Embedded Mixed-Integer Quadratic Optimization Using the OSQP Solver

Bartolomeo Stellato, Vihangkumar V. Naik, Alberto Bemporad, Paul Goulart and Stephen Boyd

***Abstract*— We present a novel branch-and-bound solver for mixed-integer quadratic programs (MIQPs) that efficiently exploits the first-order OSQP solver for the quadratic program (QP) sub-problems. Our algorithm is very robust, requires no dynamic memory allocation and is division-free once an initial factorization is computed. Thus, it suitable for embedded applications with low computing power. Moreover, it does not require any assumption on the problem data such as strict convexity of the objective function. We exploit factorization caching and warm-starting to reduce the computational cost of QP relaxations during branch-and-bound and over repeated solutions of parametric MIQPs such as those arising in embedded control, portfolio optimization, and machine learning. Numerical examples show that our method, using a simple high-level Python implementation interfaced with the OSQP solver, is competitive with established commercial solvers.**

I. Introduction

A. Problem formulation

We are interested in solving the following MIQP

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u, \\ & x_i \in \mathbb{Z}, \quad \forall i \in \mathbb{I} \end{array} \tag{1}$$

with respect to the decision vector $x \in \mathbb{R}^n$. The objective function is defined by the symmetric positive semidefinite matrix $P \in \mathbb{S}^n_+$ and vector $q \in \mathbb{R}^n$, and the linear constraints by the matrix $A \in \mathbb{R}^{m \times n}$ and vectors $l \in (\mathbb{R} \cup \{-\infty\})^m$ and $u \in (\mathbb{R} \cup \{+\infty\})^m$. The set $\mathbb{I}$ denotes the elements of x constrained to take integer values, with $p := |\mathbb{I}|$ their total number. We will refer to the cost function of (1) as $f(x) := \frac{1}{2}x^T P x + q^T x$.

Problems arising in many application domains can be expressed in the form (1), including portfolio optimization [1], [2], [3], hybrid vehicle control [4], regressor selection [5], hybrid model predictive control [6], geolocalization [7], and power systems [8], [9].

Problem (1) is $\mathcal{NP}$-hard in general since it includes mixed-integer linear programs (MILPs) as a special case [10]. Nevertheless, in the last two decades both hardware and software improvements have brought several orders of magnitude improvements in computation time [11], [12]. However, state-of-the art-solvers are still not well-suited for solving MIQPs on embedded platforms with low memory resources.

This work was supported by the People Programme (Marie Curie Actions) of the European Unions Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 607957 (TEMPO).

B. Stellato is with the Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge MA 02139, USA. stellato@mit.edu

P. Goulart is with the University of Oxford, Parks Road, Oxford, OX1 3PJ, UK. {bartolomeo.stellato, paul.goulart}@eng.ox.ac.uk

V. V. Naik and A. Bemporad are with IMT Institute for Advanced Studies Lucca, Piazza S. Francesco 19, 55100 Lucca, Italy. {vihangkumar.naik, alberto.bemporad}@imtlucca.it

S. Boyd is with the Department of Electrical Engineering, Stanford University, Stanford CA 94305, USA. boyd@stanford.edu

B. Solution methods

There are many methods for computing the optimal solution to (1) exactly [13], [14]. When the set of discrete variables is finite, the simplest approach is *exhaustive-search*, consisting of the enumeration of all possible integer combinations. The *branch-and-bound* algorithm instead searches for the optimal solution over a tree by repetitively partitioning the feasible region of integer variables into sub-domains. This technique was first introduced in the 1960s [15] to solve MILPs and later extended to mixed-integer nonlinear programs (MINLPs) [16]. *Branch-and-cut* [17] methods combine the benefits of branch-and-bound with *cutting plane* [18], [19] methods by iteratively introducing additional constraints to reduce the feasible region and thereby the number of nodes explored in the search tree. Other approaches such as *outer approximation* or *generalized Benders' decomposition* exploit the structure of the problem by alternating between the solution of a convex relaxation and of an MILP containing the feasible region. However, *branch-and-bound* is generally considered the most efficient algorithm available to solve problems of the form (1) [20], and is currently implemented in most commercial solvers [21].

Several heuristics have been proposed to compute suboptimal solutions to problem (1) when there is insufficient time or computing power to solve it to optimality. The *relax-and-round* heuristic solves a continuous relaxation of the MIQP and then rounds the fractional components to the closest integers. More advanced heuristics such as the *feasibility pump* [22] search for a solution by iteratively solving linear program (LP). Recently, an heuristic based on the alternating direction method of multipliers (ADMM) [23] has shown promising timing results relative to commercial solvers in computing good quality feasible solutions. A similar method has been proposed together with accelerated dual gradient projection [24]. The main downside of these heuristics is that they are not guaranteed to find a good solution, or even a feasible one.

