A primal active-set minimal-representation algorithm for polytopes with application to invariant-set calculations

Emil Klintberg[1], Magnus Nilsson[1], Lars Johannesson Mårdh[2], Ankit Gupta[3]

***Abstract*— This paper provides a description of a practically efficient minimal-representation algorithm for polytopes. The algorithm is based on a primal active-set method that heavily exploits warm-starts and low-rank updates of matrix factorizations in order to reduce the required computational work. By using a primal active-set method, several nonredundant inequalities can be identified for each solved linear program. Implementation details are provided both for the minimal-representation algorithm and for the underlying active-set method.**

I. Introduction

Many problems in robust control require the calculation of the Maximal Robust Positive Invariant (MRPI) set (see e.g. [1], [2]). An exact representation of the MRPI set can be complex or intractable to find, and less complicated inner approximations are therefore often used as surrogates [3]–[5]. However, arbitrary close polyhedral approximations of the MRPI set, so-called *polytopic MRPI sets*, can be calculated iteratively by using geometrically motivated methods [2], [6].

As a byproduct from the iterative calculations of polytopic MRPI sets, *redundant inequalites* are typically generated. Such inequalities are implied by other inequalities and are thus unnecessary in order to describe the set. If the redundancies are not eliminated, the number of inequalities generally grows exponentially during the calculations and the problem quickly becomes intractable. Thus, finding *minimal representations* of the polytopes, i.e. descriptions without redundant inequalities, is crucial for the effectiveness of the geometrical methods. However, finding such minimal representations often constitute computational bottlenecks.

The literature on computational methods for finding minimal representations of polytopes is relatively scarce and scattered over several decades [7]–[13]. Most methods rely on solving one Linear Program (LP) for each inequality, and classify the inequality as either redundant or necessary based on the optimal value of the LP [10], [11], [13]. Insights on the underlying optimization procedures are usually not provided, although a thoughtful choice can affect the computational times by several orders of magnitude. In this paper, we aim at addressing this by exploiting features of the underlying solver in order to enhance the algorithm.

We propose a practically efficient minimal-representation algorithm based on the primal active-set method described in [14]. Since the underlying method is always primal feasible, the algorithm is able to classify multiple inequalities by solving a single LP. This means that, on average, fewer than one LP for each inequality have to be solved. Additionally, due to the structure of the minimal-representation problem, the underlying active-set method can be warm-started (or even hot-started). To enhance the warm-starting, we provide a heuristic to minimize the difference between subsequently solved LPs. Numerical experiments indicate that this is particularly useful to achieve a computationally efficient algorithm.

The paper is organized as follows: In Section II, we recall a method for checking if a single inequality is redundant or not and introduce some notations. In Section III, the active-set minimal-representation algorithm is presented along with some implementation details. In Section IV, numerical experiments are performed to illustrate the practical performance of the algorithm. The paper is concluded in Section V.

Contributions: The contributions of this paper are threefold:

1) An optimization procedure that is especially useful for minimal representation problems of convex polytopes is presented.
2) In contrast to existing methods, it is shown that several non-redundant inequalities can be identified by solving a single LP.
3) A practically efficient heuristic to warm-start the active set method, in the context of minimal-representation problems, is described.

II. Preliminaries and notations

In this section, we recall redundant inequalities in polyhedral sets, a method for checking if a single inequality is redundant or not, and introduce basic notations.

A. Polytopes and redundant inequalities

A *polyhedron* is defined as the solution set of a finite number of linear inequalities,

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}, \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. In the following we will work with sets of the form (1) that in addition are bounded, so-called *polytopes*.

If an inequality can be removed from the description of a polytope without changing the solution set, we say that the inequality is *redundant*. Similarly, if an inequality is not redundant, it is *necessary* (or *non-redundant*). If all inequalities describing a polytope are necessary, we have a *minimal representation* of the polytope.

