Noname manuscript No.
(will be inserted by the editor)

Reformulation versus cutting-planes for robust optimization

A computational study

Dimitris Bertsimas · Iain Dunning · Miles Lubin

the date of receipt and acceptance should be inserted later

Abstract Robust optimization (RO) is a tractable method to address uncertainty in optimization problems where uncertain parameters are modeled as belonging to uncertainty sets that are commonly polyhedral or ellipsoidal. The two most frequently described methods in the literature for solving RO problems are reformulation to a deterministic optimization problem or an iterative cutting-plane method. There has been limited comparison of the two methods in the literature, and there is no guidance for when one method should be selected over the other. In this paper we perform a comprehensive computational study on a variety of problem instances for both robust linear optimization (RLO) and robust mixed-integer optimization (RMIO) problems using both methods and both polyhedral and ellipsoidal uncertainty sets. We consider multiple variants of the methods and characterize the various implementation decisions that must be made. We measure performance with multiple metrics and use statistical techniques to quantify certainty in the results. We find for polyhedral uncertainty sets that neither method dominates the other, in contrast to previous results in the literature. For ellipsoidal uncertainty sets we find that the reformulation is better for RLO problems, but there is no dominant method for RMIO problems. Given that there is no clearly dominant method, we describe a hybrid method that solves, in parallel, an instance with both the reformulation method and the cutting-plane method. We find that this hybrid approach can reduce runtimes to 50% to 75% of the runtime for any one method and suggest ways that this result can be achieved and further improved on.

Keywords Robust Optimization · Computational Benchmarking

D. Bertsimas, I. Dunning, L. Lubin
Operations Research Center, Massachusetts Institute of Technology,
77 Massachusetts Avenue, Cambridge, MA 02139, USA
E-mail: dbertsim@mit.edu, idunning@mit.edu, mlubin@mit.edu

1 Introduction

Robust optimization (RO) has emerged as both an intuitive and computationally tractable method to address the natural question of how to handle uncertainty in optimization problems. In RO the uncertain parameters in a problem are modeled as belonging to an *uncertainty set* (see, e.g. [1] or [4] for a survey). A wide variety of uncertainty sets have been described in the literature, but polyhedral and ellipsoidal sets are by far the most popular types. Constraints with uncertain parameters must be feasible for all values of the uncertain parameters in the uncertainty set, and correspondingly the objective value is taken to be the worst case value over all realizations of the uncertain parameters. For this reason RO problems, in their original statement, typically have an infinite number of constraints and cannot be solved directly. The most common approach to date for solving them in the literature is to use duality theory to *reformulate* them as a deterministic optimization problems that may have additional variables, constraints, and even change problem class (e.g. a robust linear optimization (RLO) problem with an ellipsoidal uncertainty set becomes a second-order cone problem (SOCP) [2]. Another method, used less frequently, is an iterative *cutting-plane* method (e.g. [19]) that repeatedly solves a relaxed form of the RO problem with a finite subset of the constraints, checks whether any constraints would be violated for some value of the uncertain parameters, adds them if so, and re-solves until no violated constraint exists.

There is little guidance in the literature about which method should be used for any given RO problem instance. We are aware of one computational study by Fischetti and Monaci [11] that directly compares the two methods for RLO and robust mixed-integer optimization (RMIO) problems with a polyhedral uncertainty set and finds that the cutting-plane method is superior for RLO and that reformulations are superior for RMIO. They do not attempt to indicate whether this result is statistically significant, and do not experiment with different variants of the cutting-plane method.

Rather than just consider the question of which method is superior, we consider a different question in this paper that we feel is more relevant for both practitioners and researchers, and acknowledges the ever-increasing popularity of parallel computing. We draw inspiration from modern solvers such as Gurobi [15] which, by default, simultaneously solve an LO problem with both the simplex method and an interior point method and return a solution whenever one of the methods terminates - effectively making the runtime the minimum of the two methods' individual runtimes. This can naturally be extended to the RO case: we can solve an RO problem with both the reformulation method and the cutting plane method simultaneously, and the runtime is defined by the first method to finish. We do not implement in this paper a general-purpose tool which does this simultaneous solve automatically, but the capability to do so would be a natural extension of the capabilities of an RO modeling system, such as ROME [14].

The contributions of this paper are as follows:

1. We provide a precise description of the application of the cutting-plane method to RMIO problems, including an identification of the added complexities over applying the method to RLO problems, and the variants that should be considered by those seeking to implement the method.
2. We replicate the findings of [11] over an enlarged collection of instances, and extend them by evaluating the performance of the reformulation and cutting-plane methods for RO problems with ellipsoidal uncertainty sets. We use more appropriate metrics and employ statistical techniques to more precisely characterize the differences in performance between methods.
3. We enhance the performance of the cutting-plane method for RMIO problems by designing rules on when new cutting planes should be added, and evaluate a new heuristic to run alongside the MIO solver to improve runtimes.
4. Finally we describe and evaluate our proposed hybrid method which uses both methods in parallel and stops when either method finishes. We discuss implementation considerations and possible extensions.

Structure of the paper. In Section 2, we detail the precise RO problem and uncertainty sets we evaluate. We discuss the details of the different cutting-plane algorithms and the possible variations in when new generated constraints should be added. We provide both the intuition and details for a new heuristic for RMIO problems. Section 3 explains the experimental setup, sources of data and the combinations of instance, parameters, and methods we evaluated. We analyze the results in depth to compare the performance of the two methods and their variants on both RLO and RMIO problems. Section 4 investigates the proposed hybrid method's performance relative to the individual methods and its implementation. Finally Section 5 summarizes the results.

2 Problem and Method Descriptions

We will consider the following general linear RO problem

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & \tilde{a}_1^T x \leq b_1 \quad \forall \tilde{a}_1 \in U_1 \\ & \vdots \\ & \tilde{a}_m^T x \leq b_m \quad \forall \tilde{a}_m \in U_m \\ & x \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}, \end{aligned} \tag{1}$$

where n_1 and n_2 are nonnegative integers, $b_1, \ldots, b_m$ are given scalars, c is a given vector, and $U_1, \ldots, U_m$ are the uncertainty sets for each constraint. We will denote uncertain values using a tilde, e.g. $\tilde{a}$. In row i a subset J_i of the coefficients are said to be subject to uncertainty, and all uncertain coefficients have a *nominal* value a_{ij} around which they may vary. We define the *nominal problem* to be the optimization problem identical to the above

with the exception that all coefficients are at their nominal values with no uncertainty, that is $\tilde{a}_{ij} = a_{ij}$ for all i and j. Later in this section we will detail the properties of the two types of uncertainty sets we consider in this paper - the polyhedral set introduced in [5] and the ellipsoidal set introduced in [2]. These sets are both simple to describe but have substantially different deterministic reformulations and algorithms to calculate a new cutting-plane.

Before presenting the specific details of the polyhedral and ellipsoidal uncertainty sets (in Section 2.3 and Section 2.4 respectively) we will detail the cutting-plane methods we considered for both RLO problems and RMIO problems. While similar in spirit, there is a substantial difference in the details of their implementation. The method for RLO problems (Section 2.1) is fairly simple and comparable to other constraint generation methods. The method for RMIO problems (Section 2.2) requires consideration of exactly when in the integer optimization solution process cuts should be added.

2.1 Cutting-plane method for RLO

The idea behind the cutting-plane method for RLO is similar to other row generation methods such as Bender's Decomposition. While there is a constraint for every $\tilde{a}$ in the relevant uncertainty set, only a small subset of these constraints is binding for a robust optimal solution. This suggests that only generating constraints as they are needed to ensure robustness of the solution would be an efficient technique. Given this motivation, the algorithm for RLO problems is as follows:

1. Initialize the *master problem* to be the nominal problem, that is the problem described in Eq. (1) where all $\tilde{a}_{ij}$ are replaced with their nominal values a_{ij}.
2. Solve the master problem, obtaining a solution x^*.
3. For each uncertain row i (that is, rows i such that $J_i \neq \emptyset$)
   (a) Compute $\bar{a} = \arg\max_{\tilde{a} \in U_i} \tilde{a}^T x^*$.
   (b) If $\bar{a}^T x^* > b_i + \epsilon$, add the constraint $\bar{a}^T x \leq b_i$ to the master problem.
4. If no constraints were added then we declare that x^* is the optimal robust solution to the RO and terminate. If any constraints were added we return to Step 2.