C. Embedded systems

The main focus of this work is on embedded applications where the available time and computational power are both limited and the same problem is solved many times for varying parameters. In these cases the problem structure can be exploited to accelerate subsequent solutions.

A significant part of the research on embedded optimization to date has focused on tools for convex optimization problems. Examples include the solvers CVXGEN [25], ECOS [26] and FiOrdOs [27], OSQP code generation [28] and the commercial solvers FORCES Pro [29] and ODYS QP [30]. Significant advances were also obtained by applying first-order methods to solve optimal control problems on field-programmable gate arrays (FPGAs) [31], [32].

However, reliable numerical tools are still needed for solving MIQPs on embedded systems. Some progress has been made in developing MIQP solvers narrowly tailored to control applications such as hybrid model predictive control (MPC), including those based on interior-point optimization [33], [29], active set methods [34], [35], and first-order methods [36], [23]. In [35], [24], general branch-and-bound solvers based on nonnegative least squares and dual gradient projection were presented whose performance is competitive with commercial solvers in relatively small problems, but they require strict convexity of the objective function. Hybrid MPC problems usually present a positive semidefinite matrix P because some of the auxiliary integer variables are not penalized. Thus, MIQP solvers requiring strict convexity of the objective function must add a regularization term to P which lowers the solution accuracy. The solver in [35] has been recently extended using the QP solver in [37] to avoid regularization and handle the positive semidefinite case.

D. Contributions

We propose a new robust branch-and-bound algorithm based on the recently developed solver OSQP [38] to compute the solutions to MIQPs of the form (1).

The OSQP solver is a new robust and efficient ADMM-based QP solver written in C. It is able to recognize primal and dual infeasible problems and does not require any assumption on the problem data apart from convexity. The OSQP solver is easily warm-started and thus efficiently employed within branch-and-bound schemes so that only a limited number of iterations are required to solve each sub-problem.

The proposed algorithm is an adaptation of the standard branch-and-bound method to efficiently exploit the linear algebra operations and robustness of the OSQP solver. Our approach requires only a single quasi-definite matrix factorization that is then cached and reused in all ADMM iterations of all QP sub-problems solved during branch-and-bound. Moreover, the same factorization together with the current optimal solution can also be reused in subsequent optimizations arising in parametric optimization. In contrast to state-of-the-art MIQP algorithms based on first-order methods such as [24], our approach does not require any assumption on problem data such as specific structure, positive definiteness of matrix P or linear independence of the constraints.

Our method is suitable for embedded systems since, following an initial matrix factorization that can be performed offline, it does not require dynamic memory allocation and is division-free.

We prototyped the algorithm in Python interfacing to the fast OSQP solver binaries. The code together with the examples is available at [39]. Numerical results show that our approach is faster than commercial packages in solving small/medium-scale MIQPs arising in embedded optimization.

II. MIQP SOLVER BASED ON OSQP

The branch-and-bound algorithm computes the optimal solution $x^\star$ by exploring the integer combinations in a tree [40],[14, Sec 3.1]. The search is performed by repeatedly solving QPs of the form

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u, \\ & \underline{x}_i \leq x_i \leq \overline{x}_i, \quad \forall i \in \mathbb{I} \end{array} \qquad (\mathrm{QP}(\underline{x}, \overline{x}))$$

Each $\mathrm{QP}(\underline{x}, \overline{x})$ is uniquely identified by the lower and upper bounds $(\underline{x}, \overline{x})$ imposed on the integer variables.

The algorithm starts by solving the continuous relaxation of (1) at the root node, *i.e.*, $\mathrm{QP}(-\infty, \infty)$, obtaining a solution $\tilde{x}$ and a lower bound $f(\tilde{x})$. If $\mathrm{QP}(-\infty, \infty)$ is primal or dual infeasible, then MIQP (1) is also primal or dual infeasible, respectively. If the solution satisfies all of the integer restrictions it is said to be *integer feasible* and is also a global solution of (1). Otherwise, the solution is said to be *fractional*. In that case, the algorithm searches over a tree whose nodes are sub-problems of the form $\mathrm{QP}(\underline{x}, \overline{x})$ and whose edges are the branching decisions. It is typically unnecessary to search over all possible integer combinations, and the popularity of branch-and-bound is due in part to the fact that some subtrees can be pruned before exploration. We next briefly describe the branching and pruning process.