[1]Emil Klintberg and Magnus Nilsson are with Zenuity AB and Qamcom Research and Technology AB.

[2]Lars Johannesson Mårdh is with Zenuity AB.

[3]Ankit Gupta is with the Department of Electrical Engineering, Chalmers University of Technology.

B. Single redundancy checking

The problem of identifying whether a row $a_k^T x \leq b_k$ of $Ax \leq b$ is redundant or not, can be determined by solving a LP. To that end, we consider the LP

$$x^\star = \arg\max_x \quad a_k^T x \tag{2a}$$
$$\text{s.t.} \quad Ax \leq b \tag{2b}$$

and classify row k as redundant if the optimal value is less than b_k, i.e. if $a_k^T x^\star < b_k$. In this way, it is clear that a minimal representation of (1) can naively be found by solving m LPs of the form (2). In Section III, we will refine this procedure by using a tailor-made active-set method. As a result, fewer than m LPs have to be solved while the structure of the problem is exploited in order to solve the LPs efficiently.

C. Active-set and index sets

To characterize feasible points of (2), we partition the index set of constraints $\bar{m} = \{1, \ldots, m\}$ for an $x \in \mathcal{P}$ into two disjoint sets

$$\mathcal{A}(x) = \{i \in \bar{m} \mid a_i^T x = b_i\} \tag{3a}$$
$$\mathcal{I}(x) = \{i \in \bar{m} \mid a_i^T x < b_i\} \tag{3b}$$

and introduce shorthand notations for the constraints corresponding to the sets

$$A_{\mathcal{A}} x = b_{\mathcal{A}} \tag{4a}$$
$$A_{\mathcal{I}} x < b_{\mathcal{I}} \tag{4b}$$

where $A = A_{\mathcal{A}\cup\mathcal{I}}$ and $b = b_{\mathcal{A}\cup\mathcal{I}}$. We refer to the set $\mathcal{A}(x)$ as the *active-set* at the point x.

Note that provided absence of duplicate and weakly redundant inequalities, a feasible point x is located at a vertex of $\mathcal{P}$ if $|\mathcal{A}(x)| \geq n$. We emphasize points located at a vertex by denoting $\bar{\mathcal{A}}(x) = \mathcal{A}(x)$ when $|\mathcal{A}(x)| \geq n$, and introduce the notations

$$A_{\bar{\mathcal{A}}} x = b_{\bar{\mathcal{A}}} \tag{5}$$

for the constraints that are active at the vertex.

III. An active-set minimal representation algorithm

In this section, we recall primal active-set methods, and detail the active-set minimal-representation algorithm.

A. Primal active-set methods

In primal active-set methods, dual feasibility is searched for, while primal feasibility and complementary slackness are maintained. Let us consider the LP

$$\min_x \quad f^T x \tag{6a}$$
$$\text{s.t.} \quad Ax \leq b. \tag{6b}$$

By introducing the Lagrange dual variables $\mu \in \mathbb{R}^{|\mathcal{A}(x)|}$ corresponding to the inequalities (in the following referred to as constraints) that hold with equality at x, the Karush-Kuhn-Tucker (KKT) optimality conditions of the LP can then be expressed as [15]

$$A_{\mathcal{A}}^T \mu^\star = -f \tag{7a}$$
$$A_{\mathcal{A}} x^\star = b_{\mathcal{A}} \tag{7b}$$
$$A_{\mathcal{I}} x^\star < b_{\mathcal{I}} \tag{7c}$$
$$\mu^\star \geq 0, \tag{7d}$$

where $x^\star$ and $\mu^\star$ refers to the optimal primal-dual solution of (6).

A well-known property of LPs is that the optimal solution can be sought at the vertices of the feasible region (see e.g. [16] for a comprehensive introduction to LPs). Hence, provided that we have an initial feasible point $x^{(0)}$ located at a vertex, i.e.

$$A_{\bar{\mathcal{A}}} x^{(0)} = b_{\bar{\mathcal{A}}} \tag{8a}$$
$$A_{\mathcal{I}} x^{(0)} < b_{\mathcal{I}} \tag{8b}$$

we can search for the optimum by moving from vertex to vertex until (7a) and (7d) are fulfilled. The procedure is detailed in the following.

