An oracle-based framework for robust combinatorial optimization

Enrico Bettiol[1] · Christoph Buchheim[1] · Marianna De Santis[2] · Francesco Rinaldi[3]

Abstract

We propose a general solution approach for min-max-robust counterparts of combinatorial optimization problems with uncertain linear objectives. We focus on the discrete scenario case, but our approach can be extended to other types of uncertainty sets such as polytopes or ellipsoids. Concerning the underlying certain problem, the algorithm is entirely oracle-based, i.e., our approach only requires a (primal) algorithm for solving the certain problem. It is thus particularly useful in case the certain problem is well-studied but its combinatorial structure cannot be directly exploited in a tailored robust optimization approach, or in situations where the underlying problem is only defined implicitly by a given software. The idea of our algorithm is to solve the convex relaxation of the robust problem by a simplicial decomposition approach, the main challenge being the non-differentiability of the objective function in the case of discrete or polytopal uncertainty. The resulting dual bounds are then used within a tailored branch-and-bound framework for solving the robust problem to optimality. By a computational evaluation, we show that our method outperforms straightforward linearization approaches on the robust minimum spanning tree problem. Moreover, using the Concorde solver for the certain oracle, our approach computes much better dual bounds for the robust traveling salesman problem in the same amount of time.

✉ Marianna De Santis
marianna.desantis@uniroma1.it

Enrico Bettiol
enrico.bettiol@math.tu-dortmund.de

Christoph Buchheim
christoph.buchheim@math.tu-dortmund.de

Francesco Rinaldi
rinaldi@math.unipd.it

[1] Department of Mathematics, TU Dortmund, Vogelpothsweg 87, 44227 Dortmund, Germany

[2] Department of Computer, Control and Management Engineering, Sapienza University of Rome, Via Ariosto 25, 00185 Roma, Italy

[3] Department of Mathematics, University of Padova, Via Trieste 63, 35121 Padova, Italy

1 Introduction

Robust optimization has become a wide and active research area in the last decades. The aim
is to address optimization problems with uncertain data. Unlike the stochastic optimization
problem, which usually aims at optimizing expected values, the robust optimization paradigm
tries to optimize the worst case. While stochastic optimization requires full knowledge of
the probability distributions of all uncertain problem data, robust optimization only asks for
so-called uncertainty sets containing all scenarios that need to be taken into account. While
generally leading to computationally easier problems than stochastic optimization, it is well-
known that robust counterparts of tractable combinatorial optimization problems usually turn
out to be NP-hard for most types of uncertainty sets; see, e.g., [17] or the recent survey [7]
and the references therein.

In this paper, we address robust counterparts of general combinatorial optimization prob-
lems of the type

$$\min\ c^\top x + c_0$$
$$\text{s.t.}\ \ x \in X, \tag{P}$$

where $X \subseteq \{0, 1\}^n$ is any set of binary vectors describing the feasible points of the problem
at hand. The objective function coefficients $(c_0, c) \in \mathbb{R}^{n+1}$ are considered uncertain. The
robust counterpart of (P) is then given by

$$\min\ \max_{(c_0,c)\in U}\ c^\top x + c_0$$
$$\text{s.t.}\ \ x \in X, \tag{R}$$

where $U \subseteq \mathbb{R}^{n+1}$ is the so-called *uncertainty set*, collecting all likely scenarios. Note that
allowing an uncertain constant c_0 makes the approach slightly more general, even though
the latter is not relevant in the deterministic problem (P). With respect to the considered type
of uncertainty set, our approach is rather general, but we will concentrate our exposition
on the so-called *discrete uncertainty* case, where U is given as a finite set. Other classes of
uncertainty sets often considered in the literature include polytopal or ellipsoidal sets.

While many approaches devised in the literature consider special classes of combinatorial
structures X, our aim is to devise an entirely oracle-based approach. We thus assume that,
given the objective coefficients (c_0, c), an algorithm for solving Problem (P) is available,
but we do not pose any restrictions on how this algorithm works. Our approach is thus
particularly well-suited in situations where the certain problem is well-studied but does
not have a nice enough combinatorial structure that could be exploited in a tailored robust
optimization approach. This is generally the case for NP-hard underlying problems, such as,
e.g., the traveling salesman problem. Another interesting application scenario arises when the
underlying problem is not a classical textbook optimization problem, but it is given by some
sophisticated solution software. This is the case for many real-world optimization problems
solved by practitioners, which generally apply some given optimization tools without having
an insight into the functionality of the respective algorithms. Our approach does not require
any knowledge about the underlying problem, nor about the algorithm used for solving it.

As mentioned above, robust counterparts are often NP-hard even in cases where the
underlying problem (P) is tractable. Consequently, in order to solve (R), it cannot suffice to
call the oracle for solving the certain problem a polynomial number of times. This is even true
without assuming P $\neq$ NP [5]. Instead, we propose a branch-and-bound approach, where the
main ingredient is the computation of the lower bound given by the straightforward convex

relaxation of (R), namely

$$\min \ \max_{(c_0,c)\in U} \ c^\top x + c_0$$
$$\text{s.t.} \ \ x \in \text{conv}(X). \tag{C}$$

This problem is well-defined and convex, as long as U is any compact set, which we assume throughout this paper. While ellipsoidal uncertainty leads to a smooth objective in (C), which can be exploited algorithmically [8, 9], the discrete and the polytopal uncertainty cases lead to piecewise linear objective functions, requiring different solution methods.

In our approach, Problem (C) is solved by an *inner approximation algorithm*; see, e.g., [2] and the references therein. It belongs to the class of *Simplicial Decomposition* (SD) methods. First introduced by Holloway in [14] and then further studied in [13, 21, 24, 25], SD methods currently represent a standard tool in convex optimization. Our SD method makes use of two different oracles: the first one is an algorithm for solving the convex relaxation over an inner approximation of conv(X), being the convex hull of a subset X' of X. It is important to notice that such a subroutine implicitly defines the uncertainty set U, in the sense that the set U is not explicitly part of the problem input, while the rest of our algorithm is independent of U. The second oracle is the one described above, which implicitly defines the set X and hence also conv(X). Our approach can thus be seen as an oracle-based version of a generalized SD algorithm; see, e.g., [2, 3] for further details about generalized SD. The proposed method indeed performs a two-step optimization process by handling an ever expanding inner approximation of the relaxed feasible set conv(X). At a given iteration, the method first builds up a reduced problem (whose feasible set is given by the inner approximation) and solves it by means of the first oracle. It then feeds the second oracle with the information coming from the first step to hopefully generate new extreme points that guarantee a refinement of the inner approximation. If a new point cannot be found, then the solution obtained with the last reduced problem is the optimal one. The way the refinement step is carried out is crucial to guarantee finite convergence of our method in the end.

Dropping rules (i.e., rules that allow to get rid of useless points in the inner approximation) are often used in simplicial decomposition like algorithms to keep the computational cost deriving from the first oracle small enough; see, e.g., [2, 4, 25]. As pointed out in [3], defining suitable dropping rules for a generalized simplicial decomposition, while guaranteeing finite convergence of the method, is a challenging task. We propose a simple dropping rule and analyze it in depth both from a theoretical and a computational point of view.

Some other oracle-based algorithms for robust combinatorial optimization with objective function uncertainty have been devised in the literature. In particular, tailored column generation approaches for dealing with the continuous relaxation of the given combinatorial problem are studied in [6, 16]. When considering Problem (C), those column generation algorithms turn out to be closely related to a Kelley's cutting plane approach for the problem

$$\max_{(c_0,c)\in U} \ \min_{x\in\text{conv}(X)} \ c^\top x + c_0 \ ,$$

which is equivalent to (C) in case of convex U by the minimax theorem. Another interesting approach to handle the relaxation (C) is described in [19], where the author proposes a projected subgradient method that approximately solves the projection problem at each iteration by the classical Frank-Wolfe algorithm. This approach is somehow related to gradient-sliding methods, see, e.g., [20] and the references therein, and hence obviously differs from the one described in this paper.