Given a problem $\mathrm{QP}(\underline{x}, \overline{x})$ with fractional solution, we pick a non-integer element $\tilde{x}_i$, $i \in \mathbb{I}$, and branch creating left $(-)$ and right $(+)$ child nodes with the same variable bounds $(\underline{x}, \overline{x})$ as the parent. Then we set

$$(\underline{x}_i^-, \overline{x}_i^-) = (\underline{x}_i, \lfloor \tilde{x}_i \rfloor) \qquad (\underline{x}_i^+, \overline{x}_i^+) = (\lceil \tilde{x}_i \rceil, \overline{x}_i). \qquad (2)$$

The lower bounds of both children are initialized with the lower bound of their parent, so that lower bounds are monotonically non-decreasing with tree depth. In practice, we do not maintain the whole tree in memory but keep only the leaves in the heap $\mathcal{H}$.

Pruning rules allow us to discard tree branches based on the optimality and feasibility of the current node $\mathrm{QP}(\underline{x}, \overline{x})$. Let us denote the upper bound on the objective value as U and set it to ∞ at initialization. Pruning of branches occurs in three cases:

- If the current node is infeasible, then we prune the subtree since it contains only infeasible problems.
- If the optimal value $f(\tilde{x})$ of the current node is worse than the current upper bound U, *i.e.*, $f(\tilde{x}) > U$, then we prune the node since any integer solution in the subtree will not be better than the current best one.
- If the solution $\tilde{x}$ to the current node is integer feasible, then we can prune the entire subtree because it cannot contain better feasible solutions than $\tilde{x}$. Moreover, if it improves the current upper bound, that is $f(\tilde{x}) < U$, then we can update the optimal solution and the upper bound with $x^\star \leftarrow \tilde{x}$ and $U \leftarrow f(\tilde{x})$.

Since a good upper bound U allows the pruning of unnecessary branches, it is useful to find a good quality feasible solution as quickly as possible. In order to do so, at each iteration we select a vector $\hat{x}$ whose elements $\hat{x}_i \; i \in \mathbb{I}$ are integer from the solution $\tilde{x}$ of the current node. If $\hat{x}$ satisfies the linear constraints $l \leq A\hat{x} \leq u$, then it is feasible for the original problem (1). If, in addition, $f(\hat{x})$ improves the current upper bound U, we can update the best known solution and the upper bound with $x^\star \leftarrow \hat{x}$ and $U \leftarrow f(\hat{x})$.

The complete algorithm description can be found in Algorithm 1. Note that if the QP solver used to solve the sub-problems is able to generate dual-feasible solutions, one can stop solving the relaxation prematurely as soon as the corresponding dual cost is larger than the best known upper bound U [20], [35], [24], [34], [41].

Algorithm 1 MIQP branch-and-bound

initialize $U \leftarrow \infty$, $\mathcal{H} \leftarrow \mathrm{QP}(-\infty, \infty)$
while $\mathcal{H} \neq \emptyset$ **do**
  pick and remove $\mathrm{QP}(\underline{x}, \overline{x})$ from $\mathcal{H}$
  $\tilde{x}$, $f(\tilde{x}) \leftarrow$ solve $\mathrm{QP}(\underline{x}, \overline{x})$
  if $\mathrm{QP}(\underline{x}, \overline{x})$ is *infeasible* **then**
    prune current node
  else if $f(\tilde{x}) > U$ **then**
    prune current node
  else if $\tilde{x}$ is *integer feasible* **then**
    $U \leftarrow f(\tilde{x})$, $x^\star \leftarrow \tilde{x}$
    fathom nodes in $\mathcal{H}$ with lower bound $> U$
  else
    choose integer $\hat{x}$ from $\tilde{x}$
    if $\hat{x}$ is feasible and $f(\hat{x}) < U$ **then**
      $U \leftarrow f(\hat{x})$, $x^\star \leftarrow \hat{x}$
      fathom nodes in $\mathcal{H}$ with lower bound $> U$
    branch node $\mathrm{QP}(\underline{x}, \overline{x})$

A. Strategic decisions

Algorithm 1 presents three degrees of freedom that can substantially change its performance depending on the problem instance.

1) Tree exploration: The way we pick the next node $\mathrm{QP}(\underline{x}, \overline{x})$ to explore from the heap $\mathcal{H}$ determines how the tree exploration progresses. The two most common strategies are *best-bound* and *depth-first* [14]. *Best-bound* always chooses the node with the best lower bound, usually resulting in a small number of nodes explored. Its drawback is that large amounts of memory are required in general because, in the worst-case, the whole tree is searched before a feasible solution is found. In contrast, *depth-first* always picks the deepest node (or one of the deepest nodes) in a tree, with the advantage that the heap $\mathcal{H}$ is kept as small as possible. However, *depth-first* search typically visits more total nodes than the *best-bound* approach. In this work we use a hybrid approach where *depth-first* is carried out until a feasible solution is found. Then *best-bound* is used to minimize the number of visited nodes.