At each iteration, μ is calculated by solving (7a). If $\mu \geq 0$, optimality has been reached and the optimization procedure is terminated. Otherwise, an index j corresponding to a negative dual variable is removed from $\bar{\mathcal{A}}(x)$ to form the set $\underline{\mathcal{A}}(x)$. A new vertex is then found by updating the variables according to

$$x^{(k+1)} = x^{(k)} + ts, \tag{9}$$

where $s \in \mathbb{R}^n$ is a search direction and $t \in \mathbb{R}_+$ is a step size.

The search direction s is found by projecting the negative normal vector of the removed constraint on the null space of $A_{\underline{\mathcal{A}}}$. Feasibility is thus guaranteed to be maintained for some non-negative step size since we are moving away from the removed constraint and $\underline{\mathcal{A}}$ will be part of the active-set for any step size along the ray defined by s.

The step size t is then calculated as the minimum step size in the direction of s in order to activate an inactive constraint. The index corresponding to the new active constraint is added to the index set (3a) in order to form $\bar{\mathcal{A}}(x^{(k+1)})$. The procedure is then repeated by solving (7a) and checking optimality for the new active-set.

The observant reader may have noted that the procedure requires $|\bar{\mathcal{A}}(x^{(0)})| = n$. If this is not fulfilled, we exclude $|\bar{\mathcal{A}}(x^{(0)})| - n$ constraints from the active-set to impose $\mathbf{rank}(A_{\bar{\mathcal{A}}}) = n - 1$. In cases where $|\bar{\mathcal{A}}(x)| > n$, the procedure may spend $|\bar{\mathcal{A}}(x)| - n$ iterations before moving on to the next vertex.

B. An active-set minimal-representation algorithm

If the primal active-set method is used to solve (2), it follows trivially from primal feasibility of the iterates that every constraint index that has been part of $\bar{\mathcal{A}}(x)$ correspond to a non-redundant inequality. This observation implies that

several non-redundant inequalities can be classified by solving a single LP of the form (2). As a result, fewer than m LPs have to be solved in order to find a minimal representation.

The active set minimal representation algorithm is summarized in Algorithm 1. Note that there are several steps in the algorithm that are crucial for its efficiency but whose implementations are not specified. These steps are mainly the selection of a halfplane in Step 3, the calculation of the dual variables in Step 11, and the projection onto the nullspace in Step 19. These steps are further detailed in Section III-C, III-D and III-E respectively.

Algorithm 1: Primal active-set minimal-representation

```
1  Find a vertex of P
2  while not all halfplanes identified do
3      Choose a halfplane i ∈ m̄ not yet identified
4      // Solve (2) using the primal active-set method
5      while no convergence do
6          // Calculate μ and check for convergence
7          Calculate μ using (7a)
8          if μ ≥ 0 then
9              break
10         end
11         // Calculate search direction and step size
12         Remove constraint from Ā(x^(k)) to form
             A(x^(k))
13         Calculate s ∈ null(A(x)) such that s^T a_i ≥ 0
14         Find step size t
15         Add constraint to A(x^(k)) to form Ā(x^(k+1))
16         if Added constraint not identified then
17             Mark added constraint as non-redundant
18         end
19         // Update variables
20         Update variables x^(k+1) = x^(k) + ts
21     end
22     // Classify halfplane
23     if i ∈ Ā(x^(k)) then
24         Constraint i is non-redundant
25     else
26         Constraint i is redundant
27     end
28 end
```

C. Choosing a halfplane to warm start the active-set method

The performance of active-set methods can in general be significantly improved if a so-called *warm-start* is used to initiate the algorithm [17], [18]. This means that the optimal active-set from a previously solved problem is provided as an initial guess. This has two consequences. First, the problem of finding a feasible initial point to the algorithm can be skipped. Secondly, if the provided active-set is close to the optimal active-set, the algorithm may converge in few iterations. In this section, we describe a way of choosing the next halfplane to identify, such that an efficient warm-start is achieved.

