+1} in (3) exists. Now, if $\mathcal{W}^{j+1} \neq \mathcal{W}^j$ the proposition directly follows from Lemma 1. If $\mathcal{W}^{j+1} = \mathcal{W}^j$ we will take a non-zero step in a descent direction (Step 12) which gives the desired descent.

2) If $\mathcal{W}^{j+1} \neq \mathcal{W}^j$ in Step 3, the proposition is trivially true. Otherwise, $\mathcal{W}^j$ will be updated by adding a constraint to it in Step 11 (since a bounded LP $\implies \mathcal{B} \neq \emptyset$), resulting in $\mathcal{W}^{j+1} = \mathcal{W}^j \cup \{l\} \neq \mathcal{W}^j$. ■

We are now ready to prove that Algorithm 2 terminates after a finite number of iterations.

Theorem 1 (Finite termination): For any LP in the form (1), Algorithm 2 terminates after a finite number of iterations, either by detecting infeasibility or by finding an optimal solution x^*.

Proof: First consider the case when the LP is feasible. Then from Lemma 4 we have that the algorithm makes progress in each iteration in the sense that the objective function decrease and a new working set is obtained. Now, since the set of all possible $\mathcal{W}$ is finite we get that $\mathcal{W}^k = \mathcal{A}^*$ sooner or later, at which point the algorithm terminates according to Lemma 3.

Next consider the case when the LP is infeasible. If the LP is primal infeasible, infeasibility will be detected by DAQP during the first outer iteration of Algorithm 2. Finally, if the LP is unbounded the algorithm will, again because of the descent property of the iterates and that the set of all possible $\mathcal{W}$ is finite, sooner or later result in $\mathcal{W}^j$ such that $\mathcal{B} = \emptyset$ in Step 8, resulting in the algorithm terminating. ■

IV. Numerical Experiments

We compare a C implementation of Algorithm 2 (DAQP PRX) with a set of state-of-the-art LP solvers. All experiments are performed on an Intel 2.7 GHz i7-7500U CPU and the reported solution times are the internal solution times provided by each solver. To account for the variability in pure solution times, each LP is solved five times and the median of these solves is the used solution time. Moreover, DAQP PRX is always cold started with $x^0 = 0$ and $\mathcal{W}^0 = \emptyset$. Finally, unless stated otherwise, $\epsilon = 1$ and $\eta = 10^{-6}$ are used for DAQP PRX and default settings are used for the other solvers.

A. Randomized LPs

First, we consider randomized, dense, LPs with varying n and $m = 4n$. The elements of f, A and b are generated as

$$[f]_i \sim \mathcal{N}(0,1), \quad [A]_{ij} \sim \mathcal{N}(0,1), \quad [b]_i \sim \mathcal{U}([0,1]). \quad (8)$$

We compare DAQP PRX with the primal and dual simplex methods provided by CPLEX and Gurobi. The solution times passed internally from each solver are reported in Figure 3a. For each n, the average solution time among these 100 LPs are shown as solid lines. The best- and worst-case times are also shown for DAQP PRX as dotted lines.

To separate the implementations of the solvers and the underlying LP algorithms, we also compare the average number of iterations for solving the LPs in Figure 3b. The number of iterations reported for DAQP PRX is the total number of inner iterations from all calls to DAQP, which is equivalent to the total number of working-set changes. The reported number of iterations for the other solvers are the iteration count directly passed from the solvers.

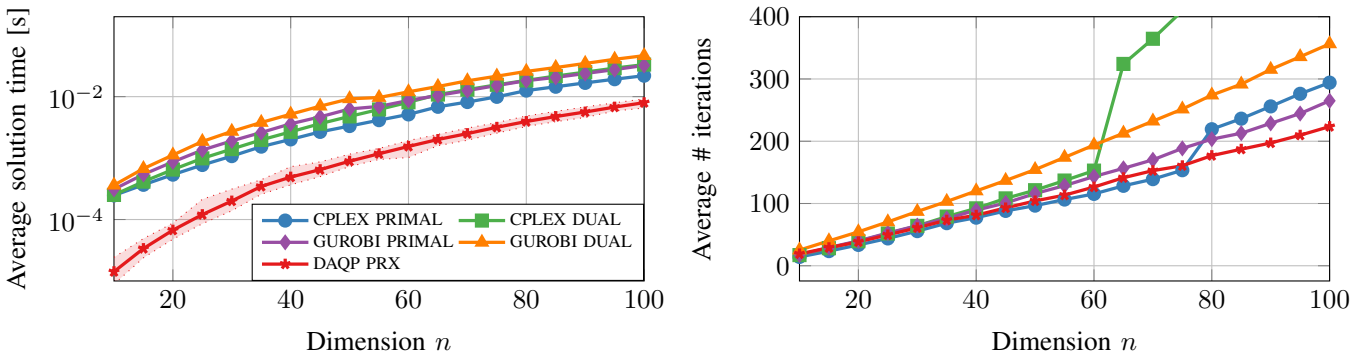
From Figure 3a it can be seen that the proposed algorithm is an order of magnitude faster than the implementations of the simplex method in CPLEX and Gurobi. Moreover, in Figure 3b it can be seen that the average number of iterations is relatively low for DAQP PRX, highlighting not only the validity of the implementation but also the underlying LP algorithm. Also, recall that the systems of linear equations solved in an iteration of DAQP PRX are *symmetric*, while they are asymmetric in the simplex method.