When aiming at general approaches that do not exploit specific characteristics of the underlying problem (P), the main alternative to oracle-based algorithms are approaches based on an IP-formulation of (P). For discrete uncertainty, the non-linear objective in (C) can easily be linearized, and this approach can be extended to infinite uncertainty sets U using a dynamic generation of worst-case scenarios, provided that a linear optimization oracle over U is given; see [22] for a general analysis and [12] for an experimental comparison with reformulation-based approaches. The scenario generation method is still applicable when having only a separation algorithm for conv(X) at hand. In the experimental evaluation presented in this paper, we compare our SD approach to such a separation oracle based approach for the discrete uncertainty case, using CPLEX to solve the resulting integer linear problems.

In the subsequent section, we describe our SD approach in more detail, concentrating on the discrete uncertainty case and with a particular focus on dropping rules. In Sect. 3, we explain how we embedded this approach into a branch-and-bound framework. An experimental evaluation is presented in Sect. 4. Section 5 concludes.

2 Computation of lower bounds

The main ingredient in our approach is the computation of the lower bound given by the convex relaxation of the robust counterpart (R). Setting $P := \text{conv}(X)$ and $f(x) := \max_{(c_0,c) \in U} c^\top x + c_0$, the problem we address is thus given as

$$\begin{aligned} \min \ & f(x) \\ \text{s.t. } & x \in P. \end{aligned} \tag{CR}$$

It is easy to see that the objective function f in (CR) is convex for any uncertainty set U, however, it is not necessarily differentiable. E.g., in case of a finite set U, differentiability is guaranteed only in points $\bar{x}$ where the scenario $(c_0, c) \in U$ maximizing $c^\top \bar{x} + c_0$ is unique. In the following, we first describe the general idea of the simplicial decomposition approach applied to the potentially non-differentiable problem (CR); see Sect. 2.1. Afterwards, we investigate a variant of the approach where vertices are dropped in case they are not needed to define the current simplex. This however requires to deal with the issue of cycling; see Sect. 2.2.

Since Problem (CR) has a convex objective function f and a convex feasible set P, optimality conditions for (CR) can be stated as follows [2, Prop. 3.1.4]:

Proposition 1 *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then, x^* minimizes f over a convex set $C \subseteq \mathbb{R}^n$ if and only if $x^* \in C$ and there exists a subgradient $g \in \partial f(x^*)$ such that*

$$g^\top(z - x^*) \geq 0 \ \ \forall z \in C.$$

By introducing the *normal cone* of C at x, defined by

$$\mathcal{N}_C(x) = \{g \in \mathbb{R}^n \mid g^\top(z - x) \leq 0, \ \forall z \in C\} \ \ x \in C,$$

we can equivalently write the optimality conditions stated in Proposition 1 as follows:

Proposition 2 *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then, x^* minimizes f over the convex set $C \subseteq \mathbb{R}^n$ if and only if $x^* \in C$ and $\partial f(x^*) \cap (-\mathcal{N}_C(x^*)) \neq \emptyset$.*

2.1 General approach

We now describe the two oracles required in our SD framework. Oracle , namely CONV-O (*CONVex hull Oracle*) essentially minimizes f over the convex hull of a finite set $V \subset \mathbb{R}^n$, looking for a point $x^* \in \text{conv}(V)$ that satisfies Proposition 2. Beyond the optimal solution x^*, we also need coefficients yielding x^* as a convex combination of points in V and a subgradient c^* of f in x^* such that $-c^*$ belongs to the normal cone of $\text{conv}(V)$ in x^*.

Oracle 1: CONV-O

Input: finite subset $V \subset \mathbb{R}^n$

Output: optimizer x^* of $\min_{x \in \text{conv}(V)} f(x)$,

$\qquad \alpha^* \in \mathbb{R}_+^V$ with $\sum_{v \in V} \alpha_v^* = 1$ such that $x^* = \sum_{v \in V} \alpha_v^* v$

$\qquad c^* \in \partial f(x^*) \cap (-\mathcal{N}_{\text{conv}(V)}(x^*))$

Oracle , namely LIN-O (*LINear optimization Oracle*), is the main oracle defining the underlying problem. It takes as input an objective vector c and returns a minimizer of $\min_{x \in X} c^\top x$, which is the same as solving Problem (P).

Oracle 2: LIN-O

Input: $c \in \mathbb{R}^n$

Output: optimizer x^* of $\min_{x \in X} c^\top x$

Using these oracles, Algorithm SD works as follows (see the pseudo-code Algorithm below): the set V^k is initialized as the singleton $V^1 = \{\hat{x}^0\}$, where $\hat{x}^0$ is an arbitrary element of X. Then, we enter a loop. At each iteration k, Oracle is first called, in order to calculate a minimizer x^k of f over $\text{conv}(V^k)$ and a subgradient

$$c^k \in \partial f(x^k) \cap (-\mathcal{N}_{\text{conv}(V^k)}(x^k)) \ .$$

Then, Oracle is called, giving as output a minimizer $\hat{x}^k$ of $(c^k)^\top x$ over $x \in X$. Note that both x^k and $\hat{x}^k$ belong to $P = \text{conv}(X)$, but not necessarily to X. Since $c^k \in -\mathcal{N}_{\text{conv}(V^k)}(x^k)$ we have that

$$(c^k)^\top x \geq (c^k)^\top x^k \quad \forall \ x \in \text{conv}(V^k).$$

This means that as long as $(c^k)^\top \hat{x}^k < (c^k)^\top x^k$ we can go further in the minimization of f over P by including the point $\hat{x}^k$ in the set V^k. Otherwise, if $c^{k^\top} \hat{x}^k \geq c^{k^\top} x^k$ we can stop our algorithm, as x^k is a minimizer of f over P and $f(x^k)$ is a lower bound for Problem (R).

In Fig. 1, we demonstrate the proceeding of Algorithm SD by an example. Starting from $\hat{x}^0$, an arbitrary element of X, Oracle is called and the subgradient c^1 as well as the optimizer $x^1 = \hat{x}^0$ are computed (picture on the left). By calling Oracle with c^1 as input, the point $\hat{x}^1$ is detected and V^2 is built as the finite set consisting of the two points $\hat{x}^0$ and $\hat{x}^1$. In the second iteration, the output of Oracle includes x^2 and c^2, while in the third and last iteration (picture on the right), Algorithm SD terminates with the optimal solution x^3, as $(c^3)^\top \hat{x}^3 = (c^3)^\top x^3$.