The computational practicality of this method relies on the fact that while adding constraints will make the current master problem solution infeasible, we are able to hot-start the optimization (in Step 2) by using the dual simplex method. As a result almost all commercial and open-source LO solvers can be used to solve the master problem. We will defer discussing guarantees of the termination of this algorithm to Sections 2.3 and 2.4.

The constraint feasibility tolerance ϵ is a parameter that may be varied depending on the needs of the application and the solver used, but should generally be an appropriately small value such as 10^{-6}. We implemented the

cutting-plane generation and related logic in C++ and used the commercial solver Gurobi 5.6 [15] to solve the master problem.

Note that a number of improvements to the classical cutting-plane method have been developed, including bundle methods and the analytic center cutting-plane method (ACCPM) [7]. These methods, in general, aim to "stabilize" the sequence of solutions x^*, which can lead to improved theoretical and practical convergence rates. These techniques have been investigated in the context of stochastic programming [20] but to our knowledge not in the context of RO. Nevertheless, a preliminary implementation of a proximal bundle method did not yield significant improvements in computation time in our experiments, and the complexity of integrating these techniques within branch and bound (for RMIO) discouraged further investigation.

2.2 Cutting-plane method for RMIO

The cutting-plane method for RMIO is more complicated than the RLO case, although the principle of only adding constraints as needed is the same. The primary difference is that detailed consideration must be given to the most appropriate time to add new cuts, and that whatever timing is selected guarantees that the final solution is indeed robust.

Perhaps the most obvious place to add cuts is at the root LO relaxation in the branch-and-bound tree. However, it is not sufficient to apply the RLO cutting-plane method to just the root relaxation as the optimal robust integer solution may be affected by constraints that are not active at the fractional solution of the relaxation. Consider this simple example that demonstrates this behavior:

$$\begin{aligned} \max \quad & x \\ \text{subject to} \quad & x \leq 1.5 \\ & u \cdot x \geq 1 \quad \forall u \in [0.9, 1.1] \\ & x \text{ integer.} \end{aligned}$$

The relaxed master problem for this RMIO consists of two constraints, $x \leq 1.5$ and $x \geq 1$ (taking the nominal value of u to be 1), leading to the trivial solution $x^*_{relax} = 1.5$ when we relax integrality. This solution is feasible to all constraints, so no new cutting planes are generated at the root. If we now branch once on the value of x (i.e. $x \leq 1$ or $x \geq 2$) we will obtain the integer solution $x = 1$ which is not feasible with respect to the uncertain constraint - in fact, there is no feasible integer solution to this problem. Although making the root relaxation feasible to the uncertain constraints doesn't guarantee the feasibility of subsequent integer solutions, the addition of constraints at the root may guide the search along more favorable lines - we will analyze this effect in our computational results.

Another possibility is to check for and add new constraints at every node in the branch-and-bound tree, ensuring that the fractional solution at each node is feasible with respect to the uncertain constraints before further branching. This appears to have been the approach taken in [11]. While this will ensure that any integer solution produced by branching will be feasible to the uncertain constraints, there is likely to be a heavy computational cost as we add cuts that are *unnecessary*, in the sense that they are being added to nodes on a branch of the search tree that may not produce an optimal integer solution or improve the bound.

A third possibility is to only check for and add new constraints when we obtain candidate integer-feasible solutions. These constraints are often described as *lazy constraints* as they are not explicitly provided to the solver before they are needed. Instead they are implemented by providing a *callback* to the solver that is responsible for checking the feasibility of all integer solutions and reporting any violated constraints. The solver then discards the candidate integer solution (if it is not feasible) and adds the violated constraints to the active nodes of the branch-and-bound tree. Like adding constraints at each node, using lazy constraints is sufficient to guarantee that the final integer solution is feasible with respect to the uncertain constraints, but given the relatively small number of integer solutions for many MIO problems we decided that it would be a far more efficient method.

One option that is not practical is to completely solve the master problem to provable optimality before adding new constraints. If we wait until the end we are most likely to be adding useful constraints but we pay a huge computational cost as we essentially solve the master problem from scratch multiple times; unlike the case of RLO, no hot-start is possible as even the previous integer solution cannot be provided as an incumbent solution. Lazy constraints are a more appropriate technique as they are added before the solve is completed, allowing us to retain the bound corresponding to the fractional solution.

The final cutting-plane method we implemented is as follows:

1. As for RLO, initialize the *master problem* to be the nominal problem, that is the problem described in Eq. (1) where all $\tilde{a}_{ij}$ are replaced with their nominal values a_{ij}.
2. Apply the RLO cutting-plane method to the root fractional relaxation of the RMIO master problem until the fractional solution is feasible with respect to all uncertain constraints. Note that while we may generate unnecessary cuts with respect to the final integer solution, we may also gain by requiring less cuts later on when we have progressed further down the tree.
3. The MIO solver now begins the branch-and-bound process to solve the master problem.
   - Whenever an integer solution is found we will check all uncertain constraints to see if any are violated by the candidate solution. If any violations are detected we report these new lazy constraints to the MIO

solver, which will discard the candidate integer solution. This ensures that only cuts active at integer solutions are added, and that any integer solution will be feasible with respect to the uncertain constraints. If no constraints are violated, the MIO solver will accept the integer solution as its incumbent solution, and the integrality gap will be evaluated relative to this solution.
- If we do find that the integer solution violates one of the uncertain constraints, we will apply a heuristic that will try to obtain a feasible integer solution using this candidate integer solution as a starting point. For more details, see Section 2.2.1.

We used Gurobi 5.6 [15] to solve the MIO problem, with all other logic implemented in C++.

2.2.1 A heuristic to repair infeasible candidate integer solutions

The MIO solver will find multiple integer solutions as it attempts to solve the master problem to provable optimality. When it does so we will check whether any new constraints can be generated from the uncertainty sets that would make the candidate integer solution infeasible. If we do find any, we add these lazy constraints to the master problem and declare the found integer solution to be infeasible. The solver will discard the solution and keep on searching for another integer solution.

We hypothesize that in many cases these integer solutions that violated the uncertain constraints can be made feasible with respect to these constraints *solely by varying the value of the continuous variables*. The solver cannot explore this possibility by itself, as it is unaware of the uncertain constraints until we provide them. This may lead to possibly high-quality integer solutions being rejected and only re-discovered later in the search process. To address this issue we developed a heuristic that attempts to *repair* these integer solutions:

1. Our input is an integer solution x^* to the master problem that violates one or more uncertain constraints. Create a duplicate of the master problem, called the *subproblem*.
2. Fix the values of all integer variables in the subproblem to their values in the integer solution x^*. It follows that the subproblem is now equivalent to the master problem for a RLO problem.
3. Using the cutting-plane method for RLO problems to solve the subproblem.
4. If a solution to the subproblem is found, then this solution is a valid integer solution for the original problem that satisfies all constraints, uncertain and deterministic. Report this solution to the MIO solver so that it may update its incumbent solution and bounds.
5. If no solution can be found for the subproblem then we take no action, and conclude that we must change the values of integer variables in order to obtain a robust integer solution.