Finally, we report how the number of iterations for Algorithm 1 and Algorithm 2 vary for different values on ϵ in Figure 4. The number of iterations increases rapidly for Algorithm 1 when $\epsilon \to \infty$ due to the step $-\frac{1}{\epsilon} f$ becoming smaller (resulting in the issue described in Section III-A). In contrast, the number of iterations for Algorithm 2 remains relatively small even when ϵ increase.

Remark 4 (Selecting ϵ): Note that since ϵ scales f, the size of f itself is important when selecting ϵ. Moreover, the size of the feasible set $Ax \leq b$ is also relevant in this selection. As a general rule of thumb, selecting $\epsilon \approx ||f|| \frac{||A||}{||b - Ac||}$, where $c \in \mathbb{R}^n$ is close to the center of the feasible set, usually gives good results. Nonetheless, as is apparent from Figure 4, the number of DAQP-PRX iterations does not significantly change with respect to ϵ for a wide range of values.

B. Applications to explicit MPC

To show the practicality of the proposed LP method further, we consider LPs that need to be solved when the explicit solutions [11] for a set of linear MPC problems are computed.



(a) Solution time

(b) Number of iterations

Fig. 3. Average solution time and number of iterations over 100 randomly generated LPs according to (8) for varying n and $m = 4n$. The dotted lines for DAQP in 3a marks the best- and worst-case solution time recorded when solving the 100 LPs.

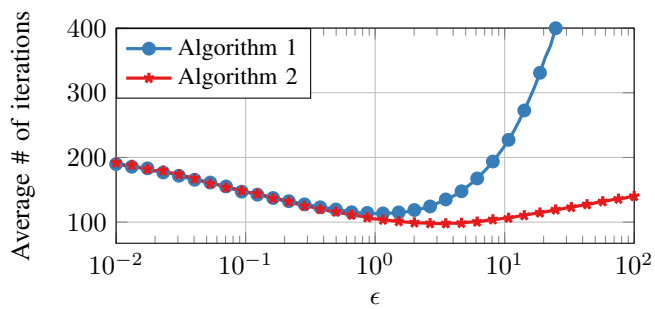


Fig. 4. Average number of iterations over 250 randomly generated LPs according to (8) with $n = 50$, $m = 200$, for varying values of ϵ.

In particular, we consider LPs that are solved to remove redundant constraint and to compute Chebyshev centers, central in geometrical methods [3]–[6]; and for determining feasibility and degeneracy of critical regions, central in combinatorial methods [7]–[10]. The MPC problems considered are the control of an inverted pendulum on a cart, a DC motor, and a ATFI-16 aircraft, all of which are tutorial problems in the Model Predictive Control Toolbox in MATLAB. We use MPT 3.0 [24], specifically the MATLAB functions `mpt_mpqp_26` and `mpt_enum_pqp`, for computing the explicit solution of the resulting multi-parametric QPs.

1) Redundancy removal: First, we consider LPs that are solved to remove redundant constraints from a polyhedron $P \triangleq \{x \in \mathbb{R}^n : [A]_i x \leq [b]_i\}$. This operation often takes up a significant fraction of the time in geometrical multi-parametric programming algorithms [4]. Moreover, redundancy removal is often performed in post-processing of solutions computed by combinatorial methods [9].

The jth constraint in a H-representation of P is said to be redundant if $P = \{x \in \mathbb{R}^n : [A]_i x \leq [b]_i, i \neq j\}$ and redundancy of constraint j can be determined by solving the LP

$$\begin{aligned} s_j = \underset{x}{\text{minimize}} \quad & [b]_j - [A]_j x \\ \text{subject to} \quad & Ax \leq b. \end{aligned} \tag{9}$$

If $s_j > 0$, the jth constraint is redundant and can be removed from the H-representation of P without changing the underlying polyhedron [25].

Solution times for solving LPs in the form (9) in the MPT 3.0 function `minHRep` are reported in Table I.