We claim that Algorithm SD terminates after finitely many iterations with a correct result. For showing this, first observe

Algorithm 1: SD

Input: Oracle (CONV-O) and Oracle (LIN-O)

Output: optimizer x^* of (CR)

1: compute any $\hat{x}^0 \in X$ by calling Oracle with arbitrary objective
2: set $V^1 = \{\hat{x}^0\}$
3: **for** $k = 1, 2, \ldots$ **do**
4:     compute α^k, x^k, c^k by calling Oracle for the set V^k compute $\hat{x}^k$ by calling Oracle with objective c^k
5:     **if** $(c^k)^\top \hat{x}^k \geq (c^k)^\top x^k$ **then**
6:         STOP: x^k minimizes f over P
7:     **end if**
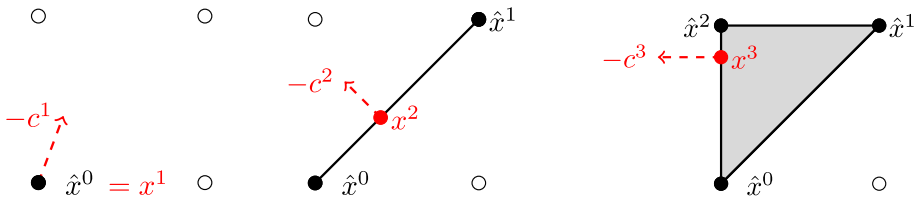8:   set $V^{k+1} := V^k \cup \{\hat{x}^k\}$
9: **end for**

Fig. 1 Illustration of Algorithm SD with $X = \{0, 1\}^2$

Lemma 1 *At every iteration k of Algorithm SD, a lower bound for Problem (CR) is given by $f(x^k) + (c^k)^\top(\hat{x}^k - x^k)$ with $x^k \in \mathrm{conv}(V^k)$ computed by Oracle and $\hat{x}^k \in X$ computed by Oracle .*

Proof Define $c_0^k := f(x^k) - (c^k)^\top x^k$, where $x^k \in \mathrm{conv}(V^k)$ is the minimizer of f over $\mathrm{conv}(V^k)$ computed by Oracle . Since $c^k \in \partial f(x^k)$, and by the choice of $\hat{x}^k \in X$, we obtain

$$f(\bar{x}) \geq f(x^k) + (c^k)^\top(\bar{x} - x^k) = c_0^k + (c^k)^\top \bar{x} \geq c_0^k + \min_{x \in P}(c^k)^\top x = c_0^k + (c^k)^\top \hat{x}^k$$

for all $\bar{x} \in P$, where the last equality holds since optimizing $(c^k)^\top x$ over X is equivalent to optimizing it over its convex hull P. Therefore, $c_0^k + (c^k)^\top \hat{x}^k = f(x^k) + (c^k)^\top(\hat{x}^k - x^k)$ is a lower bound for Problem (CR). $\square$

Theorem 3 *Algorithm SD terminates after a finite number of iterations with a correct result.*

Proof Correctness immediately follows from Lemma 1, since $f(x^k)$ is clearly an upper bound for Problem (CR) and the algorithm only terminates when $(c^k)^\top \hat{x}^k \geq (c^k)^\top x^k$. So it remains to show finiteness. Since $c^k \in -\mathcal{N}_{\mathrm{conv}(V^k)}(x^k)$ we have that

$$(c^k)^\top x \geq (c^k)^\top x^k \quad \forall\, x \in \mathrm{conv}(V^k).$$

This means that in case Algorithm SD does not terminate at iteration k, the point $\hat{x}^k \in X$ does not belong to V^k, so that V^{k+1} is a strict extension of V^k. The result then follows from the finiteness of X. $\square$

Note that this proof of convergence relies on our general assumption that X is a finite set and on the fact that we never eliminate vertices of V^k. The situation is more complicated when such an elimination is allowed, as discussed in Sect. 2.2 below.

Discrete Uncertainty. In the remainder of this subsection, we concentrate on the important special case that U consists of a finite number of scenarios $\{c_1, c_2, \ldots, c_m\} \subseteq \mathbb{R}^{n+1}$, where

we denote $c_i = (\tilde{c}_i, \bar{c}_i)$ with the uncertain constant being $\tilde{c}_i$. In this case, Oracle can be realized as follows: first note that Oracle essentially needs to solve the problem

$$\min_{x \in \text{conv}(V^k)} f(x) = \min_{x \in \text{conv}(V^k)} \max\{\bar{c}_1^\top x + \tilde{c}_1, \bar{c}_2^\top x + \tilde{c}_2, \dots, \bar{c}_m^\top x + \tilde{c}_m\}. \tag{1}$$

We denote by x^k the minimizer of (1), adopting the same notation used within Algorithm SD. As mentioned in [3], having a finite number of scenarios is one of the special cases where the calculation of a subgradient $c^k \in \partial f(x^k) \cap (-\mathcal{N}_{\text{conv}(V^k)}(x^k))$ can be obtained as a byproduct of the solution of (1). For sake of completeness, we report how the subgradient c^k is derived. Problem (1) can be rewritten as

$$\begin{aligned} \min \ & z \\ \text{s.t.} \ & \bar{c}_j^\top x + \tilde{c}_j \le z, \quad j = 1, \dots, m \\ & x \in \text{conv}(V^k) \,. \end{aligned} \tag{2}$$

From the optimality conditions of (2), we have that the optimal solution (x^k, z^k), together with the dual optimal variables λ_j^k, satisfies

$z^k = f(x^k) = \max\{\bar{c}_1^\top x^k + \tilde{c}_1, \bar{c}_2^\top x^k + \tilde{c}_2, \dots, \bar{c}_m^\top x^k + \tilde{c}_m\}$

$x^k \in \text{conv}(V^k), \quad \bar{c}_j^\top x^k + \tilde{c}_j \le z^k \quad j = 1, \dots, m$ (primal feasibility)

$(x^k, z^k) \in \underset{x \in \text{conv}(V^k),\, z \in \mathbb{R}}{\text{argmin}} \left\{ \left(1 - \sum_{j=1}^m \lambda_j^k\right) z + \sum_{j=1}^m \lambda_j^k \bar{c}_j^\top x \right\}$ (Lagrangian optimality)

$\lambda_j^k \ge 0,$ (dual feasibility)

$\lambda_j^k = 0 \quad \text{if} \quad \bar{c}_j^\top x^k + \tilde{c}_j < z^k = f(x^k) \quad j = 1, \dots, m$ (complementary slackness)

Note that

$$\begin{aligned} \min \ & \left(1 - \sum_{j=1}^m \lambda_j^k\right) z + \sum_{j=1}^m \lambda_j^k \bar{c}_j^\top x \\ \text{s.t.} \ & x \in \text{conv}(V^k) \\ & z \in \mathbb{R} \end{aligned}$$

has a solution only if $\sum_{j=1}^m \lambda_j^k = 1$, as it would be unbounded otherwise. Then, from Lagrangian optimality, we have

$$\left(\sum_{j=1}^m \lambda_j^k \bar{c}_j\right)^\top (x - x^k) \ge 0, \qquad \forall x \in \text{conv}(V^k). \tag{3}$$

It can be shown [1, p. 199] that the vector $c^k := \sum_{j=1}^m \lambda_j^k \bar{c}_j$ is a subgradient of f at x^k, and (3) implies that $-c^k$ belongs to the normal cone of $\text{conv}(V^k)$ at x^k, so that we indeed have

$$c^k \in \partial f(x^k) \cap (-\mathcal{N}_{\text{conv}(V^k)}(x^k)) \,.$$

Summarizing, when the set U is finite, an Oracle (CONV-O) suited for our purposes can be implemented by any linear programming solver able to address Problem (2), rewritten considering the α_v^k as variables. In this way, x^k is obtained as the convex combination of α_v^k.

Whenever the function f is differentiable, the choice of c^k is unique, since $\partial f(x^k) = \{\nabla f(x^k)\}$ in this case. On contrary, in case of finite U, this function is piecewise linear, so one may ask the question whether there is some freedom in the choice of c^k, which could potentially be exploited in order to find particularly promising search directions. However, it turns out that even in the discrete uncertainty case, the subgradient c^k is usually unique. As

shown in the following, when the scenarios $c_1, \ldots, c_m$ are chosen (or perturbed) randomly, with independently and continuously distributed entries, uniqueness is guaranteed with probability one. This essentially follows from the fact that the set of all uncertainty sets U leading to non-unique subgradients is not full-dimensional in $\mathbb{R}^{m(n+1)}$.