We take a number of steps to improve the computational tractability of this heuristic. For example, we keep only one copy of the LO subproblem in

memory and use it for all runs of the heuristic. This avoids the overhead of creating multiple copies, and we can use the dual simplex method to efficiently solve the subproblem given a new candidate solution. An additional effect is that we accumulate generated constraints from previous heuristic runs, further reducing the time for new integer solutions to be *repaired* or rejected. Secondly, we can take the lazy constraints we generate for the master problem and apply them to the heuristic subproblem to avoid re-generating them later. Finally all constraints we generate for the heuristic subproblem are valid for the master MIO, so we can add them to the master as lazy constraints to avoid generating integer solutions that will violate uncertain constraints in the first place.

The benefits of the heuristic are not guaranteed and may vary from instance-to-instance. The heuristic adds computational load that can only be made up for by improvements in speed with which we tighten the bound. In particular, by adding constraints back into the MIO master problem we may avoid needlessly generating non-robust integer solutions, but at the price of longer solve times at each node. We will explore these trade-offs between using the heuristic or not in Section 3.2.2.

2.3 Polyhedral uncertainty sets

We considered the polyhedral uncertainty set defined in [5] with two properties: for a set U_i corresponding to constraint i

- the uncertain coefficients $\tilde{a}_{ij}\ \forall j \in J_i$ lie in the interval $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$, where a_{ij} is the *nominal value*, and
- at most Γ uncertain coefficients in the row are allowed to differ from their nominal value.

One may formulate a LO problem to find the coefficients $\tilde{a}_{ij} \in U_i$ that would most violate the constraint $\tilde{a}_i^T x \leq b_i$. In [5] the authors demonstrate that duality theory allows us to replace the original uncertain constraint with a finite set of new deterministic constraints and auxiliary variables corresponding to the constraints and variables of the dual of this problem. As detailed in Table 1, this reformulation preserves the problem class of the original RO problem - a RLO problem becomes a LO problem, and likewise a RMIO problem becomes a MIO problem.

While we could generate new constraints by solving the aforementioned LO problem, we can exploit the structure to solve the problem more efficiently by applying the following algorithm to each uncertain constraint i:

- Given a current solution x to the master problem, calculate the absolute deviations $z_j = \hat{a}_{ij} |x_j|$ for all $j \in J_i$.
- Sort z_j (all non-negative by definition) from largest to smallest magnitude. The indices J' corresponding to the Γ largest values of z_j will cause the maximum violation of the constraint i if set to their upper or lower bounds.
- If the constraint can be violated by setting those coefficients to their bounds while all other coefficients remain at their nominal values, add this violated constraint.

As we are only interested in a subset of the coefficients of size Γ this algorithm will run in $O(n + \Gamma \log \Gamma)$ time per constraint using an efficient partial sorting algorithm. We also note that many other polyhedral uncertainty sets exist that will require different cutting-plane algorithms, but we consider this set to be representative of many commonly-seen polyhedral sets in the literature. Termination of the cutting-plane algorithm is guaranteed in the case of the polyhedral uncertainty set: cutting-plane generation is equivalent to optimizing a linear function over a polyhedron, and there are only finitely many extreme points of the uncertainty set.

2.4 Ellipsoidal uncertainty sets

We evaluated the ellipsoidal uncertainty sets introduced in [2,3], which model the coefficients $\tilde{a}_i \in U_i$ as belonging to the ellipse

$$\sqrt{\sum_{j \in J_i} \left(\frac{\tilde{a}_{ij} - a_{ij}}{\hat{a}_{ij}} \right)^2} \leq \Gamma.$$

The reformulation method for ellipsoidal sets replaces the uncertain constraints with new second-order cone constraints; thus LO problems become second-order cone problems (SOCPs), and MIO problems become mixed-integer SOCPs (MISOCPs) (Table 1).

The algorithm to generate a new deterministic constraint for a given uncertain constraint is equivalent to finding the maximizer of a linear function over a ball, which has a closed-form expression. If we consider the Karush-Kuhn-Tucker conditions for the cutting-plane problem $\max_{\tilde{a} \in U_i} \tilde{a}^T x$ we find that we can efficiently determine a new constraint $\bar{a}$ in $O(n)$ time by evaluating

$$\bar{a}_j^* = \frac{\Gamma}{\left\| \hat{A} x \right\|} \hat{a}_j^2 x_j + a_j$$

where $\hat{A}$ is a diagonal matrix where the diagonal elements are the elements of $\hat{a}$. Note that this closed-form solution applies to generalizations which consider correlation matrices (non-axis aligned ellipses).

An interesting point of difference between the ellipsoidal cutting-plane method and the ellipsoidal reformulation is that the cutting-plane method does not modify the problem's class as no quadratic terms appear in the generated constraints. However, unlike for polyhedral uncertainty sets, finite termination of the cutting-plane method applied to ellipsoidal uncertainty sets is not theoretically guaranteed, which perhaps discourages the implementation of this method in practice. Nevertheless, we will see that this method is indeed practical in many cases.

Table 1 Problem class of the deterministic reformulations and cutting-plane method master problems for RLO and RMIO problems, dependent on the choice of uncertainty set. The polyhedral set does not change the problem class. The cutting-plane method master problem remains in the same problem class as the original RO problem regardless of the set selected, but the most efficient algorithm to generate a new cut is dependent on the set.

	Polyhedral set		Ellipsoidal set	
Robust Problem	RLO	RMIO	RLO	RMIO
Reformulation	LP	MIO	SOCP	MISOCP
Cutting-plane master	LP	MIO	LO	MIO
Cut generation	Sorting		Closed-form	

3 Computational Benchmarking

Obtaining a large number of diverse RO problems was achieved by taking instances from standard LO and MIO problem libraries and converting them to RO problems. We obtained test problems from four sources:

1. LO problems from NETLIB [13], all of which are relatively easy for modern solvers in their nominal forms.
2. LO problems from Hans Mittelmann's benchmark library [18].
3. MIO problems from MIPLIB3 [6].
4. MIPLIB 2010, the current standard MIO benchmark library [16]. We selected only the "easy" instances, where the nominal problem is solvable to provable optimality in under an hour.

Uncertain coefficients are identified using a procedure based on the method in [3], that is similar to methods in other papers [5, 11]):

- A coefficient is *certain* if it is representable as a rational number with denominator between 1 and 100 or a small multiple of 10.
- It is otherwise declared to be *uncertain*, with a 2% deviation allowed from the nominal value: $\hat{a}_{ij} = 0.02\,|a_{ij}|$.
- All equality constraints are certain.

After removing all instances which had no uncertain coefficients, we were able to use 183 LO problems and 40 MIO problems.

We compared the reformulation and the cutting-plane methods for RLO (Section 3.1) and RMIO (Section 3.2). We tried three different values of Γ (1, 3, and 5) to test if the solution methods were sensitive to the amount of robustness. We used single-threaded computation throughout and a time limit of 1,000 seconds was enforced on all instances. All solver settings were left at their defaults unless otherwise stated.

For the case of the polyhedral reformulation for RLO problem we controlled for the choice between interior point ("barrier") and dual simplex as the solution method. In particular, for the interior point method we disabled "crossover" (obtaining an optimal basic feasible solution) as we didn't consider it relevant to the experiment at hand (the solution to the reformulation was not needed for any further use). The choice of solution method is significant in this case as we restricted the solver to one thread in all benchmarks - if

multiple threads were available, we could use both simultaneously and take the faster of the two. This implication is explored further in Section 4.

We did not experiment with different solver options for the polyhedral set RMIO reformulation but did experiment with a solver option that chooses which algorithm to use to solve the ellipsoidal reformulation (a MISOCP). There are two choices available in Gurobi 5.6 : either a linearized outer-approximation, similar to the cutting plane method we employ, or solving the nonlinear continuous relaxation directly at each node. We solved the reformulation with each algorithm to investigate their relative performance for the RO problems.

We use multiple measures to try to capture different aspects of the relative performance of the various methods and parameters. As the time to solve each problem varies dramatically we used only unitless measures, and avoided measures that are heavily affected by skew like the arithmetic mean employed in [11]. We complement these numerical results with the graphical *performance profiles* [9] which show a more complete picture of the relative runtimes for each method. Where possible we will relate the measures described below to these performance profiles.