2) Branching variable selection: When branching we must choose amongst the candidate fractional elements of $\tilde{x}$ to determine bounds for the new nodes as in (2). The goal is to maximize the increase in the objective function with the branching so that it becomes easier to fathom nodes in the subtrees. In this work we use a *maximum fractional part branching* rule, *i.e.*, we select the variable with maximum integer violation [42], [20]. More sophisticated branching heuristics like *strong branching* or *pseudocost branching* [14] can be chosen to predict the increase in the value function in the child nodes, but it is not clear that they improve the practical performance [20], [43] and we do not employ them.

3) Compute an integer solution: Each time we compute a relaxed solution $\tilde{x}$, we also search for an integer feasible solution $\hat{x}$ with the help of a heuristic. The main idea is to quickly find good upper bounds allowing us to prune as many nodes as possible [14]. We use *nearest-neighbor rounding* by computing a simple rounding of the fractional elements of $\tilde{x}$. However, the rounded vector produced by this method may not be feasible, and more sophisticated heuristics can be applied to ensure that a feasible solution will be found; see *MILP-based rounding* [44] or the *feasibility pump* [22]. We do not to use these heuristics because they require solving several LPs that might be more expensive than just progressing in the tree search, since the required matrix factorization would be different than the one in Algorithm 2. Note that in the case of purely binary variables, tailored schemes such as *sum-up-rounding* could also be applied [45].

B. The OSQP solver

In order to solve the sub-problems in Algorithm (1), we require an efficient QP solver able both to compute optimal solutions or to certify infeasibility. Moreover, the QP solver should support warm-starting of the solution from parent to children nodes to exploit the structural similarity between sub-problems.

The OSQP solver [38] is a new efficient solver based on the ADMM [46] that fits all the requirements for being efficiently embedded into a branch-and-bound scheme. The OSQP solver computes the solution to QPs of the form

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u, \end{array} \tag{3}$$

where the parameters P, q, A, l, u are the same as in problem (1). Note that every sub-problem ($\mathrm{QP}(\underline{x}, \overline{x})$) can be written in this form by including the bounds $\underline{x}$ and $\overline{x}$ in the linear constraints.

The OSQP solution procedure can be found in Algorithm 2. Scalars $\rho, \sigma > 0$ are the *step-size parameters*

Algorithm 2 OSQP solver

given initial values x^0, z^0, y^0 and parameters ρ, σ, α
repeat
  solve $\begin{bmatrix} P + \sigma I & A^T \\ A & -\frac{1}{\rho} I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \frac{1}{\rho} y^k \end{bmatrix}$
  $\tilde{z}^{k+1} \leftarrow z^k + \frac{1}{\rho}(\nu^{k+1} - y^k)$
  $x^{k+1} \leftarrow \alpha \tilde{x}^{k+1} + (1-\alpha) x^k$
  $z^{k+1} \leftarrow \Pi\left(\alpha \tilde{z}^{k+1} + (1-\alpha) z^k + \frac{1}{\rho} y^k\right)$
  $y^{k+1} \leftarrow y^k + \rho\left(\alpha \tilde{z}^{k+1} + (1-\alpha) z^k - z^{k+1}\right)$
until termination conditions are satisfied

and $\alpha \in (0,2)$ is the *relaxation-parameter*. The operator Π is the projection onto the element-wise separable set $\{z \in \mathbb{R}^m \mid l \leq z \leq u\}$ with the closed-form solution

$$\Pi(x) = \max\left(\min\left(z, u\right), l\right).$$

The most expensive part of the algorithm is the solution of the linear system in the first step that is via permuted sparse LDL^T factorization [47] followed by forward and backward substitution. Since the coefficient matrix in this linear system is quasi-definite it always has a well-defined LDL^T factorization, with L being a lower triangular matrix with unit diagonal elements and D a diagonal matrix with nonzero diagonal elements [48]. Note that the *matrix does not change* as it does not depend on the bounds $\underline{x}$ and $\overline{x}$ of the sub-problems defining each node of the branch-and-bound tree. It must therefore be factorized only once before the first iteration. This factorization is then cached and used in all subsequent ADMM iterations. In addition, it does not change when we solve problem (1) for varying vectors q, l, u. The other algorithm steps are computationally much cheaper and involve only scalar-vector multiplications, vector additions and element-wise projections (clipping).

At each iteration, the algorithm produces iterates (x^k, z^k, y^k) whose primal and dual residuals are defined as

$$\begin{aligned} r^k_{\text{prim}} &= Ax^k - z^k \\ r^k_{\text{dual}} &= Px^k + q + A^T y^k. \end{aligned}$$

If problem (3) is solvable, the residuals converge to zero as $k \to \infty$ [46]. The algorithm stops when the Euclidean norm of the residuals is below predefined tolerances $\varepsilon_{\text{prim}} > 0$ and $\varepsilon_{\text{dual}} > 0$. Note that $\varepsilon_{\text{prim}}, \varepsilon_{\text{dual}}$ are often chosen relative to the scaling of problem iterates, see [46, Sec. 3.3].