Theorem 4 *Assume that all scenarios in $U = \{c_1, \ldots, c_m\}$ are perturbed by any continuously distributed random vector in $\mathbb{R}^{m(n+1)}$ with full-dimensional support. Then, with probability one, the set $\partial f(x^k) \cap (-\mathcal{N}_{\mathrm{conv}(V^k)}(x^k))$ is a singleton in each iteration.*

Proof By definition, there exist z^k and α^k such that (z^k, x^k, α^k) is a basic optimal solution of

$$
\begin{aligned}
\min \ & z \\
\text{s.t.} \ & \bar{c}_j^\top x + \tilde{c}_j \leq z, \quad j = 1, \ldots, m \\
& x = \sum_{v \in V^k} \alpha_v v \\
& \alpha \geq 0 \\
& \sum_{v \in V^k} \alpha_v = 1 .
\end{aligned}
\tag{4}
$$

Define $A^= := \{v \in V^k \mid \alpha_v^k = 0\}$ and $C^= := \{j \in \{1, \ldots, m\} \mid \bar{c}_j^\top x^k + \tilde{c}_j = z^k\}$. The feasible set of (4) has dimension $|V^k|$, since α can be freely chosen from a simplex of dimension $|V^k| - 1$ while x depends linearly on α and z adds another dimension to the feasible set. Since (z^k, x^k, α^k) is a basic solution of (4), it follows that $|C^=| + |A^=| \geq |V^k|$. Moreover, equality holds with probability one. Indeed, due to the continuous distribution of the left hand side coefficients in the constraints $\bar{c}_j^\top x + \tilde{c}_j \leq z$, the optimal solution of (4) is degenerate with probability zero.

Now $\partial f(x^k) = \mathrm{conv}\{\bar{c}_j \mid j \in C^=\}$ has dimension at most $|C^=| - 1$ and $\mathcal{N}_{\mathrm{conv}(V^k)}(x^k)$ has dimension $n - (|V^k| - |A^=| - 1)$. Consequently, the sum of the two dimensions is at most n with probability one, but both sets are defined in $\mathbb{R}^n$ and $\partial f(x^k)$ is a convex combination of vectors with continuously distributed entries. Hence, again with probability one, the sets $\partial f(x^k)$ and $-\mathcal{N}_{\mathrm{conv}(V^k)}(x^k)$ intersect in at most one point. □

2.2 Vertex dropping rule

The running time of an iteration of Algorithm SD strongly depends on the size of V^k. The overall performance could thus benefit from a dropping rule for elements of V^k. A straightforward idea is to eliminate vertices not needed to define the minimizer of f over V^k. We thus consider the following modified update rule:

$$
V^{k+1} := \{v \in V^k \mid \alpha_v^k > 0\} \cup \{\hat{x}^k\}.
\tag{drop}
$$

In the following, we will refer to Algorithm SD where V^k is updated according to (drop) as Algorithm SD-DROP. In case of a non-differentiable function f, Algorithm SD-DROP may cycle, as shown in the following example.

Example 1 Let us consider the following problem

$$
\begin{aligned}
\min \ & \max\{x_1 - x_2, x_2 - x_1\} \\
\text{s.t.} \ & x_1 + x_2 \leq 1 \\
& x_1, x_2 \in \{0, 1\},
\end{aligned}
$$

where $\mathrm{conv}(X) = \{x \in \mathbb{R}^2 \mid x_1 + x_2 \leq 1, \ x_1, x_2 \geq 0\}$. Starting from $x^1 = \binom{0}{0}$, Algorithm SD-DROP will perform the following iterations:

Fig. 2 Illustration of Example 1
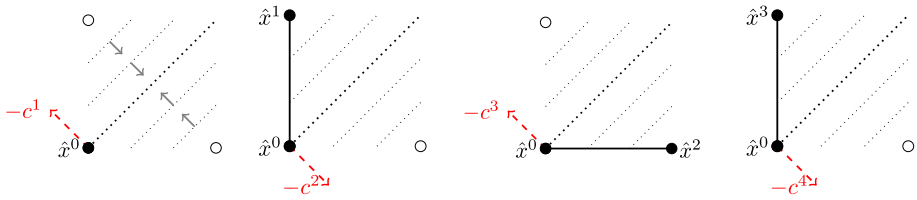
k=1: $x^1 = \binom{0}{0}, V^1 = \{x^1\}, \alpha^1 = (1)$ and $\partial f(x^1) \cap (-\mathcal{N}_{\text{conv}(V^1)}(x^1)) = \text{conv}\{\binom{1}{-1}, \binom{-1}{1}\}$.
    We choose $c^1 = \binom{1}{-1}$. Then $\hat{x}^1 = \binom{0}{1}, V^2 = \{\binom{0}{0}, \binom{0}{1}\}$
k=2: $x^2 = \binom{0}{0}, \alpha^2 = \binom{1}{0}$ and $\partial f(x^2) \cap (-\mathcal{N}_{\text{conv}(V^2)}(x^2)) = \text{conv}\{\binom{0}{0}, \binom{-1}{1}\}$.
    We choose $c^2 = \binom{-1}{1}$. Then $\hat{x}^2 = \binom{1}{0}, V^3 = \{\binom{0}{0}, \binom{1}{0}\}$
k=3: $x^3 = \binom{0}{0}, \alpha^3 = \binom{1}{0}$ and $\partial f(x^3) \cap (-\mathcal{N}_{\text{conv}(V^3)}(x^3)) = \text{conv}\{\binom{0}{0}, \binom{1}{-1}\}$.
    We choose $c^3 = \binom{1}{-1}$. Then $\hat{x}^3 = \binom{0}{1}, V^4 = \{\binom{0}{0}, \binom{0}{1}\}$.

At iteration $k = 3$, we thus get $V^4 = V^2$ and the algorithm cycles. See Fig. 2 for an illustration.

Considering this example, two questions may arise. Firstly, the solution x^1 is actually optimal, so that choosing a better subgradient (namely zero) would have stopped the algorithm immediately. Secondly, the scenarios $(0, 1, -1)^\top$ and $(0, -1, 1)^\top$ defining the uncertainty set U contain negative entries. The following example shows that neither of the two features causes cycling:

Example 2 Let us consider the following problem

$$\min \ \max\{x_1, x_2\}$$
$$\text{s.t.} \ \ x_1 + x_2 \geq 1$$
$$x_1, x_2 \in \{0, 1\},$$

where $\text{conv}(X) = \{x \in \mathbb{R}^2 \mid x_1 + x_2 \geq 1, \ x_1, x_2 \leq 1\}$. Starting from $x^1 = \binom{1}{1}$, Algorithm SD-DROP will perform the following iterations:

k=1: $x^1 = \binom{1}{1}, V^1 = \{x^1\}, \alpha^1 = (1)$ and $\partial f(x^1) \cap (-\mathcal{N}_{\text{conv}(V^1)}(x^1)) = \text{conv}\{\binom{1}{0}, \binom{0}{1}\}$.
    We choose $c^1 = \binom{1}{0}$. Then $\hat{x}^1 = \binom{0}{1}, V^2 = \{\binom{1}{1}, \binom{0}{1}\}$
k=2: x^2 may be $\binom{1}{1}$ again, with $c^2 = c^1$ and $\hat{x}^2 = \hat{x}^1$. Since $\alpha^2 = \binom{1}{0}$, we eliminate $\binom{0}{1}$,
    hence $V^3 = \{\binom{1}{1}\} = V^1$.

See Fig. 3 for an illustration.

It is easy to see that cycling cannot occur when f is differentiable. In fact, in this case, if x^k is not optimal for Problem (CR), we have $f(x^{k+1}) < f(x^k)$, since $-c^k$ is a descent direction. In particular, Algorithm SD-DROP terminates after a finite number of iterations.