The first and most basic measure is the *percentage of problems solved fastest* by each method. This corresponds to the y-axis values in the performance profiles for a performance ratio of 1. This measure is sensitive to small variations in runtime, especially for smaller problems, and does not capture any of the worst-case behaviors of the methods.

The second and third measures are both summary statistics of normalized runtimes. More precisely, for a particular combination of problem class (RLO, RMIO), uncertainty set (polyhedral, ellipsoidal), and Γ we define T_k^M to be the runtime of method M for instance $k \in \{1, \ldots, K\}$, and define T_k^* to be the best runtime for instance k across all methods, that is $T_k^* = \min_M T_k^M$. Then the second measure is, for each method M, to take the *median of* $\left\{ \frac{T_k^M}{T_k^*} \right\}$. The median is a natural measure of the central tendency as it avoids any issues with outliers, although it is somewhat lacking in power when more than 50% of the instances take the same value, as is sometimes the case for our data. The third measure, which doesn't have this shortcoming, is to take the *geometric mean* of the same data for each method M, that is to calculate

$$\left(\prod_{k=1}^{K} \frac{T_k^M}{T_k^*} \right)^{-K}. \tag{2}$$

We note that the arithmetic mean can give misleading results for ratios [12] and that the geometric mean is considered a suitable alternative.

A natural question is to ask how confident we are in the second and third measures, which we address with confidence intervals. To do so, we must frame our collection of test problems as samples from an infinite population of possible optimization problems. We are sampling neither randomly nor independently from this population, so these intervals must be treated with some

Table 2 Summary of RLO benchmark results for the polyhedral uncertainty set for $\Gamma = 5$. *% Best* is the fraction of problems for which a method had the lowest run time. *Median* and *Geom. Mean* are the median and geometric mean respectively of the run times for each instance normalized by the best time across methods. The percentage best for the pseudo-method *Reformulation (Best of Both)* is calculated as the sum of the two reformulation percentages. The two entries marked with an asterisk represent a separate comparison made only between those two methods in isolation from the alternatives.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	75.5	1.00 (1.00, 1.00)	1.52 (1.31, 1.89)
Reformulation (Barrier)	15.8	1.57 (1.40, 1.77)	1.88 (1.72, 2.10)
Reformulation (Dual Simplex)	8.7	1.48 (1.40, 1.54)	1.69 (1.57, 1.85)
Reformulation (Best of Both)	24.5	1.27 (1.20, 1.54)	1.39 (1.32, 1.49)
Cutting-plane *	83.7	1.00 (1.00, 1.00)	1.39 (1.22, 1.68)
Reformulation (Dual Simplex) *	16.3	1.42 (1.39, 1.54)	1.54 (1.46, 1.65)

Table 3 Summary of RLO benchmark results for the ellipsoidal uncertainty set for $\Gamma = 5$. *% Best* is the fraction of problems for which a method had the lowest run time. *Median* and *Geom. Mean* are the median and geometric mean respectively of the run times for each instance normalized by the best time across methods.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	44.8	1.06 (1.00, 1.12)	1.76 (1.51, 2.18)
Reformulation	55.2	1.00 (1.00, 1.00)	1.38 (1.27, 1.57)

caution. As the distribution of these measures is not normal, and the sample sizes are not particularly large, we estimate 95% confidence intervals using bootstrapping (e.g. [10]) which avoids some of the assumptions that would otherwise have to be made to get useful results.

Finally we found that runtimes were relatively invariant to the choice of Γ. Any differences were less than the general minor variations caused by running the benchmarks multiple times. For this reason we present results only for $\Gamma = 5$ throughout the paper. Additionally although we are not interested in comparing in any way the merits of polyhedral versus ellipsoidal sets, we wish to note that the relative "protection" level implied by a given Γ differs for the polyhedral and ellipsoidal sets.

3.1 Results for RLO

We found that the cutting-plane method was the fastest method for most (76%) RLO problems with polyhedral uncertainty sets (Table 2), which matches previous observations [11]. It also had a lower runtime geometric mean than either of the two reformulation methods tried, although the difference is not substantial when viewed together with the confidence intervals. We also performed an alternative comparison between the dual simplex reformulation method and the cutting-plane method that again shows that the cutting-plane methods is faster on a vast majority of instances, but without a definitive difference in geometric mean. An explanation for this is to be found in Figure 1 which shows that the cutting plane method is within a factor of two of the fastest run-

Table 4 Summary of RMIO benchmark results for the polyhedral uncertainty set for $\Gamma = 5$. *% Best* is the fraction of problems for which a method had the lowest run time. *Median* and *Geom. Mean* are the median and geometric mean respectively of the run times for each instance normalized by the best time across methods. The two entries marked with an asterisk represent a separate comparison made only between those two methods in isolation from the alternatives.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	32.5	1.39 (1.00, 2.39)	2.96 (2.00, 5.26)
Cutting-plane & root	20.0	1.19 (1.01, 1.41)	1.68 (1.40, 2.19)
Cutting-plane & heur.	2.5	1.37 (1.07, 2.12)	2.92 (1.96, 5.23)
Cutting-plane & root & heur.	7.5	1.22 (1.04, 1.55)	1.73 (1.43, 2.23)
Reformulation	37.5	1.28 (1.00, 1.77)	1.88 (1.50, 2.68)
Cutting-plane & root *	57.5	1.00 (1.00, 1.00)	1.58 (1.31, 2.07)
Reformulation *	42.5	1.05 (1.00, 1.62)	1.76 (1.41, 2.56)

ning time for approximately 90% of problems, but can be at least an order of magnitude slower than reformulation for the remaining 10%. Finally we have included a pseudo-method in Table 2 that takes the minimum of the two reformulation times, as many LO solvers can use both methods simultaneously. We see that while cutting-planes are still superior for a majority of problems, the geometric mean of the "best-of-both" reformulation method is lower than the cutting-plane method, although their essentially overlapping confidence intervals suggest not too much importance should be placed on this difference.

This result is inverted for RLO problems with ellipsoidal uncertainty sets, where the reformulation is superior in a small majority of cases (Table 3). Given this, it is unsurprising that there is essentially no difference in their median normalized running times. The geometric mean is more conclusive, with the reformulation method taking a clearer lead. This again seems to be due to significantly worse performance in a small number of instances, as evidenced by the performance profile in Figure 1.

3.2 Results for RMIO

For RMIO problems we have five major variations available:

1. Reformulation (in two minor variations for ellipsoidal sets).
2. Cutting-plane, without cutting-planes added at root node, heuristic disabled.
3. Cutting-plane, **with** cutting-plane method applied to root node, heuristic disabled.
4. Cutting-plane, without cutting-planes added at root node, heuristic **enabled**.
5. Cutting-plane, **with** cutting-plane method applied to root node, heuristic **enabled**.

We now investigate in more details the relative benefits of generating constraints at the root node (Section 3.2.1) and of the heuristic (Section 3.2.2).