If the problem is primal infeasible, the algorithm produces a vector $v \in \mathbb{R}^m$ serving as a certificate of infeasibility, *i.e.*,

$$A^T v = 0, \quad u^T v_+ + l^T v_- < 0,$$

where $v_+ = \max(v, 0) > 0$ and $v_- = \min(v, 0) < 0$. On the other hand, if the problem is dual infeasible, the algorithm generates a vector $s \in \mathbb{R}^n$ certifying dual infeasibility, *i.e.*,

$$Ps = 0, \quad q^T s < 0, \quad (As)_i = \begin{cases} 0 & l_i, u_i \in \mathbb{R} \\ \geq 0 & u_i = +\infty, l_i \in \mathbb{R} \\ \leq 0 & l_i = -\infty, u_i \in \mathbb{R} \end{cases}.$$

For more details we refer the reader to [38].

III. Numerical results

Our algorithm, named miOSQP, has been implemented in Python and interfaced to the OSQP compiled binaries from [38]. Timing benchmarks are compared to GUROBI Optimizer v7.0.2 [21] with the default options on a Macbook Pro 2.8GHz Intel Core i7 with 16GB RAM running Python 3.5. In addition, both algorithms are executed single-threaded for fairness. The code and all benchmark examples are available at [39].

TABLE I
Timings in ms for random MIQPs with varying n, m and q

			miOSQP			GUROBI	
n	m	p	t_{avg}	$t_{\max}$	$t_{\text{OSQP}}[\%]$	t_{avg}	$t_{\max}$
10	5	2	6.81	54.72	4.60	1.94	3.22
10	100	2	2.40	3.79	27.41	12.43	17.49
50	25	5	4.69	7.95	39.31	20.19	22.18
50	200	10	52.97	142.93	74.06	110.15	149.42
100	50	2	6.05	9.10	59.16	53.06	72.38
100	200	15	103.03	343.42	68.21	258.66	390.91
150	100	5	38.06	61.45	63.10	263.04	340.43
150	300	20	349.97	700.15	68.40	932.16	1327.71

A. Random MIQPs

We generated random MIQPs with varying dimensions n, m and number of integer variables p. The entries of P are computed as $P = MM^T$ where $M \in \mathbb{R}^{n \times n}$ is generated from the uniform distribution $\mathcal{U}(0,1)$ with 70% nonzero elements and the linear part of the cost q with the normal distribution $\mathcal{N}(0,1)$. The constraints are generated as $A \sim \mathcal{U}(0,1)$, $l \sim \mathcal{U}(0,1) - 2$ and $l \sim \mathcal{U}(0,1) + 2$. Each problem is solved 10 times, from which we compute both the average and the maximum execution times. The results are shown in Table I. miOSQP usually outperforms GUROBI with up to $7\times$ improvements in computation time. We also reported the percentage of the miOSQP computation time used by the inner QP solutions. Depending on the problem size, the Python overhead can be significant. For example, when $(n, m, p) = (10, 5, 2)$ OSQP takes only $4.6\,\%$ of the total computation time and GUROBI turns out to be faster. This suggests that further speedups could be obtained by using a low-level branch-and-bound implementation.

B. Power converter control

We consider the hybrid system model of a three-level voltage source converter driving a medium-voltage induction machine [9]. The system dynamics can be described as a discrete-time linear system with integer inputs

$$x_{t+1} = Ax_t + Bu_t$$

where the state x_t has dimension 12 representing the internal motor currents and voltages [9]. The input vector has dimension 6 including the three semiconductor devices positions with possible values $\{-1, 0, 1\}$ and three additional binary components required to model the system, $u_t \in \{-1, 0, 1\}^3 \times \{0, 1\}^3$. We would like to compute the optimal inputs so that the internal currents track the reference sinusoids. This can be obtained by solving:

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{T} \gamma^t l(x_t) + \gamma^T V(x_T) \\ \text{subject to} & x_{t+1} = Ax_t + Bu_t \\ & x_0 = x_{\text{init}} \\ & \|u_t - u_{t-1}\|_\infty \leq 1 \\ & u_t \in \{-1, 0, 1\}^3 \times \{0, 1\}^3 \end{array} \tag{4}$$

where $\gamma \in (0,1)$ is a discount factor and $l(x_t)$ a quadratic state penalty cost penalizing the currents' deviation from the reference sinusoids. Note that the reference sinusoids are embedded into the system dynamics so that (4) becomes

a regulation problem. The third constraint is enforced to avoid shoot-through in the inverter positions (changes from -1 to 1 or vice-versa) that could damage the components. The states x_t are for $t = 0, 1, \ldots, T$ and the inputs u_t for $t = 0, 1, \ldots, T-1$. The initial state is denoted x_{init}. The function V represents the final stage cost approximating an infinite horizon cost.