For the case of finite U, differentiability is not given, and cycling can occur as seen in the examples above. However, we can still show a weaker result: we will prove that a small random perturbation of the scenario entries ensures that the objective function value $f(x^k)$ strictly decreases in every iteration and thus can avoid cycling (with probability one). For this, we first need the following observations.
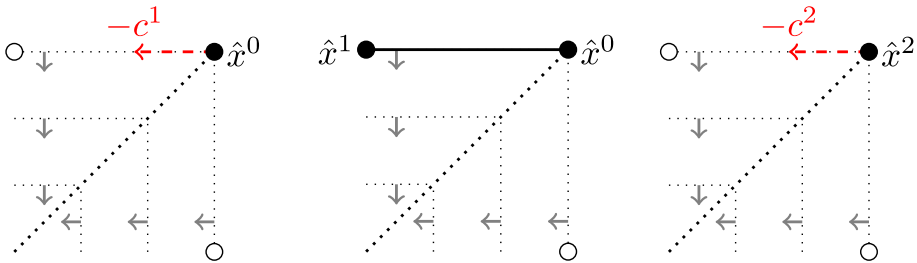
Fig. 3 Illustration of Example 2

Lemma 2 *Let L be an affine subspace of $\mathbb{R}^n$ and assume that $f|_L$, the restriction of f to L, is differentiable in $x \in L$. Consider $y \in L$ and $g \in \partial f(x)$ with $g^\top(y - x) < 0$. Then $y - x$ is a descent direction of f in x.*

Proof Let $d := y - x$. Then $\frac{\partial f}{\partial d}(x) = -\frac{\partial f}{\partial(-d)}(x)$, since $f|_L$ is differentiable in x. From $g \in \partial f(x)$ we obtain $\frac{\partial f}{\partial(-d)}(x) \geq g^\top(-d)$. Hence $\frac{\partial f}{\partial d}(x) \leq g^\top d < 0$. □

Lemma 3 *Consider an iteration k in which Algorithm SD-DROP does not terminate. Let L be an affine subspace of $\mathbb{R}^n$ containing x^k such that*

(i) $f|_L$ is differentiable in x^k,
(ii) $\dim(L \cap \mathrm{aff}(V^{k+1})) \geq 1$,
(iii) and c^k is not orthogonal to $L \cap \mathrm{aff}(V^{k+1})$.

Then $f(x^{k+1}) < f(x^k)$.

Proof By (ii), there exists some $y \in L \cap \mathrm{aff}(V^{k+1})$ with $y \neq x^k$. Since L and $\mathrm{aff}(V^{k+1})$ are affine spaces both containing x^k, we may choose y such that $(c^k)^\top(y - x^k) \leq 0$ (otherwise replace y by $2x^k - y$). By (iii), we may even assume that $(c^k)^\top(y - x^k) < 0$. Using Lemma 2 and (i), we thus derive that $y - x^k$ is a descent direction of f in x^k. It thus remains to show that $x^k + \varepsilon(y - x^k) \in \mathrm{conv}(V^{k+1})$ for some $\varepsilon > 0$, which implies that some $\bar{x} \in \mathrm{conv}(V^{k+1})$ has a strictly smaller objective value than x^k and hence $f(x^{k+1}) \leq f(\bar{x}) < f(x^k)$. Since $y \in \mathrm{aff}(V^{k+1})$, and by the definition of V^{k+1} according to (drop), we can write

$$y = x^k + \sum_{v \in \bar{V}^k} \gamma_v(v - x^k) + \delta(\hat{x}^k - x^k) , \tag{5}$$

where we define $\bar{V}^k := \{v \in V^k \mid \alpha_v^k > 0\}$. As x^k belongs to the relative interior of $\mathrm{conv}(\bar{V}^k)$, there exists $\bar{\varepsilon} > 0$ such that $x^k + \sum_{v \in \bar{V}^k} \bar{\varepsilon}\gamma_v(v - x^k) \in \mathrm{conv}(\bar{V}^k)$. Now, since both $c^k \in -\mathcal{N}_{\mathrm{conv}(V^k)}(x^k)$ and x^k is in the relative interior of $\mathrm{conv}(\bar{V}^k)$, we have $(c^k)^\top(v - x^k) = 0$ for all $v \in \bar{V}^k$. In addition,

since SD-DROP does not terminate in iteration k, we have $(c^k)^\top \hat{x}^k < (c^k)^\top x^k$. Together with $(c^k)^\top(y - x^k) < 0$,

we derive $\delta > 0$ from multiplying (5) by c^k. By choosing $\bar{\varepsilon} \leq \frac{1}{\delta}$, we may assume that $x^k + \bar{\varepsilon}\delta(\hat{x}^k - x^k) \in \mathrm{conv}(V^{k+1})$. Altogether, we derive that $x^k + \frac{1}{2}\bar{\varepsilon}(y - x^k) \in \mathrm{conv}(V^{k+1})$. □

Theorem 5 *Assume that all scenarios in $U = \{c_1, \dots, c_m\}$ are perturbed by any continuously distributed random vector in $\mathbb{R}^{m(n+1)}$ with full-dimensional support. Then, with probability one, Algorithm SD-DROP terminates after finitely many iterations.*

Proof Consider any iteration k in which Algorithm SD-DROP does not terminate. Then it suffices to show that $f(x^{k+1}) < f(x^k)$ with probability one, since X is finite. For this, let L be the maximal affine space such that $x^k \in L$ and $f|_L$ is differentiable in x^k. By the definition of f, its epigraph epi(f) is a polyhedron, and L is obtained by projecting the minimal face of epi(f) containing $(x^k, f(x^k))$ onto $\mathbb{R}^n$ and taking the affine hull of the projection. In particular, L is an affine subspace of $\mathbb{R}^n$ containing x^k which depends continuously on the perturbation of $c_1, \ldots, c_m$, whose dimension is $n - \dim(\partial f(x^k))$. We claim that the conditions (ii) and (iii) of Lemma 3 are satisfied by L with probability one, so that the result follows. Using the same notation as in the proof of Theorem 4 and setting $\bar{V}^k := \{v \in V^k \mid \alpha_v^k > 0\}$, we note that $\dim(\text{aff } \bar{V}^k) = |V^k| - |A^=| - 1$ and, as shown in the proof of Theorem 4, $|C^=| + |A^=| = |V^k|$ with probability one. Hence, we have:

$$\dim(L) = n - \dim(\partial f(x^k)) = n - (|C^=| - 1) = n - (|V^k| - |A^=| - 1) = n - \dim(\text{aff } \bar{V}^k)$$

with probability one. Thus, with probability one, we obtain $\dim(L) + \dim(\text{aff } \bar{V}^k) = n$ and hence $\dim(L) + \dim(\text{aff } V^{k+1}) = n + 1$, because $\hat{x}^k \notin \text{aff } \bar{V}^k$ and thus $\dim(\text{aff } V^{k+1}) = \dim(\text{aff } \bar{V}^k) + 1$. Thus (ii) holds with probability one. For showing (iii), we use again that $\dim(L \cap \text{aff } V^{k+1}) \geq 1$ with probability one. This implies that the probability of the fixed vector c^k being orthogonal to $L \cap \text{aff } V^{k+1}$ is zero. $\qquad\square$

Theorem 5 shows that cycling can be avoided by applying small random perturbations to the scenarios $c_1, \ldots, c_m$, e.g., by choosing $(\hat{c}_j)_i \in [(c_j)_i - \varepsilon, (c_j)_i + \varepsilon]$ uniformly at random for some $\varepsilon > 0$, independently for all $j = 1, \ldots, m$ and $i = 0, \ldots, n$. As X is finite, the optimal solution of the perturbed problem (R) will agree with an optimizer of the unperturbed problem if ε is small enough (even though this is not true for the relaxation (CR)). Note that Theorem 5 requires that also the constant in the objective function is perturbed.

Remark 1 In practice, the perturbation applied in Theorem 5 is not necessary, because small numerical errors arising in the optimization process will have the same effect. In our experiments described in Sect. 4, we do not explicitly apply any perturbation.