2) Computing Chebyshev center: Next, we consider LPs for computing the Chebyshev center of a polyhedron P, which is another important operation in geometrical mpQP algorithms. Given a polytope P with H-representation $Ax \leq b$ its Chebyshev center can be obtained by solving the LP

$$\begin{aligned} \underset{x,r}{\text{maximize}} \quad & r \\ \text{subject to} \quad & [A]_i x + \|[A]_i\|_2 r \leq [b]_i, \quad \forall i. \end{aligned} \tag{10}$$

Moreover, if $Ax \leq b$ is assumed to be a minimal H-representation of P, the Chebyshev center on the facet given by the jth hyperplane is obtained by solving the LP

$$\begin{aligned} \underset{x,r}{\text{maximize}} \quad & r \\ \text{subject to} \quad & [A]_j x = [b]_j, \\ & [A]_i x + \|[A]_i\|_2 r \leq [b]_i, \quad \forall i \neq j. \end{aligned} \tag{11}$$

Solution times for solving LPs in the form (10) and (11) encountered in the MPT 3.0 function `chebyCenter` are reported in Table I.

3) Feasibility and degeneracy detection: Finally, we consider LPs that are solved in combinatorial mpQP algorithms [7]–[9] for determining if critical regions are empty or not, and for detecting if the solution on the critical region is degenerate. Given an active set $\mathcal{A}$, the corresponding critical region is given by $C_{\mathcal{A}} \triangleq \{x \in \mathbb{R}^N : s_{\bar{\mathcal{A}}}(x) \geq 0,\ \lambda_{\mathcal{A}}(x) \geq 0,\ Ax \leq b\}$, where both $s_{\bar{\mathcal{A}}}(x)$ and $\lambda_{\mathcal{A}}(x)$ are affine expressions of x, making $C_{\mathcal{A}}$ a polyhedron (cf., e.g., Theorem 2 in [3] for details). Determining if $C_{\mathcal{A}}$ is non-empty, in combination with detecting degenerate solutions, can be done by solving the LP

$$\begin{aligned} \underset{t,x}{\text{maximize}} \quad & t \\ \text{subject to} \quad & s_{\bar{\mathcal{A}}}(x) \geq t, \quad \lambda_{\mathcal{A}}(x) \geq t, \quad Ax \leq b. \end{aligned} \tag{12}$$

If $t \geq 0$, $C_{\mathcal{A}}$ is non-empty. Furthermore, when $t = 0$, strict complementarity does not hold for the corresponding solution (see, e.g., [7] or [9] for details).

Solution times for solving LPs in the form (12) encountered in the MPT 3.0 function `mpt_enum_pqp` are reported in Table I.

TABLE I

SOLUTION TIMES FOR VARYING LP SOLVERS FOR SOLVING LPS ENCOUNTERED IN MPT 3.0 WHEN COMPUTING EXPLICIT SOLUTIONS. FOR EACH SCENARIO, n IS THE DIMENSION OF THE DECISION VARIABLE AND N IS THE NUMBER OF SOLVED LPS.

			Average solution time [ms]					Worst-case solution time [ms]				
	n	N	DAQP PRX	CPLEX	GRB	GLPK	MSK	DAQP PRX	CPLEX	GRB	GLPK	MSK
Redundancy removal												
DC motor	6	662	0.0050	0.22	0.21	0.13	0.52	0.08	0.35	0.34	0.20	1.3
Inverted pendulum	8	1101	0.0063	0.23	0.22	0.13	0.46	0.016	0.28	0.27	0.20	0.8
Aircraft	10	5613	0.0065	0.26	0.24	0.16	0.62	0.023	0.35	0.37	0.22	2.2
Chebyshev center												
DC motor	7	490	0.009	0.26	0.23	0.21	0.66	0.025	0.37	0.35	13.4	4.2
Inverted pendulum	9	554	0.015	0.31	0.25	0.22	0.73	0.030	0.40	0.37	0.37	1.6
Aircraft	11	3377	0.012	0.30	0.26	0.25	0.87	0.037	0.51	0.50	8.9	2.7
Feas./degen. detection												
DC motor	7	161	0.008	0.25	0.21	0.15	0.67	0.017	0.31	0.25	0.22	1.8
Inverted pendulum	9	365	0.013	0.26	0.23	0.17	0.58	0.021	0.32	0.31	2.4	1.1
Aircraft	11	891	0.014	0.33	0.28	0.29	0.89	0.033	0.54	0.50	7.2	1.8

4) Discussion: In accordance with the result from Section IV-A, Table I shows that the implementation of the proposed LP algorithm outperforms other LP solvers with over an order of magnitude speedup, both on average and in the worst case. The proposed method could be terminated early when the redundancy removal LPs in (9) and the feasibility LPs in (12) are solved since we are only interested in the sign of the optimal objective value, not the actual value. Such early termination would, obviously, reduce the solution time further.