Since the computational cost grows exponentially with the horizon length T, the authors of [9] computed a final stage cost V using the approximate dynamic programming (ADP) approach to shorten the horizon length to $T = 1$ or 2. This allows good control performance while keeping the number of input combinations manageable to enable *exhaustive search*. This technique becomes prohibitive for longer horizons because the number of input combinations grows exponentially with T. By using our proposed approach we show that the computation time can be kept in the ms time-scale even with longer horizons.

By eliminating the states, problem (4) can be rewritten in the form (1) with all variables being integer, *i.e.*, $n = p$, and solved with miOSQP.

We performed closed-loop simulations for horizons $T = 1, 2, \ldots, 5$ for 3 periods each; one for reaching the steady state behavior and two for the actual simulation. Each period has 800 time steps. The computation times are averaged over all the solutions for each simulation. Both GUROBI and our algorithm were warm-started with the solution at the previous time step. The timings are shown in Figure 2. The currents and inputs behavior are shown in Figure 1.

The timing comparison shows a consistently better behavior of miOSQP compared to GUROBI: approximately 3x improvements across all the horizon lengths. Note that the problem has 27 integer feasible input combinations per stage which, for horizon 5 amounts to a total of $14,348,907$ worst case number of nodes to be evaluated. However, miOSQP always computes the optimal input after searching only a few hundred nodes. In addition, thanks to the warm-starting capabilities of OSQP only 30 ADMM iterations are required on average to solve each individual QP. Note that the average computation time required to solve the QP sub-problems amounts to $30\,\%$ of the miOSQP solution time.

IV. Conclusion

This paper has proposed a new MIQP algorithm based on branch-and-bound combined to the OSQP solver. Thanks to factorization caching and warm starting, our method is able to efficiently compute globally optimal solutions. Moreover, it is very robust and division-free and thereby suitable for embedded architectures. Numerical examples show that our method with a simple, high-level implementation shows better timing benchmarks than commercial solvers for small to medium-size problems arising in embedded applications.

The algorithm still has considerable scope for performance improvement. In particular, a C implementation will greatly reduce the computation time. From the Python overhead seen in our examples, we expect to gain at least $50\,\%$ time reduction with a tailored serial implementation. Parallel solution of the branch-and-bound nodes will also bring significant speedups. In addition, advanced branching techniques together with cutting planes generation and heuristics will reduce the number of visited nodes. We could also obtain performance improvements from premature pruning of the branch-and-bound nodes in case the dual cost during the sub-problems iterations exceeds the best known upper bound. At the moment our algorithm does not exploit this feature because OSQP produces dual feasible solutions only at convergence and not during the iterations. Finally, it would be interesting to compare miOSQP with other problem sets such as the QPLIB [49] or the portfolio optimization problems in [3].

References

[1] D. Bienstock, "Computational study of a family of mixed-integer quadratic programming problems," *Mathematical Programming*, vol. 74, no. 2, pp. 121–140, 1996.

[2] P. Bonami and M. A. Lejeune, "An exact solution approach for portfolio optimization problems under stochastic and integer constraints," *Operations Research*, vol. 57, no. 3, pp. 650–670, 2009.

[3] J. P. Vielma, S. Ahmed, and G. L. Nemhauser, "A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs," *INFORMS Journal on Computing*, vol. 20, no. 3, pp. 438–450, 2008.

[4] N. Murgovski, L. Johannesson, J. Sjberg, and B. Egardt, "Component sizing of a plug-in hybrid electric powertrain via convex optimization," *Mechatronics*, vol. 22, no. 1, pp. 106 – 120, 2012.

[5] D. Bertsimas, A. King, and R. Mazumder, "Best subset selection via a modern optimization lens," *Ann. Statist.*, vol. 44, no. 2, pp. 813–852, 04 2016.

[6] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.

[7] P. Teunissen, "The least-squares ambiguity decorrelation adjustment: a method for fast GPS integer ambiguity estimation," *Journal of Geodesy*, vol. 70, no. 1-2, pp. 65–82, 1995.

[8] T. Geyer and D. E. Quevedo, "Multistep finite control set model predictive control for power electronics," *IEEE Transactions on Power Electronics*, vol. 29, no. 12, pp. 6836–6846, Dec 2014.

[9] B. Stellato, T. Geyer, and P. J. Goulart, "High-speed finite control set model predictive control for power electronics," *IEEE Transactions on Power Electronics*, vol. 32, no. 5, pp. 4007–4020, May 2017.