To understand the rationale behind the procedure, we start by looking at the calculation of the step size t (see Step 20 of Algorithm 1). There, we are interested in finding the minimal step size such that a previously inactive constraint becomes active, or similarly to calculate

$$\begin{aligned} t = \arg\max \quad & u \\ \text{s.t.} \quad & A_{\mathcal{I}}(x^{(k)} + us) \leq b_{\mathcal{I}}. \end{aligned} \tag{10}$$

Naturally, as a byproduct from the solution of (10) the following components are available:

$$p_1 = A_{\mathcal{I}} x^{(k)} \tag{11a}$$

$$p_2 = A_{\mathcal{I}} s. \tag{11b}$$

This will be instrumental in the warm starting strategy.

Let us now consider two LPs of the form (2), with gradients a_{k_1} and a_{k_2} and optimal solutions $x_1^\star$ and $x_2^\star$ respectively. Note that the optimal solutions are likely to be close (i.e. $\|x_1^\star - x_2^\star\|$ is small) if the angle between $x_1^\star$ and a_{k_2} (and/or between $x_2^\star$ and a_{k_1}) is small, i.e. if

$$a_{k_2}^T x_1^\star \approx \|a_{k_2}\| \|x_1^\star\|. \tag{12}$$

Based on the observation above, it is thus natural (at Step 3 of Algorithm 1) to select the halfplane that is not yet identified and is closest to fulfill (12) with equality. The computational work that is needed to implement such a strategy is dominated by the calculation of $A_{\mathcal{I}} x^\star$. However, (provided that we have performed at least 2 iterations when solving the previous LP) the needed matrix-vector multiplication can be obtained cheaply as

$$A_{\mathcal{I}} x^\star = p_1 + t p_2 \tag{13}$$

by re-using p_1 and p_2 from the last step size calculation.

Computational tests indicate that the described warm-starting strategy greatly enhances the computational times.

D. Calculating Lagrange dual variables

Recall that μ is provided by the linear system of equations

$$A_{\bar{\mathcal{A}}}^T \mu = -f, \tag{14}$$

where $A_{\bar{\mathcal{A}}}$ is invertible. We note that the coefficient matrix is only undergoing a low-rank change between iterations (i.e. one column is altered). This can be exploited by factorizing the coefficient matrix as $A_{\bar{\mathcal{A}}}^T = \bar{Q}\bar{R}$, and updating the factorizations using a low-rank update. Note that since a QR factorization can be calculated without pivoting, it is easier to low-rank update compared to for example an LU factorization. In this way, a full factorization only has to be performed in the first iteration, and thereafter we can rely on low-rank updates.

E. Projecting on the null space of the active constraints

To calculate s, the negated normal vector of the removed constraint is projected onto the null space of $A_{\underline{\mathcal{A}}}$. A computationally stable way of performing this is to factor $A_{\underline{\mathcal{A}}}^T = \underline{Q}\underline{R}$, and calculate s as

$$s = \begin{cases} q_n, & a_j q_n < 0 \\ -q_n, & \text{otherwise} \end{cases}, \tag{15}$$

where a_j denotes the normal vector of the removed constraint and the unit vector q_n denotes the rightmost column of $\underline{Q}$.

Similarly as for $A_{\bar{\mathcal{A}}}$, only a rank one matrix differ between $A_{\underline{\mathcal{A}}}$ at subsequent iterations. This allows for performing only one full factorization in the first iteration, and then rely on low-rank updates to calculate q_n.