Table 5 Summary of RMIO benchmark results for the ellipsoidal uncertainty set for $\Gamma = 5$. *% Best* is the fraction of problems for which a method had the lowest run time. *Median* and *Geom. Mean* are the median and geometric mean respectively of the run times for each instance normalized by the best time across methods. The two entries marked with an asterisk represent a separate comparison made only between those two methods in isolation from the alternatives.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	30.0	1.08 (1.00, 1.42)	2.51 (1.66, 4.53)
Cutting-plane & root	22.5	1.20 (1.00, 1.55)	1.83 (1.44, 2.80)
Cutting-plane & heur.	20.0	1.02 (1.00, 1.39)	2.52 (1.67, 4.64)
Cutting-plane & root & heur.	2.5	1.33 (1.03, 1.59)	1.88 (1.47, 2.81)
Reformulation	12.5	1.83 (1.01, 8.59)	5.92 (3.31,12.54)
Reformulation (outer-approx)	12.5	1.27 (1.01, 1.41)	2.03 (1.52, 3.20)
Cutting-plane & root *	72.5	1.00 (1.00, 1.00)	1.31 (1.12, 1.89)
Reformulation (outer-approx) *	27.5	1.07 (1.00, 1.25)	1.45 (1.24, 1.96)

The results are summarized in Tables 4 and 5 for the polyhedral and ellipsoidal uncertainty sets respectively. For the polyhedral set we found that the various cutting-plane methods were fastest in 62.5% of instances, although there was little to separate the cutting-plane methods and reformulation in the medians. There was no meaningful difference in the geometric means with the notable exception of the two cutting-plane variants that did not add cuts at the root. We proceeded to simplify the comparison to just the cutting-plane method with cuts at the root and the reformulation. When viewed in this light there seems to be a slight edge to cutting-planes, but the difference is too small to confidently declare as significant. This contradicts the results in [11] which concluded that cutting-planes were "significantly worse" than reformulation; this could be attributed to the enlarged test set, the addition of cutting planes at the root, the measurement metric used, or a combination of these factors.

For ellipsoidal uncertainty sets the cutting-plane method was better than reformulation in even more instances (75%) than for the polyhedral sets. Interestingly, and in contrast to the polyhedral set results, the cutting-plane variations without cuts at the root node performed better in the median than any other variation, although the geometric mean results reverse this result. Additionally the outer-approximation variant of the reformulation appears to perform far better overall than the alternative, perhaps due to the structure of reformulated RMIOs. Finally we isolated the best cutting-plane and reformulation variants and compared them, indicating that cutting-planes had an edge but that again the difference in geometric mean was too small relative to the uncertainty to be declared significant.

3.2.1 Treatment of root node

In Section 2.2 we considered the option of applying the RLO cutting-plane method to the root relaxation of the RMIO master problem. While this would possibly generate unnecessary constraints, we hypothesized that it may guide the search process to avoid considering integer solutions that are not feasible with respect to the uncertain constraints. We experimented to determine

Table 6 RMIO benchmark results for $\Gamma = 5$ comparing performance when cuts are added at the root node or not, with methods paired based on whether the heuristic was also used and which uncertainty set they are for. All metrics calculated as described in Table 4.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Poly., no root cuts	50.0	1.00 (1.00, 1.16)	1.96 (1.43, 3.49)
Poly., root cuts	50.0	1.00 (1.00, 1.01)	1.12 (1.05, 1.29)
Poly., heur. & no root cuts	52.5	1.00 (1.00, 1.00)	1.86 (1.35, 3.24)
Poly., heur. & root cuts	47.5	1.00 (1.00, 1.00)	1.10 (1.05, 1.22)
Ell., no root cuts	60.0	1.00 (1.00, 1.00)	1.71 (1.26, 3.03)
Ell., root cuts	40.0	1.00 (1.00, 1.01)	1.25 (1.13, 1.54)
Ell., heur. & no root cuts	65.0	1.00 (1.00, 1.00)	1.67 (1.23, 2.94)
Ell., heur. & root cuts	35.0	1.00 (1.00, 1.09)	1.24 (1.13, 1.49)

Table 7 RMIO benchmark results for $\Gamma = 5$ comparing performance when cuts are added at the root node or not, with methods paired based on which uncertainty set they are for and whether the instances are feasible or not. All metrics calculated as described in Table 4.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Poly., feas., no root cuts	56.5	1.00 (1.00, 1.00)	1.27 (1.13, 1.77)
Poly., feas., root cuts	43.5	1.00 (1.00, 1.01)	1.10 (1.04, 1.32)
Poly., infeas., no root cuts	18.2	22.99 (1.00,29.92)	12.43 (1.87,27.05)
Poly., infeas., root cuts	81.8	1.00 (1.00, 1.00)	1.22 (1.00, 1.49)
Ell., feas., no root cuts	67.2	1.00 (1.00, 1.00)	1.08 (1.03, 1.20)
Ell., feas., root cuts	32.8	1.00 (1.00, 1.02)	1.25 (1.12, 1.61)
Ell., infeas., no root cuts	30.8	20.50 (1.00,29.68)	8.54 (1.67,24.84)
Ell., infeas., root cuts	69.2	1.00 (1.00, 1.00)	1.24 (1.00, 1.78)

whether this option helps or not by solving with and without it, and the summary of the results is presented in in Table 6, where all results are relative to the better of with and without root cuts. We note that there is essentially no difference in median performance, and the difference only appears in the geometric means. In particular we note that not adding cuts at the root can result in much longer solve times, dragging the geometric means higher. This behaviour is fairly consistent across uncertainty set type and whether the heuristic is used or not.

Further investigation reveals that the difference in solve times is mostly due to its superior performance on infeasible instances, where the difference between the two is drastic in both median and geometric mean. Something similar is suggested in [11], although not explored fully. This can be seen in the performance profiles in Figure 2 where the no-root-cut lines fall underneath the other variations on the right side of the plots. For feasible instances there is not a substantial difference between the two. We conclude from this evidence that adding constraints to make the root fractional solution feasible for all uncertain constraints is generally advisable, and that if there is a reasonable expectation that infeasiblity is possible this option should definitely be used.

Table 8 RMIO benchmark results for $\Gamma = 5$ comparing performance when the heuristic is used to not using it, with methods paired based on which uncertainty set they are for. All metrics calculated as described in Table 4.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Poly., root cuts	67.5	1.00 (1.00, 1.00)	1.01 (1.00, 1.06)
Poly., heur. & root cuts	32.5	1.00 (1.00, 1.01)	1.04 (1.02, 1.07)
Ell., root cuts	67.5	1.00 (1.00, 1.00)	1.02 (1.00, 1.07)
Ell., heur. & root cuts	32.5	1.00 (1.00, 1.01)	1.05 (1.02, 1.09)

3.2.2 Repair heuristic

Section 2.2.1 details a heuristic that takes integer solutions produced during branch-and-bound that violate one or more uncertain constraints and attempts to make them feasible by modifying only the continuous variables. We evaluated the solution times with the heuristic enabled relative to the solution times with it disabled, and the summary of results is presented in Table 8. Using the heuristic was better than not using it in approximately one-third of the cases for both polyhedral and ellipsoidal sets. The geometric means were roughly one, indicating that on average the heuristic doesn't make much of a difference for the problems we considered.

3.3 Implementation Considerations

The implementation details for the individual methods could make a substantial difference. In particular, MISOCP functionality in commercial solvers is relatively new compared to LO and MIO as of the time of writing, which suggests that an efficient, problem-specific cutting plane method like the type demonstrated in this paper may have an edge over a generic solver. Over time, one might expect the performance of MISOCP reformulations to improve with the maturity of commercial implementations. This may explain why the cutting-plane method was better for a majority of the RMIO ellipsoidal uncertainty set instances we evaluated. On the other hand, MIO solvers have presolve functionality that is capable of powerful problem reductions, which would be to the benefit of reformulations. By implementing our cutting-plane generation code in C++ and connecting to the solver using *in memory* libraries, we avoided as much overhead as possible; cutting-plane overhead may grow with a less tightly coupled connection or if a slower language is used to generate cuts. We also used single-threaded computation throughout - cutting-plane generation may interact with a multi-threaded solver in unexpected ways that may slow down the solver.