Note that Theorem 4 also holds when eliminating vertices. In particular, this implies that, when starting from the same set V^k, the next subgradient c^k is the same with or without elimination (with probability 1). However, in a later iteration, the set V^k and hence the optimal solution x^k may be different in the two cases, and thus also the subgradients. In our experiments, we observed that vertices being eliminated were sometimes re-generated in subsequent iterations.

Removing all vertices with zero weight might be too aggressive as a dropping strategy, as some of the vertices removed in the first iterations might be useful in the subsequent iterations. A more conservative strategy might hence be eliminating vertices $v \in V^k$ with zero weight only if they define a strict ascent direction, i.e., if $(c^k)^\top (v - x^k) \geq \varepsilon$ with a fixed threshold $\varepsilon > 0$. We thus consider the following modified update rule:

$$V^{k+1} := \{v \in V^k \mid \alpha_v^k > 0 \text{ or } (c^k)^\top (v - x^k) < \varepsilon\} \cup \{\hat{x}^k\}. \qquad \text{(drop2)}$$

Both dropping strategies (drop) and (drop2) will be carefully analyzed in Sect. 4.

3 Embedding SD into a branch-and-bound scheme

In order to solve Problem (R) to optimality, we embed Algorithm SD into a branch-and-bound scheme, which we will denote by BB-SD. Recall that Algorithm SD is able to solve

Problem (CR), the continuous relaxation of Problem (R), in a finite number of iterations, yielding an optimizer x^* and a set of feasible solutions V^* such that $x^* \in \text{conv}(V^*)$.

Within BB-SD we adopt a *depth first search (DFS)*. This choice is motivated by the fact that we need an enumeration strategy that provides primal solutions quickly, assuming that we do not have access to any problem-specific heuristics. Moreover, the branching rule implemented within BB-SD branches on variables that are fractional in the continuous relaxation, by means of the canonical disjunction. More precisely, we branch on the fractional variable x_i closest to one and produce two child nodes: in the node considered first, we fix the branching variable to 1, in the other node we fix it to 0. This choice, combined with DFS, typically allows to quickly find integer solutions, which are sparse for many combinatorial problems. Note that all nodes in BB-SD remain feasible. Indeed, since $x^* \in \text{conv}(V^*)$, regardless of how we select the fractional variable x_i to branch on, the set V^* must contain both solutions with $x_i = 0$ and $x_i = 1$

Note that some specific features of Algorithm SD can be exploited within BB-SD. Firstly, all binary vertices generated at some node of the branch-and-bound tree can be reused in the child nodes. Indeed, if we branch on fractional variables, each such vertex must be feasible in one of the child nodes, and can thus be inherited. This initial set of vertices enables us to *warmstart* the SD algorithm at every child node. Moreover, thanks to Lemma 1, every iteration of SD leads to a valid lower bound on the solution of the convex relaxation considered, meaning that *early pruning* can be performed. More precisely, at every node we either need to solve the convex relaxation to optimality or we can stop as soon as SD computes a lower bound greater than the current upper bound. In both cases, thanks to Theorem 3, the number of iterations performed by SD is finite.

As emphasized above, we assume that Problem (P) can be accessed only by an optimization oracle. Therefore, even when dealing with specific combinatorial problems, we do not exploit any structure to define primal heuristics within BB-SD. Nevertheless, at every node of BB-SD, we easily get an upper bound by evaluating the objective function on all the generated extreme points and taking the minimal value among them.

4 Numerical results

To test the performance of our algorithm SD and of the branch-and-bound scheme BB-SD, we considered instances of Problem (R) with two different underlying problems: the Spanning Tree problem (Sect. 4.1) and the Traveling Salesman problem (Sect. 4.2). The standard models for these problems use an exponential number of constraints that can be separated efficiently. In the case of the Spanning Tree problem, this exponential set of constraints yields a complete linear formulation, while this is not the case for the NP-hard Traveling Salesman problem. For the robust Minimum Spanning Tree Problem, we report a comparison between BB-SD and the MILP solver of CPLEX 12.9 [15]. For the robust Traveling Salesman Problem, we focus on the continuous relaxations, thus reporting a comparison on the bounds obtained at the root node of the branch-and-bound tree.

In the implementation of SD, for both the robust Minimum Spanning Tree Problem (r-MSTP) and the robust Traveling Salesman Problem (r-TSP), Oracle (LIN-O) is defined according to the underlying problem: for the r-MSTP we implemented the standard Kruskal algorithm [18], a well-known polynomial-time algorithm. For the r-TSP, we used the implementation of the solver Concorde [10]. Since the TSP is NP-hard, the computational times

needed to call the linear Oracle differ significantly in the two problems, as seen later in the numerical experiments.

Except for the Oracle , we used exactly the same implementation for both problems. In particular, we applied the same Oracle for both r-MSTP and r-TSP. Problem (2) is rewritten by expanding the condition $x \in \text{conv}(V^k)$. By using the LP formulation (4) and eliminating the x variables, we obtain the following equivalent formulation:

$$
\begin{aligned}
\min \ & z \\
\text{s.t.} \ & \sum_{v \in V^k} \bar{c}_j^v \alpha_v + \tilde{c}_j \leq z, \quad j = 1, \dots, m \\
& \sum_{v \in V^k} \alpha_v = 1 \\
& \alpha_v \geq 0, \quad v \in V^k,
\end{aligned}
\tag{6}
$$

where $\bar{c}_j^v = \bar{c}_j^\top v$, for every $v \in V^k$. Problem (6) is solved with the LP solver of CPLEX 12.9. Note that the number of constraints depends on the number of scenarios, and the number of variables corresponds to the cardinality of V^k and thus increases at every iteration in our approach SD. The dropping rule implemented in SD-DROP may reduce the size of this problem, thus potentially leading to practical improvements in the running time.

Our numerical experiments are organized as follows. We start by analyzing the performance of Algorithm SD. For this, in Sect. 4.1.1, we compare the use of different dropping rules on continuous relaxations of instances of the Robust MST. Then, on the same instances, we evaluate the performance of Algorithm BB-SD in Sect. 4.1.2, showing the benefits of warmstarting Algorithm SD along the nodes. Still for the Robust MST, we compare the performance of Algorithm BB-SD and CPLEX on the generated instances in Sect. 4.1.3.

In Sect. 4.2 we investigate the Robust TSP. We emphasize that our approach computes exactly the same bound as CPLEX for the Robust MST, while this is not the case for the Robust TSP: by using an exact linear optimization oracle (Oracle) we implicitly optimize over the TSP polytope while CPLEX must rely only on a relaxation. This is why for the Robust TSP we limit ourselves to the comparison between SD and CPLEX on the continuous relaxation of problem (R) at the root node. We implemented BB-SD in C++ and all the tests were run in single thread on an Intel Xeon processor CPU E5-2670 running at 2.60 GHz (16 cores) with 64 GB RAM.

4.1 Spanning Tree Problem

Given an undirected weighted graph $G = (N, E)$, a minimum spanning tree is a subset of edges that connects all vertices, without any cycles and with the minimum total edge weight. We use the following formulation of the Robust Minimum Spanning Tree problem:

$$
\begin{aligned}
\min \ & \max_{c \in U} c^\top x \\
\text{s.t.} \ & \sum_{(u,v) \in E} x_{u,v} = |N| - 1 \\
& \sum_{u,v \in X} x_{u,v} \leq |X| - 1, \ \forall \emptyset \neq X \subseteq N \\
& x \in \{0,1\}^E
\end{aligned}
\tag{r-MSTP}
$$

The binary vector x in (r-MSTP) corresponds to a set $E' \subseteq E$ of edges in G. The second class of constraints in (r-MSTP) excludes cycles in (N, E'). Together with this, the first constraint enforces that (N, E') is connected.