In [14], the search direction is elegantly calculated by modifying R in two steps. However, we advocate using the procedure described in this subsection since this allows for an easy implementation using "off the shelf" factorization routines. This is also supported by practical experiments where the time spent on matrix factorizations often is negligible due to the low-rank updates.

F. Finding an initial vertex

Step 1 of Algorithm 1 is to find an initial vertex. We do not delve deeply into the details of this step since it is only needed once per problem and any established linear programming algorithm can be used to find a vertex. However, it is worth emphasizing that the described procedure can be started from any interior point if the negative gradient is used instead of the negative normal vectors in the calculation of search directions, and Step 12 of Algorithm 1 is skipped until $|\bar{\mathcal{A}}| = n$ is reached.

Essentially, this implies that a ray-shot is performed from the interior point in the direction of the negative gradient. The first constraint that is reached is added to the active-set. The direction of the ray-shot is then corrected to lay in the null space of the active constraints, and the procedure is repeated until n active constraints have been found.

G. Additional comments

Note that in the warm-start of the algorithm, the matrix factorizations from the last iteration of the previous problem can be provided in addition to the active-set. This is sometimes referred to as a hot start (see e.g. [17], [18]) in contrast to a warm-start where only the active set is provided. This means that full factorizations only have to be performed at the first iteration when the first LP is solved. Thereafter, we can rely on low-rank updates.

In invariant-set calculations, the polytopes are often symmetric around the origin. This means that redundancy only has to be verified for half of the inequalities, and the computational efficiency of the algorithm is then (nearly) doubled. In Section IV, we elaborate on this by using numerical experiments.

Finally, we note that once an inequality is identified as redundant, it can be omitted from succeeding LPs in the minimal representation procedure. As a result, the number of constraints is decreasing for the sequence of LPs that is solved in the algorithm.

H. Computational complexity

The computational bottleneck in Newton-type methods (such as e.g. active-set methods) typically lies in the matrix factorizations which generally has cubic complexity in the number of variables. However, due to the low-rank updates, the complexity reduces to quadratic in the number of variables (n in our case), and should therefore not be problematic. This is supported by numerical tests which show that the time spent on factorizations often is negligible.

The cost of the matrix-vector multiplication $y = Ax$ is $2mn$ flops if $A \in \mathbb{R}^{m\times n}$. In invariant-set calculations, we often have $m \gg n$. This indicates that the step size calculation can be time consuming since we compute (11). This is supported by numerical experiments, where (11) often is the most time consuming operation.

I. Comparison towards an alternative method

A similar "primal feasible" minimal-representation algorithm can be constructed by using the simplex algorithm (see e.g. [19] or any textbook on linear programming for an introduction on the simplex algorithm). However, to run the simplex algorithm, the problem has to be formulated on the so-called standard form. As a consequence, an $m \times m$ matrix has to be factorized at each iteration of the algorithm. This is a severe drawback since we often consider examples where $m \gg n$.

We can also observe that the primal active-set method has a computational complexity per iteration that is comparable to the simplex algorithm deployed on the *dual problem* of (6), i.e. the so-called *dual simplex algorithm*. However, the dual simplex algorithm is not primal feasible, and cannot be used in a primal feasible minimal-representation algorithm.

IV. Numerical experiments

In this section, we evaluate the proposed method via numerical experiments.

We consider symmetric polytopes of the form (1), where the elements of the coefficient matrix and the right-hand-side vector are generated using the uniform distributions

$$A_{i,j} = -A_{i+\frac{m}{2},j} \sim \mathcal{U}(-10, 10),\ i \in \left[1, \frac{m}{2}\right],\ j \in [1, n] \tag{16a}$$

$$b_i = b_{i+\frac{m}{2}} \sim \mathcal{U}(1, 10), \quad i \in \left[1, \frac{m}{2}\right]. \tag{16b}$$

The algorithm was implemented in MATLAB, and the experiments were performed on a 2.60 GHz Intel Core i7 Processor with 8 GB of RAM, running Windows 7. The performance is presented for various problem dimensions in Table I. For each problem dimension, the average performance of 20 problems is presented when the symmetry is and is not exploited, and when the warm-starting heuristic described in Section III-C is and is not used. Note that the dimensions of the problems reported in Table I are large compared to most examples considered in the literature.