4 Evaluation of Hybrid Method

In the previous section we mainly focused on the direct comparison of different variations of the cutting-plane and reformulation methods for solving RO prob-

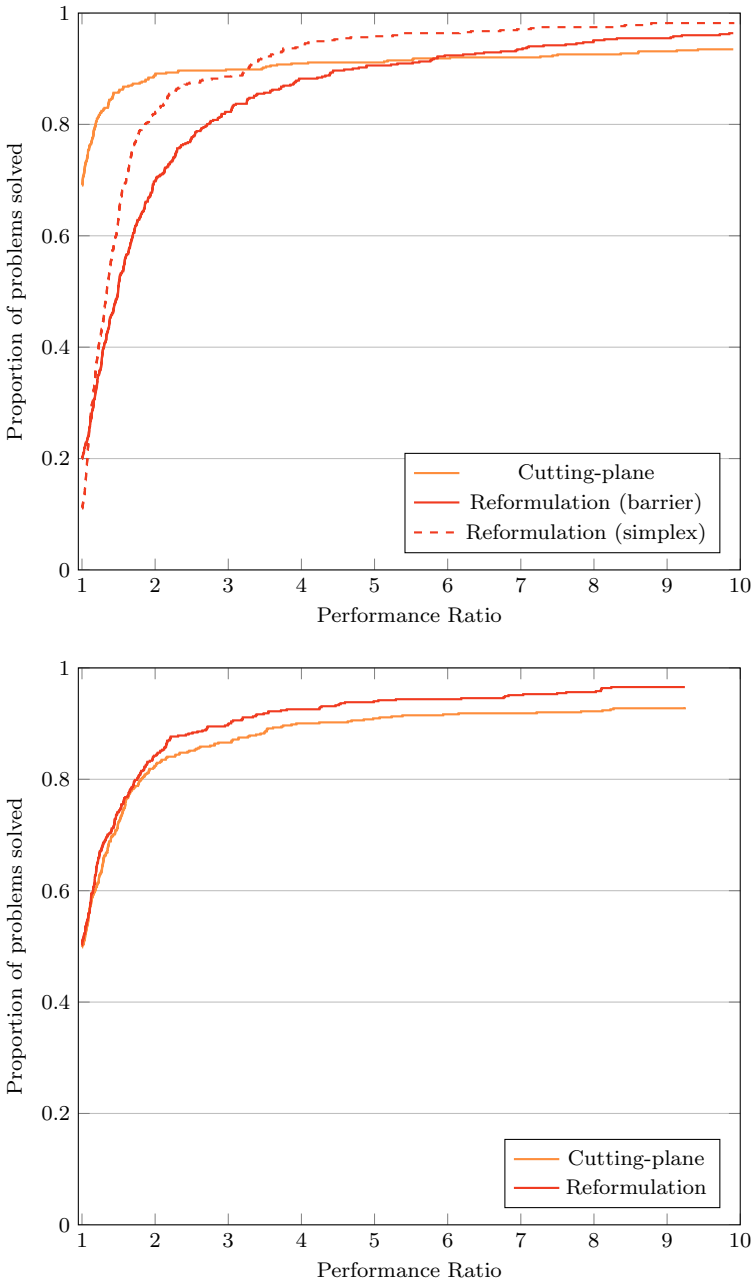

Fig. 1 Performance profiles [9] for RLO instances with polyhedral and ellipsoidal uncertainty, above and below, respectively. Higher lines indicate superior performance.
.

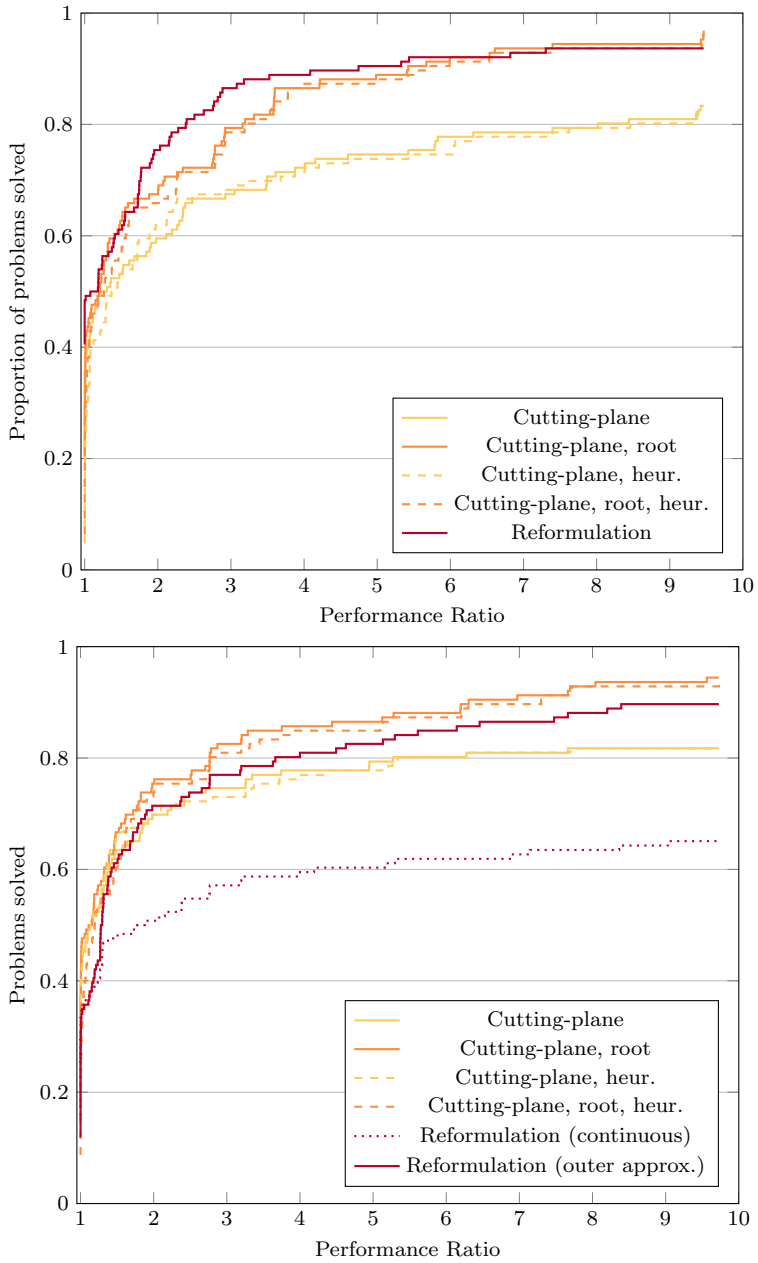

Fig. 2 Performance profiles [9] for RMIO instances with polyhedral and ellipsoidal uncertainty, above and below, respectively. Higher lines indicate superior performance. "root" indicates adding constraints at the B&B root note, and "heur" indicates that the proposed feasibility heuristic is enabled.

Table 9 Summary of hybrid method RLO benchmark results for the polyhedral uncertainty set with $\Gamma = 5$. Both measures are functions of the hybrid method runtime relative to another method's runtime (Equation 3).

	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	1.00 (1.00, 1.00)	0.66 (0.53, 0.76)
Reformulation (Barrier)	0.64 (0.56, 0.71)	0.53 (0.48, 0.58)
Reformulation (Dual Simplex)	0.68 (0.65, 0.71)	0.59 (0.54, 0.64)
Reformulation (Best of Both)	0.79 (0.73, 0.83)	0.72 (0.67, 0.76)

lems. In this section, we change the focus to how best to use this information in practice. We propose applying both the reformulation and the cutting-plane method *simultaneously* to the same problem, in much the same way that some modern solvers apply both the simplex and interior point methods to solving a (deterministic) linear optimization problem.

We define the *hybrid* method each problem class and uncertainty set type as follows:

- RLO problems with polyhedral uncertainty sets: simultaneously solve with the cutting-plane method and both reformulation methods (simplex and interior point).
- RLO problems with ellipsoidal uncertainty sets: simultaneously solve with the cutting-plane method and the reformulation method.
- RMIO problems with polyhedral uncertainty sets: simultaneously solve with the cutting-plane method (with root cuts) and with the reformulation method.
- RMIO problems with ellipsoidal uncertainty sets: simultaneously solve with the cutting-plane method (with root cuts) and with the reformulation method (outer-approximation).

.