The objective function can easily be linearized by introducing a new variable $z \in \mathbb{R}$ and constraints $z \geq c^\top x$ for all $c \in U$. In the above model, the number of inequalities is exponential in the input size, hence we have to use a separation algorithm within CPLEX.

Table 1 Comparison between different dropping rules (1000 scenarios)

N	E	d0		d1		d2	
		time (s)	#it	time (s)	#it	time (s)	#it
20	190	0.27	69.3	0.30	105.0	0.25	73.1
30	435	0.85	111.1	0.95	184.4	0.73	110.6
40	780	1.72	152.2	2.31	295.2	1.59	155.6
50	1225	2.55	173.6	3.74	353.6	2.44	172.4
60	1770	3.36	208.5	4.65	416.6	3.28	205.5

For our experiments, we consider a benchmark of randomly generated instances of r-MSTP. We build complete graphs of five different sizes (from 20 to 60 nodes). The nominal costs are real numbers randomly chosen in the interval [1, 2]. For each size we randomly generate 10 different nominal cost vectors. The scenarios $c \in U$ are generated by adding to the vector of nominal costs a random unit vector, multiplied by a scalar factor β. We consider three such factors 1, 2, and 3, and generate three different numbers of scenarios (#sc) 10, 100, and 1000. In total, then, we have a benchmark of 450 instances, available at https://github.com/enribet/MST-Instances/.

In a first experiment, we analyze different dropping rules within algorithm SD. Then, we show how performing a warm start along the branch-and-bound iterations leads to a significant reduction in terms of number of iterations compared to a cold start. Finally, our branch-and-bound method BB-SD is compared to the MILP solver of CPLEX. Within CPLEX, we apply a dynamic separation algorithm using a callback adding lazy constraints, adopting a simple implementation based on the Ford-Fulkerson algorithm.

4.1.1 Comparison of dropping rules

In this section, we focus on the performance of algorithm SD for solving the continuous relaxation of our r-MSTP instances. In particular, we compare the performance of SD implementing three different dropping rules. Since all instances with 10 or 100 scenarios are solved in less than 0.1 seconds, we only consider the continuous relaxations of instances with 1000 scenarios, meaning that the evaluation is carried out on 150 instances. As dropping rules, we implemented the following:

- d0: meaning that no dropping rule is applied within SD;
- d1: meaning that we update the set V^k according to rule (drop) defined in Sect. 2.2, i.e., we eliminate all vertices $v \in V^k$ such that $\alpha_v^k = 0$ at every iteration of SD;
- d2: meaning that we update the set V^k according to rule (drop2) defined in Sect. 2.2, i.e., we eliminate vertices $v \in V^k$ such that $\alpha_v^k = 0$ *only if* they provide a strict ascent direction, i.e., if $(c^k)^\top (v - x^k) \geq \varepsilon$ with a fixed threshold $\varepsilon > 0$. In our numerical experiments we used $\varepsilon = 0.01 \cdot ||c^k||_2$.

As mentioned before, for each $|N|$ we built 30 instances, 10 for each factor β. In Table 1, we report the average running times in seconds (time) and the average numbers of iterations (#it) for each version of SD; note that the number n of variables in (P) is given by $|E|$ here. To further analyze the performance of the three different rules, we also report in Table 2 the median, the minimum and the maximum running time attained in seconds.

For our comparison we also use performance profiles (PP) as proposed by Dolan and Moré [11]. Given a set of solvers $\mathcal{S}$ and a set of problems $\mathcal{P}$, the performance of a solver $s \in \mathcal{S}$ on
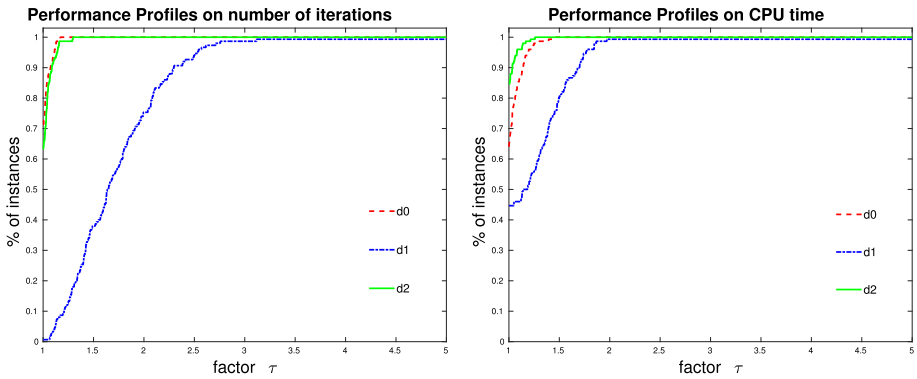
Fig. 4 Comparison between different dropping rules (1000 scenarios)

Table 2 Comparison between different dropping rules (1000 scenarios) – focus on CPU time

N	E	d0			d1			d2		
		median (s)	min (s)	max (s)	median (s)	min (s)	max (s)	median (s)	min (s)	max (s)
20	190	0.22	0.04	0.76	0.21	0.03	0.96	0.21	0.04	0.71
30	435	0.81	0.15	1.76	0.73	0.17	2.37	0.59	0.15	1.58
40	780	1.49	0.45	3.79	1.79	0.38	5.69	1.33	0.34	3.69
50	1225	2.12	0.44	6.04	3.26	0.5	10.34	2.22	0.41	5.63
60	1770	2.82	0.66	8.38	3.57	0.75	15.46	2.73	0.59	9.53

problem $p \in \mathcal{P}$ is compared against the best performance obtained by any solver in $\mathcal{S}$ on the same problem. The performance ratio is defined as $r_{p,s} = t_{p,s} / \min\{t_{p,s'} \mid s' \in \mathcal{S}\}$, where $t_{p,s}$ is the measure we want to compare. The performance profile for $s \in S$ is the plot of the cumulative distribution function $\rho_s(\tau) = |\{p \in \mathcal{P} \mid r_{p,s} \leq \tau\}|/|\mathcal{P}|$. In Fig. 4, we report the performance profiles related to the CPU time (in seconds) and the number of iterations.

We notice that dropping rule d2 allows SD to have slightly better performance in terms of CPU time, despite being not always better with respect to SD with no dropping rule in terms of number of iterations. Indeed, in the MST problem, the linear oracle (Oracle) calls are extremely fast, while most of the computational time is needed to solve the problem (6). This explains why, although some more iterations are needed, dropping some vertices and hence reducing the size of problem (6) can improve the overall performance of the algorithm. On the other hand, it is clear from the results that eliminating all inactive vertices as in Algorithm SD-DROP (rule d1) is not beneficial for SD as both the number of iterations and the CPU time increase.

4.1.2 Warmstart benefits

In the following, we evaluate our branch-and-bound method BB-SD. In particular, we will compare the performance of BB-SD considering both SD with no dropping rule (d0) and SD with dropping rule (d2). The comparison is done on all 450 instances of r-MST. As mentioned before, for each combination of $|N|$ and #sc, we built 30 instances, 10 for each value of the scalar factor β. In Table 3, we first compare the performance of BB-SD by considering SD with no dropping rule with and without warmstart (d0 no ws vs d0 with ws). In the