V. CONCLUSION

We have proposed an LP algorithm based on performing proximal-point iterations and solving LDPs using a dual active-set QP solver. An implementation of the algorithm yielded an order of magnitude speedup in solution time for solving small/medium size LPs compared with state-of-the-art LP solvers. In particular, we have shown how multi-parametric programming algorithms can benefit from using the proposed method for redundancy removal, computing Chebyshev centers and for detecting feasibility/degeneracy of critical regions.

As a final remark, the main performance limitation for efficiently solving larger LPs is that the inner QP solver does not exploit sparsity. Whether the solver can be extended to solve larger LPs efficiently by using sparse linear algebra routines, for example, by using the LDL^T updates proposed in [26], is a subject for future research.

REFERENCES

[1] C. V. Rao and J. B. Rawlings, "Linear programming and model predictive control," *Journal of Process Control*, vol. 10, no. 2-3, pp. 283–289, 2000.

[2] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[3] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[4] P. Tøndel, T. A. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," *Automatica*, vol. 39, no. 3, pp. 489–497, 2003.

[5] P. Grieder, F. Borrelli, F. Torrisi, and M. Morari, "Computation of the constrained infinite time linear quadratic regulator," *Automatica*, vol. 40, no. 4, pp. 701–708, 2004.

[6] C. N. Jones and M. Morrari, "Multiparametric linear complementarity problems," in *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE, 2006, pp. 5687–5692.

[7] A. Gupta, S. Bhartiya, and P. Nataraj, "A novel approach to multiparametric quadratic programming," *Automatica*, vol. 47, no. 9, pp. 2112–2117, 2011.

[8] R. Oberdieck, N. A. Diangelakis, and E. N. Pistikopoulos, "Explicit model predictive control: A connected-graph approach," *Automatica*, vol. 76, pp. 103–112, 2017.

[9] P. Ahmadi-Moshkenani, T. A. Johansen, and S. Olaru, "Combinatorial approach toward multiparametric quadratic programming based on characterizing adjacent critical regions," *IEEE Transactions on Automatic Control*, vol. 63, no. 10, pp. 3221–3231, 2018.

[10] M. Herceg, C. N. Jones, M. Kvasnica, and M. Morari, "Enumeration-based approach to solving parametric linear complementarity problems," *Automatica*, vol. 62, pp. 243–248, 2015.

[11] A. Bemporad, "Explicit model predictive control," in *Encyclopedia of Systems and Control*, J. Baillieul and T. Samad, Eds. London: Springer London, 2019, pp. 1–7.

[12] S. V. Rakovic and M. Baric, "Parameterized robust control invariant sets for linear systems: Theoretical advances and computational remarks," *IEEE Transactions on Automatic Control*, vol. 55, no. 7, pp. 1599–1614, 2010.

[13] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[14] A. Beck, *First-order methods in optimization*. SIAM, 2017.

[15] D. Arnström, A. Bemporad, and D. Axehill, "A dual active-set solver for embedded quadratic programming using recursive LDL' updates," *arXiv preprint arXiv:2103.16236*, 2021.

[16] G. B. Dantzig, *Linear programming and extensions*. Princeton university press, 1998, vol. 48.

[17] S. J. Wright, *Primal-dual interior-point methods*. SIAM, 1997.

[18] R. H. Bartels and G. H. Golub, "The simplex method of linear programming using LU decomposition," *Communications of the ACM*, vol. 12, no. 5, pp. 266–268, 1969.

[19] C. F. Van Loan and G. H. Golub, *Matrix computations*. Johns Hopkins University Press Baltimore, 1983.

[20] A. Bemporad, "A numerically stable solver for positive semidefinite quadratic programs based on nonnegative least squares," *IEEE Transactions on Automatic Control*, vol. 63, no. 2, pp. 525–531, 2017.

[21] D. Arnström, A. Bemporad, and D. Axehill, "Complexity certification of proximal-point methods for numerically stable quadratic programming," *IEEE Control Systems Letters*, vol. 5, no. 4, pp. 1381–1386, 2021.

[22] M. J. Best and K. Ritter, *Linear programming*. Prentice Hall Upper Saddle River, NJ, 1985.

[23] S. Sloan, "A steepest edge active set algorithm for solving sparse linear programming problems," *International Journal for Numerical Methods in Engineering*, vol. 26, no. 12, pp. 2671–2685, 1988.

[24] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *2013 European control conference (ECC)*. IEEE, 2013, pp. 502–510.

[25] K. Fukuda, "Frequently asked questions in polyhedral computation," *ETH, Zurich, Switzerland*, 2004. [Online]. Available: https://people.inf.ethz.ch/fukudak/polyfaq/

[26] T. A. Davis and W. W. Hager, "Row modifications of a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 3, pp. 621–639, 2005.