The statistic we are interested in is a slight variant of the one considered in Section 3. For a particular combination of problem class (RLO, RMIO), uncertainty set (polyhedral, ellipsoidal), and Γ define T_k^H to be run time for the hybrid method. We are interested in the reduction in running time for the hybrid method versus any single method M, that is

$$\frac{T_k^H}{T_k^M}. \tag{3}$$

This quantity is bounded by 0 and 1 for RLO problems by construction, but may be greater than 1 for RMIO problems (for example, if we compare against the heuristic method and for a particular instance using the heuristic would have been better). We will consider the median, geometric mean, and general distribution of this quantity.

Table 10 Summary of hybrid method RLO benchmark results for the ellipsoidal uncertainty set with $\Gamma = 5$. Both measures are functions of the hybrid method runtime relative to another method's runtime (Equation 3).

	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	1.00 (0.88, 1.00)	0.57 (0.46, 0.66)
Reformulation	0.83 (0.78, 0.97)	0.68 (0.60, 0.73)

Table 11 Summary of hybrid method RMIO benchmark results for the polyhedral uncertainty set with $\Gamma = 5$. Both measures are functions of the hybrid method runtime relative to another method's runtime (Equation 3).

	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	0.72 (0.33, 0.99)	0.36 (0.20, 0.55)
Cutting-plane & root	1.00 (0.35, 1.00)	0.63 (0.49, 0.77)
Cutting-plane & heur.	0.79 (0.51, 0.98)	0.37 (0.20, 0.55)
Cutting-plane & root & heur.	0.95 (0.66, 0.99)	0.62 (0.47, 0.75)
Reformulation	0.95 (0.59, 1.00)	0.57 (0.39, 0.70)

4.1 Results for RLO

For the polyhedral uncertainty set (Table 9) we defined the hybrid method to be the best of all three methods. As the cutting-plane method was best in 75% of instances, there is no median improvement versus the cutting-plane method, and roughly 15% median improvement versus the better of the reformulations. However, if we look at the geometric means then the hybrid method is significantly better, with runtimes about 50% to 75% of any single method.

For the ellipsoidal uncertainty set (Table 10) the hybrid method is simply the better of the two available methods. It is nearly twice as fast on average as using the cutting-plane method and has runtimes less than about 70% on average of the reformulation method. Its interesting to note that the performance difference for the median is not as dramatic suggesting that for most problems there is not much difference, but for some instances the difference is substantial.

4.2 Results for RMIO

The hybrid method for RMIO problems with the polyhedral uncertainty set (Table 11) is defined to be the best of the cutting-plane method with root cuts and the reformulation method. As with RLO problems we see only moderate improvements in median, but the improvement in geometric mean shows improvements in runtime by a factor of approximately two to three.

For ellipsoidal uncertainty sets (Table 12) the hybrid method the best of the cutting-plane method with root cuts and the reformulation (outer-approximation) method. Once more we see only moderate improvements in median, but the improvement in geometric mean shows improvements in runtime up to a factor of approximately two (and even more for the alternative reformulation solution method).

Table 12 Summary of hybrid method RMIO benchmark results for the ellipsoidal uncertainty set with $\Gamma = 5$. Both measures are functions of the hybrid method runtime relative to another method's runtime (Equation 3).

	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	0.99 (0.85, 1.00)	0.55 (0.29, 0.86)
Cutting-plane & root	1.00 (0.78, 1.00)	0.76 (0.52, 0.89)
Cutting-plane & heur.	1.00 (0.77, 1.00)	0.55 (0.29, 0.85)
Cutting-plane & root & heur.	0.99 (0.93, 1.00)	0.74 (0.52, 0.87)
Reformulation	0.83 (0.14, 1.00)	0.24 (0.10, 0.47)
Reformulation (outer-approx)	0.93 (0.78, 1.00)	0.69 (0.51, 0.81)

Histograms of the hybrid runtime versus method runtimes for both uncertainty sets are displayed in Figure 3. We can see that, as expected, the histograms for the methods selected for inclusion in the hybrid method are mostly bunched around 1.0, but that for the other methods there is another peak between 0.0 and 0.2 for the instances that were solved substantially quicker with the hybrid method. It is these instances that drive the difference in geometric mean and show where a hybrid method could offer the most value.

4.3 Implementation Considerations

We note that implementation details may restrict the realizable benefits of the hybrid method. In particular, we considered only single-threaded computation for each individual method, and assumed there was no performance impact from running multiple methods simultaneously. To examine these assumptions and their impact on practical implementations we will assume that we are running on machine that can run $T \geq 2$ threads independently with minimal impact on each other. For example, a "quad core" machine could run $T = 4$ threads with minimal impact. We will analyze the impact of these realities on the two problem classes independently.

For the RLO problem we used three methods, so would require $T \geq 3$ to obtain the results stated above. An examination of the performance profiles suggests that if T is only two then we should use the cutting-plane method and the simplex method for reformulation. If $T \geq 4$ the question of what to do with the extra threads arises. While the reformulation method with the simplex algorithm is single-threaded, the interior-point reformulation method can be run in multi-threaded mode. The cutting-plane method relies on the simplex method to solve the master problem, but the cutting-planes could be generated in parallel. Thus on a machine with $T = 8$, for example, one thread could be allocated to solving the reformulation with the simplex method, two threads could be allocated to the cutting-plane method, and the remainder to the interior-point method for the reformulation.

Both the hybrid methods for RMIO problems use only two methods, requiring only that $T \geq 2$. However, both cutting-plane and reformulation methods for RMIO can use many threads, begging the question of how to distribute T threads among them. This question can only practically be answered with