The average solution time is consistently smallest when both the warm-start heuristic and the symmetry is exploited. By either neglecting the symmetry or not using the warm-starting heuristic, the average computational times are increased approximately by a factor 2.

Moreover, observe that the warm-starting heuristic is effective at reducing the average number of iterations required to solve a LP. The average solution times are as a result

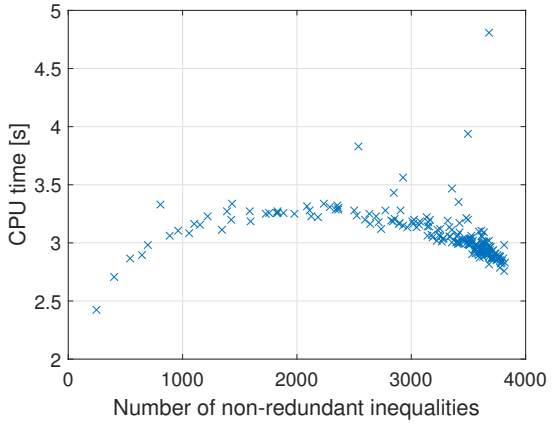

Fig. 1. CPU time (measured using `tic/toc`) vs number of non-redundant inequalities. Every cross corresponds to a minimal-representation problem.

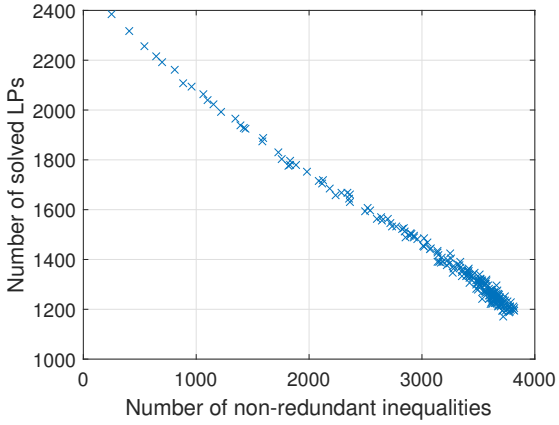

Fig. 2. Number of solved LPs vs vs number of non-redundant inequalities. Every cross corresponds to a minimal-representation problem.

consistently lowered when the heuristic is used. It is, however, interesting to note that the average number of solved LPs increases. This can be explained by the observation that the number of vertices that has to be traversed in order to find the solution increases, and thus also the probability of finding non-redundant inequalities on the way.

A. *Effect of redundancy on computational times*

The effect of redundancy on the computational times is not clearly visible in Table I. In this subsection, we elaborate on this effect.

To that end, we generate 200 polytopes of the form (1), with $n = 10$ and $m = 5000$, and report solution statistics in Figure 1, Figure 2 and Figure 3. In Figure 1, we observe that the solution time is relatively constant for different degrees of redundancy. Additionally, from Figure 2 and Figure 3, we note that the number of solved LPs tend to decrease whereas the number of iterations per LP tend to increase with the number of non-redundant inequalities.

Since several non-redundant inequalities can be identified by solving a single LP, it is intuitive that fewer LPs are solved on average for problems with a large portion of non-redundant inequalities. This explains the trend in Figure 2. Moreover, we note that the solution of (2) can be identical for a maximum of n non-redundant inequalities, whereas this is not the case for redundant inequalities. This is a possible explanation of the increasing trend in Figure 3.

Finally, we recall from Section III-G that inequalities identified as redundant can be omitted from succeeding LPs in the algorithm. Thus, the LPs solved in problems involving few non-redundant inequalities are on average significantly smaller compared to the LPs that are solved in problems with many non-redundant inequalities. This is possibly part of an explanation for the short solution times for problems involving few non-redundant inequalities in Figure 1.