[10] G. Nemhauser and L. Wolsey, *Computational Complexity*. John Wiley & Sons, Inc., 1988, pp. 114–145.

[11] R. E. Bixby, "A brief history of linear and mixed-integer programming computation," *Documenta Mathematica*, pp. 107–121, 2010.

[12] G. Nemhauser, "Integer programming: a global impact," in *EURO, INFORMS, Rome, Italy*, 2013.

[13] Lee, Jon and Leyffer, Seven, *Mixed Integer Nonlinear Programming*, ser. The IMA Volumes in Mathematics and its Applications, J. Lee and S. Leyffer, Eds. New York, NY: Springer New York, 2012, vol. 154.

[14] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, "Mixed-integer nonlinear optimization," *Acta Numerica*, vol. 22, pp. 1–131, Apr. 2013.

[15] A. H. Land and A. G. Doig, *An Automatic Method for Solving Discrete Programming Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 105–132.

[16] R. J. Dakin, "A tree-search algorithm for mixed integer programming problems," *The Computer Journal*, vol. 8, no. 3, pp. 250–255, 1965.

[17] R. A. Stubbs and S. Mehrotra, "A branch-and-cut method for 0-1 mixed convex programming," *Mathematical Programming*, vol. 86, no. 3, pp. 515–532, 1999.

[18] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs," *Bulletin of the American Mathematical Society*, vol. 64, no. 5, pp. 275–278, 09 1958.

[19] V. Chvàtal, W. Cook, and M. Hartmann, "On cutting-plane proofs in combinatorial optimization," *Linear Algebra and its Applications*, vol. 114, pp. 455 – 499, 1989.

[20] R. Fletcher and S. Leyffer, "Numerical experience with lower bounds for MIQP branch-and-bound," *SIAM Journal on Optimization*, vol. 8, no. 2, pp. 604–616, 1998.

[21] Gurobi Optimization Inc., "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com

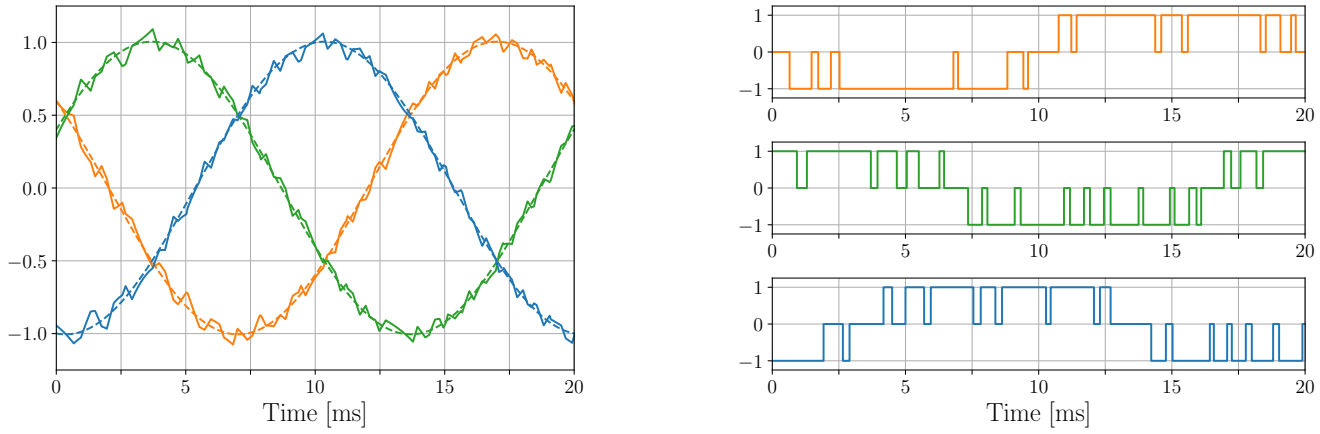

Fig. 1. Power converter simulation results with miOSQP solver and $N = 3$

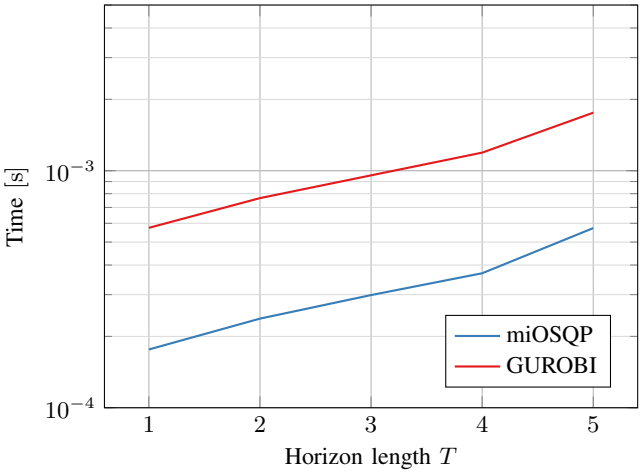

Fig. 2. Average power converter MIQP solution time comparison between miOSQP and GUROBI

[22] M. Fischetti, F. Glover, and A. Lodi, "The feasibility pump," *Mathematical Programming*, vol. 104, no. 1, pp. 91–104, 2005.