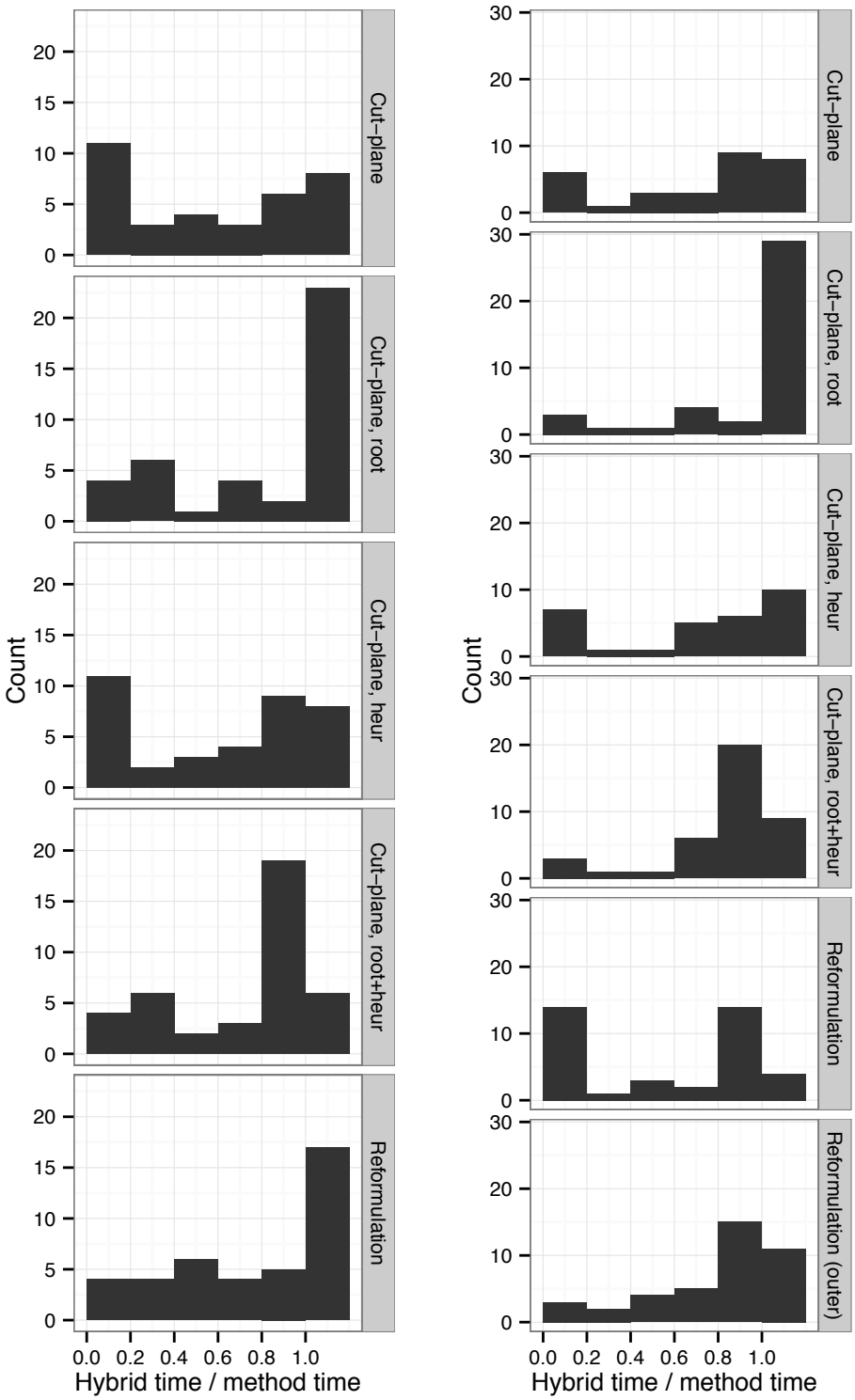

Fig. 3 Histograms of the runtime for the hybrid method normalized by each of the individual methods (Equation 3). The left column is for RMIO problems with polyhedral uncertainty sets, and the right column is for RMIO problems with ellipsoidal uncertainty sets.

further instance-specific experimentation beyond the scope of this paper. In general, however, it is far from trivial to achieve a perfect speed-up factor of T when using T threads to solve a single MIO problem [17], which suggests that a hybrid approach could provide an improvement over a pure strategy of providing all available threads to a single solver.

A parallel hybrid strategy could be further enhanced by communicating between the two solvers. Consider first the reformulation at the presolve stage: any inferences made about variable bounds, including fixing variables, determined from the reformulation are also valid for the cutting-plane method. Thus the cutting-plane method can benefit from the structure of the reformulation, which could provide improvements above those predicted above. Furthermore a feasible solution found with the cutting-plane method is also feasible for the reformulation method, allowing us to tighten the integrality gap from one direction. Finally the lower bound from the reformulation method is also a valid lower bound for the cutting-plane algorithm, allowing us to tighten the integrality gap from the other direction. Carvajal et al. [8] reported success with a similar approach in the context of general MIO problems.

5 Discussion

The relative benefits of cutting planes versus reformulations are a mixed story when compared directly against each other. The two main metrics considered were the median and geometric mean of the runtime for each method normalized by the best runtime across methods. In general, the performance gap between them was not large in either metric for any combination of problem class (RLO, RMIO) and uncertainty set, but especially in the median. The geometric mean generally revealed greater differences due to small numbers of instances that are dramatically better with one method than the other. This is reflected in the tails of the performance profiles in Figures 1 and 2. We summarize the results as follows:

- For RLO problems with polyhedral uncertainty sets, there was some evidence that the cutting-plane method was superior for a majority of cases, but there is no clear overall winner.
- For RLO problems with ellipsoidal uncertainty sets, the reformulation method was better than the cutting-plane method.
- For RMIO problems with polyhedral uncertainty sets, there was no clear winner between the cutting-plane method (with cuts added at the root) and the reformulation method.
- For RMIO problems with ellipsoidal uncertainty sets, there was some evidence that the cutting-plane method (with cuts added at the root) was better than the reformulation method (using the outer-approximation option).

The most practical benefit of benchmarking the methods was to determine the performance of a hybrid solver that runs both simultaneously. When we

consider the performance of this hybrid method we see reductions in runtime to 50% to 75% across all combinations of problem and uncertainty set. This is a clear win, and suggests that a hypothetical robust optimization solver should follow this hybrid strategy. This would be achieved by modeling the RO problem directly (perhaps with a RO-specific modeling language), then relying on the modeling tool or RO solver to automatically produce both the reformulation and the cutting-plane generator. As we noted in 4.3, further synergies over simply running both methods in isolation may be possible, at the cost of increased implementation complexity.

Future avenues for the study of computational aspects of RO would include benchmarking more and tougher MIO instances that take longer to solve. As discussed in Section 2.1, methods from nonlinear and stochastic programming may be able to reduce the number of cuts required to reach feasibility, although our preliminary experiments in this domain were not successful. Novel warm-starting methods, presolve techniques and additional heuristics may improve both methods, but especially the cutting-plane method. Finally we note that reformulations have a block structure similar to that seen in stochastic programming, which may be exploitable to speed up the solver.

Acknowlegements

The authors would like to thank the two anonymous reviewers for their help in improving the rigour and focus of the paper. This research was partially supported by ONR grant N00014-12-1-0999. M. Lubin was supported by the DOE Computational Science Graduate Fellowship, which is provided under grant number DE-FG02-97ER25308.

References

1. Ben-Tal, A., El Ghaoui, L., Nemirovski, A.: Robust optimization. Princeton University Press (2009)
2. Ben-Tal, A., Nemirovski, A.: Robust solutions of uncertain linear programs. Operations Research Letters **25**(1), 1 – 13 (1999)
3. Ben-Tal, A., Nemirovski, A.: Robust solutions of linear programming problems contaminated with uncertain data. Mathematical Programming **88**, 411–424 (2000)
4. Bertsimas, D., Brown, D.B., Caramanis, C.: Theory and applications of robust optimization. SIAM review **53**(3), 464–501 (2011)
5. Bertsimas, D., Sim, M.: The price of robustness. Operations Research **52**(1), 35–53 (2004)
6. Bixby, R., Ceria, S., McZeal, C., Savelsberg, M.: An updated mixed integer programming library: MIPLIB 3.0. Optima **58**, 12–15 (1998)
7. Briant, O., Lemarchal, C., Meurdesoif, P., Michel, S., Perrot, N., Vanderbeck, F.: Comparison of bundle and classical column generation. Mathematical Programming **113**(2), 299–344 (2008)
8. Carvajal, R., Ahmed, S., Nemhauser, G., Furman, K., Goel, V., Shao, Y.: Using diversification, communication and parallelism to solve mixed-integer linear programs. Operations Research Letters **42**(2), 186 – 189 (2014)
9. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical Programming **91**, 201–213 (2002)

10. Efron, B., Tibshirani, R.J.: An introduction to the bootstrap, vol. 57. CRC press (1994)
11. Fischetti, M., Monaci, M.: Cutting plane versus compact formulations for uncertain (integer) linear programs. Mathematical Programming Computation **4**, 239–273 (2012)
12. Fleming, P.J., Wallace, J.J.: How not to lie with statistics: the correct way to summarize benchmark results. Communications of the ACM **29**(3), 218–221 (1986)
13. Gay, D.M.: Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Newsletter **13**, 10–12 (1985)
14. Goh, J., Sim, M.: Robust optimization made easy with rome. Operations Research **59**(4), 973–985 (2011)
15. Gurobi Optimization Inc.: Gurobi optimizer reference manual (2014). URL http://www.gurobi.com
16. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Mathematical Programming Computation **3**(2), 103–163 (2011)
17. Koch, T., Ralphs, T., Shinano, Y.: Could we use a million cores to solve an integer program? Mathematical Methods of Operations Research **76**(1), 67–93 (2012)
18. Mittelmann, H.: Benchmark of parallel LP solvers. URL http://plato.asu.edu/ftp/lpcom.html
19. Mutapcic, A., Boyd, S.: Cutting-set methods for robust convex optimization with pessimizing oracles. Optimization Methods & Software **24**(3), 381–406 (2009)
20. Zverovich, V., Fbin, C., Ellison, E., Mitra, G.: A computational study of a solver system for processing two-stage stochastic LPs with enhanced benders decomposition. Mathematical Programming Computation **4**(3), 211–238 (2012)