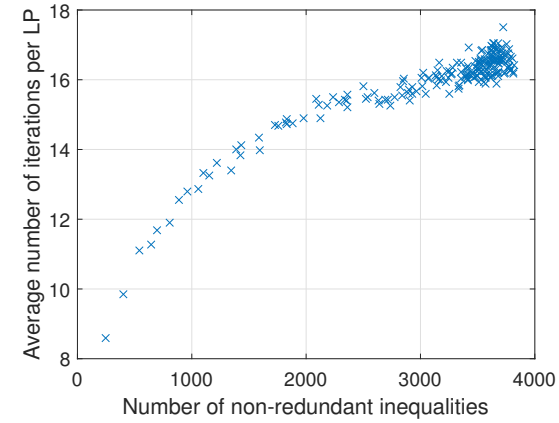

Fig. 3. Average number of iterations per LP vs number of non-redundant inequalities. Every cross corresponds to a minimal-representation problem.

V. Concluding remarks

In this paper, we have provided a description of a practically efficient minimal-representation algorithm of polytopes. In contrast to existing minimal-representation algorithms, properties of the underlying optimization procedure was exploited in order to alleviate the computational burden. Numerical experiments show that demanding examples can be solved in relatively short computational times using a simple implementation in MATLAB.

One appealing property of the proposed algorithm is that several non-redundant inequalities can be identified by solving a single LP. It is relatively straightforward to also identify several *redundant inequalities* by solving a single LP. However, this requires solving an additional linear system of equations (using existing matrix factorizations) at every iteration of the optimization procedure for every inequality for which we want to verify redundancy. Numerical tests indicate that it is hard, especially in higher dimensions, to narrow down the number of potential inequalities such that the strategy becomes useful. In lower dimensions, it may be useful, although our numerical experiments indicate that it is superfluous if the warm-starting heuristic is used.

TABLE I

THE AVERAGE CPU TIME (MEASURED USING TIC/TOC), THE AVERAGE NUMBER OF ITERATIONS PER LP AND THE AVERAGE NUMBER OF SOLVED LPS PER MINIMAL-REPRESENTATION PROBLEM FOR VARIOUS PROBLEM DIMENSIONS.

n	m	sym.	warm-start heu.	Avg. sol. time	Avg. no. of it. / LP	Avg. no of solved LPs / problem	Avg. no. of nonred. ineq.
10	1000	yes	yes	0.27 s	9.89	420.85	175.60
10	1000	yes	no	0.51 s	21.28	419.35	175.60
10	1000	no	yes	0.47 s	8.97	843.75	175.60
10	1000	no	no	0.99 s	21.30	839.80	175.60
10	10000	yes	yes	10.80 s	10.45	4692.95	669.70
10	10000	yes	no	25.65 s	31.13	4681.80	669.70
10	10000	no	yes	19.36 s	9.55	9386.05	669.70
10	10000	no	no	49.67 s	31.13	9366.35	669.70
10	20000	yes	yes	39.24 s	10.68	9518.55	1046.20
10	20000	yes	no	99.00 s	34.61	9503.85	1046.20
10	20000	no	yes	72.13 s	9.75	19037.80	1046.20
10	20000	no	no	194.14 s	34.61	19007.15	1046.20
10	40000	yes	yes	154.53 s	10.76	19259.15	1596.60
10	40000	yes	no	414.94 s	38.04	19235.95	1596.60
10	40000	no	yes	286.48 s	9.85	38523.25	1596.60
10	40000	no	no	823.24 s	38.04	38472.60	1596.60
20	1000	yes	yes	0.73 s	28.38	353.90	399.80
20	1000	yes	no	1.14 s	46.24	350.45	399.80
20	1000	no	yes	1.35 s	26.69	708.15	399.80
20	1000	no	no	2.26 s	46.32	702.20	399.80
20	10000	yes	yes	36.61 s	36.60	4364.40	1695.20
20	10000	yes	no	67.60 s	70.65	4348.85	1695.20
20	10000	no	yes	70.10 s	34.66	8730.45	1695.20
20	10000	no	no	133.03 s	70.66	8698.70	1695.20
20	20000	yes	yes	140.80 s	38.55	8963.15	2757.90
20	20000	yes	no	265.17 s	77.88	8939.40	2757.90
20	20000	no	yes	263.39 s	36.58	17923.10	2757.90
20	20000	no	no	525.18 s	77.90	17873.50	2757.90
20	40000	yes	yes	577.26 s	40.51	18280.80	4566.00
20	40000	yes	no	1164.42 s	85.65	18239.80	4566.00
20	40000	no	yes	1096.81 s	38.47	36566.85	4566.00
20	40000	no	no	2318.33 s	85.64	36480.40	4566.00