[23] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad, "A simple effective heuristic for embedded mixed-integer quadratic programming," *International Journal of Control*, pp. 1–11, 2017.

[24] V. V. Naik and A. Bemporad, "Embedded mixed-integer quadratic optimization using accelerated dual gradient projection," in *20th IFAC World Congress*, Toulouse, France, 2017.

[25] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[26] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *2013 European Control Conference (ECC)*, July 2013, pp. 3071–3076.

[27] F. Ullmann, "FiOrdOs: A Matlab Toolbox for C-Code Generation for First Order Methods," Master's thesis, ETH Zürich, 2011.

[28] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, "Embedded code generation using the OSQP solver," in *IEEE Conference on Decision and Control (CDC)*, Dec. 2017.

[29] A. Domahidi and J. Jerez, "FORCES Professional," embotech GmbH (http://embotech.com/FORCES-Pro), Jul. 2014.

[30] G. Cimini, A. Bemporad, and D. Bernardini, "ODYS QP Solver," ODYS S.r.l. (http://odys.it/qp), Sep. 2017.

[31] J. Jerez, P. Goulart, S. Richter, G. Constantinides, E. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014.

[32] M. Rubagotti, P. Patrinos, A. Guiggiani, and A. Bemporad, "Real-time model predictive control based on dual gradient projection: Theory and fixed-point FPGA implementation," *International Journal of Robust and Nonlinear Control*, vol. 26, no. 15, pp. 3292–3310, 2016.

[33] D. Frick, A. Domahidi, and M. Morari, "Embedded optimization for mixed logical dynamical systems," *Computers & Chemical Engineering*, vol. 72, pp. 21 – 33, 2015.

[34] D. Axehill and A. Hansson, "A mixed integer dual quadratic programming algorithm tailored for MPC," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec 2006, pp. 5693–5698.

[35] A. Bemporad, "Solving mixed-integer quadratic programs via nonnegative least squares," *IFAC-PapersOnLine*, vol. 48, no. 23, pp. 73 – 79, 2015.

[36] D. Frick, J. L. Jerez, A. Domahidi, A. Georghiou, and M. Morari, "Low-complexity iterative method for hybrid MPC," *ArXiv e-prints*, 2016.

[37] A. Bemporad, "A numerically stable solver for positive semi-definite quadratic programs based on nonnegative least squares," *IEEE Transactions on Automatic Control*, vol. 63, no. 2, pp. 525–531, 2018.

[38] B. Stellato and G. Banjac, "OSQP: An operator splitting solver for quadratic programs," *GitHub*, 2017. [Online]. Available: https://github.com/oxfordcontrol/osqp

[39] B. Stellato, "MIQP solver based on osqp," *GitHub*, 2017. [Online]. Available: https://github.com/OxfordControl/miosqp

[40] S. Boyd and J. Mattingley, "Branch and Bound Methods," *Lecture notes*, 2010. [Online]. Available: https://stanford.edu/class/ee364b/lectures/bb_notes.pdf

[41] C. Buchheim, M. De Santis, S. Lucidi, F. Rinaldi, and L. Trieu, "A feasible active set method with reoptimization for convex quadratic mixed-integer programming," *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1695–1714, 2016.

[42] R. Breu and C.-A. Burdet, *Branch and bound experiments in zero-one programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1974, pp. 1–50.

[43] O. K. Gupta and A. Ravindran, "Branch and bound experiments in convex nonlinear integer programming." *Management Science*, vol. 31, no. 12, pp. 1533–1546, 1985.

[44] G. Nannicini and P. Belotti, "Rounding-based heuristics for nonconvex MINLPs," *Mathematical Programming Computation*, vol. 4, no. 1, pp. 1–31, 2012.

[45] S. Sager, M. Jung, and C. Kirches, "Combinatorial integral approximation," *Mathematical Methods of Operations Research*, vol. 73, no. 3, p. 363, 2011.

[46] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[47] T. Davis, *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.

[48] R. Vanderbei, "Symmetric quasi-definite matrices," *SIAM Journal on Optimization*, vol. 5, no. 1, pp. 100–113, 1995.

[49] F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann, N. Sahinidis, S. Vigerske, and A. Wiegele, "QPLIB: A library of quadratic programming instances," February 2017. [Online]. Available: http://www.optimization-online.org/DB_HTML/2017/02/5846.html