VI. ACKNOWLEDGEMENTS

This work was supported by the Vinnova FFI Complex Control Program, grant No. 2015-02309.

REFERENCES

[1] F. Blanchini and S. Miani, *Set-Theoretic Methods in Control*. Birkhäuser, 2008.

[2] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for linear and hybrid systems*, 2015.

[3] T. Alamo, D. M. noz de la Peña, D. Limon, and E. F. Camacho, "Constrained Min-Max Predictive Control: Modifications of the Objective Function Leading to Polynomial Complexity," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 710–714, 2005.

[4] T. B. Blanco, M. Cannon, and B. D. Moor, "On efficient computation of low-complexity controlled invariant sets for uncertain linear systems," *International Journal of Control*, vol. 83, pp. 1339–1346, 2010.

[5] N. Athanasopoulos, G. Bitsoris, and M. Lazar, "Construction of invariant polytopic sets with specified complexity," *International Journal of Control*, vol. 87, 2014.

[6] S. Miani and C. Savorgnan, "MAXIS-G: a software for computing polyhedral invariant sets for constrained LPV systems," in *Conference on Decision and Control*, 2005.

[7] J. C. G. Boot, "On trivial and binding constraints in programming problems," *Management Science*, vol. 8, no. 4, pp. 419–441, 1962.

[8] A. Shelfi, "Reduction of linear inequality constraints and determination of all feasible extreme points," Ph.D. dissertation, 1969.

[9] S. Zionts and J. Wallenius, "Identifying efficient vectors: Some theory and computational results," *Operations Research*, vol. 28, pp. 785–793, 1980.

[10] K. L. Clarkson, "More output-sensitive geometric algorithms," in *Annual Symposium on Foundations of Computer Science*, 1994.

[11] K. Fukuda, B. Gärtner, and M. Szedlak, "Combinatorial redundancy detection," *Annals of Operations Research*, vol. 34, pp. 315–328, 2015.

[12] A. Bemporad, "A Multiparametric Quadratic Programming Algorithm With Polyhedral Computations Based on Nonnegative Least Squares," *IEEE on Automatic Control*, vol. 60, no. 11, pp. 2892–2903, 2015.

[13] A. Marechal and M. Perin, "Efficient elimination of redundancies in polyhedra by raytracing," in *International Conference on Verification, Model Checking, and Abstract Interpretation*, 2017.

[14] P. Gill and W. Murray, "A numerically stable form of the simplex algorithm," *Linear algebra and its applications*, 1973.

[15] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge: University Press, 2004.

[16] N. Andreasson, A. Evgrafov, and M. Patriksson, *Introduction to Continuous Optimization*, 2006.

[17] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, pp. 327–363, 2014.

[18] T. Johnson, C. Kirches, and A. Wächter, "An active-set quadratic programming method based on sequential hot-starts," *SIAM Journal on Optimization*, 2015.

[19] G. B. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963.