Table 3 Using `BB-SD` with and without warmstart

N	E	#sc	d0 no ws				d0 with ws			
			#sol	time (s)	#nodes	#it	#sol	time (s)	#nodes	#it
20	190	10	30	1.49	6.72e+2	8.00e+3	30	0.84	6.69e+2	4.23e+3
		100	30	60.03	6.85e+3	1.63e+5	30	33.44	6.87e+3	8.56e+4
		1000	24	476.59	6.74e+3	1.66e+5	27	588.38	1.24e+4	1.78e+5
30	435	10	30	13.12	3.15e+3	5.97e+4	30	7.34	3.52e+3	3.21e+4
		100	25	602.07	3.55e+4	1.15e+5	27	451.05	5.04e+4	8.29e+5
		1000	11	493.33	3.47e+3	8.30e+4	14	642.41	8.64e+3	1.12e+5
40	780	10	30	110.32	1.59e+4	4.10e+5	30	51.77	1.59e+4	1.91e+5
		10	17	764.88	3.24e+4	1.10e+6	19	632.17	5.25e+4	8.65e+5
		10	8	1592.79	7.67e+3	1.78e+5	9	837.77	9.12e+3	9.24e+4
50	1225	10	30	441.38	4.35e+4	1.32e+6	30	201.05	4.40e+4	6.10e+5
		100	10	137.71	7.26e+3	1.53e+5	13	558.02	3.47e+4	5.81e+5
		1000	7	1209.52	3.83e+3	9.81e+4	9	810.38	5.92e+3	6.64e+4
60	1770	10	25	513.71	4.21e+4	1.28e+6	30	575.38	9.81e+4	1.47e+6
		100	9	240.12	9.04e+3	2.09e+5	12	689.81	3.50e+4	5.77e+5
		1000	2	867.80	2.08e+3	5.77e+4	3	1136.77	6.47e+3	7.58e+4

table, we report the number of instances solved within the time limit of one hour (#sol), the average running times in seconds (time), the average number of nodes (#nodes) and the average number of iterations (#it) for each version of `SD`. To further analyze the performance of `BB-SD`, we also report in Table 4 the median, the minimum and the maximum running time attained in seconds. All metrics are taken over the instances solved within the time limit. It is clear from the results that the warmstart leads to a considerable decrease in the average number of iterations and consequently a significant decrease in CPU time. The same behavior can be noticed when looking at Table 5 and Table 6 where we compare `BB-SD` using dropping rule `d2` with and without warmstart (`d2 no ws` vs `d2 with ws`).

In Fig. 5, we further report the performance profiles with respect to the CPU time of the four versions of `BB-SD`. The versions of `BB-SD` with no elimination (`d0`) and with dropping rule `d2` show very similar performances. The profiles clearly show that `BB-SD` with no warmstart is almost two times slower than `BB-SD` with warmstart. In fact, the profiles are getting closer only when $\tau \geq 1.8$. This can be noticed also by looking at the metrics reported in Tables 3-6. Note that from our results it is clear that the instances become harder with a higher number of scenarios. We can also notice that `BB-SD` with `d2 with ws` shows slightly better performances when looking at the hardest instances, namely those with 1000 scenarios.

4.1.3 Comparison between BB-SD and CPLEX

We now compare our branch-and-bound algorithm `BB-SD` with `CPLEX` on our r-MSTP instances. As already mentioned, within the MILP solver of `CPLEX`, we apply a dynamic separation algorithm using a callback adding lazy constraints, adopting a simple implementation based on the Ford-Fulkerson algorithm. The comparison is made with `BB-SD` implementing the dropping rule `d2` and allowing warmstart. In Table 7, we report for each

Table 4 Using `BB-SD` with and without warmstart – focus on CPU time

N	E	#sc	d0 no ws				d0 with ws			
			#sol	median (s)	min (s)	max (s)	#sol	median (s)	min (s)	max (s)
20	190	10	30	0.505	0.01	7.84	30	0.28	0.01	4.53
		100	30	12.88	0.24	634.19	30	6.67	0.16	367.72
		1000	24	129.485	2.08	3199.37	27	100.79	1.00	3381.95
30	435	10	30	3.74	0.08	130.15	30	1.91	0.05	64.92
		100	25	116.7	1.95	3318.87	27	70.11	1.00	2411.94
		1000	11	127.16	53.81	3094.10	14	107.29	26.51	2454.02
40	780	10	30	47.25	0.24	771.56	30	19.58	0.14	358.47
		10	17	91.2	1.25	3286.76	19	124.17	0.74	3550.3
		10	8	1603.91	221.19	3497.91	9	827.92	95.49	1809.2
50	1225	10	30	107.57	0.5	1774.98	30	50.93	0.3	820.17
		100	10	61.83	18.79	495.49	13	55.32	8.17	3367.21
		1000	7	823.86	257.84	3435.23	9	425.75	86.55	1816.43
60	1770	10	25	253.83	3.68	2680.46	30	142.68	1.79	3209.57
		100	9	144.65	36.48	1056.5	12	83.62	18.72	3032.88
		1000	2	867.795	855.32	880.27	3	433.79	349.25	2627.27

Table 5 Using `BB-SD` with and without warmstart, applying dropping rule `d2`

N	E	#sc	d2 no ws				d2 with ws			
			#sol	time (s)	#nodes	#it	#sol	time (s)	#nodes	#it
20	190	10	30	1.55	6.75e+2	8.25e+3	30	0.88	6.82e+2	4.39e+3
		100	30	61.11	6.85e+3	1.69e+5	30	34.38	6.88e+3	8.82e+4
		1000	24	480.61	6.74e+3	1.70e+5	27	592.45	1.24e+4	1.82e+5
30	435	10	30	14.62	3.39e+3	6.52e+4	30	7.58	3.52e+3	3.24e+4
		100	25	614.89	3.55e+4	1.16e+6	27	461.70	5.04e+4	8.34e+5
		1000	12	695.62	4.76e+3	1.27e+5	14	576.20	7.51e+3	9.94e+4
40	780	10	30	115.76	1.57e+4	4.10e+5	30	53.22	1.56e+4	1.88e+5
		100	17	776.65	3.21e+4	1.08e+6	18	473.66	3.88e+4	6.37e+5
		1000	7	1266.34	6.69e+3	1.42e+5	9	856.06	9.13e+3	9.26e+4
50	1225	10	30	523.91	4.73e+4	1.44e+6	30	229.78	4.71e+4	6.52e+5
		100	10	143.25	7.26e+3	1.53e+5	13	580.32	3.41e+4	5.75e+5
		1000	7	1194.76	3.83e+3	9.81e+4	9	848.74	5.92e+3	6.64e+4
60	1770	10	25	592.07	4.26e+4	1.29e+6	30	584.95	9.07e+4	1.36e+6
		100	9	249.59	9.04e+3	2.09e+5	12	718.41	3.50e+4	5.77e+5
		1000	2	859.83	2.08e+3	5.77e+4	3	1188.03	6.47e+3	7.58e+4

Table 6 Using `BB-SD` with and without warmstart, applying dropping rule `d2` – focus on CPU time

N	E	#sc	d2 no ws				d2 with ws			
			#sol	median (s)	min (s)	max (s)	#sol	median (s)	min (s)	max(s)
20	190	10	30	0.515	0.01	8.2	30	0.29	0.01	4.96
		100	30	13.14	0.21	648.88	30	6.80	0.13	380.00
		1000	24	127.41	2.06	3273.04	27	97.18	1.01	3444.12
30	435	10	30	3.87	0.08	135.63	30	1.96	0.05	67.2
		100	25	118.93	2.01	3388.87	27	71.67	1.01	2481.13
		1000	12	124.66	54.08	3257.92	14	107.24	26.8	2097.43
40	780	10	30	44.545	0.27	824.24	30	44.545	0.27	824.24
		100	17	92.88	1.26	3235.81	18	85.29	0.75	2556.65
		1000	7	1363.21	215.43	2525.02	9	790.28	95.72	1798.67
50	1225	10	30	120.555	0.53	2087.87	30	53.99	0.31	884.55
		100	10	63.96	19.02	522.4	13	56.1	8.44	3495.11
		1000	7	785.54	194.14	3480.99	9	427.07	79.56	2027.78
60	1770	10	25	292.88	4.08	2955.47	30	154.38	1.93	2411.76
		100	9	152.7	38.17	1102.45	12	84.66	19.25	3229.65
		1000	2	859.825	859.07	860.58	3	437.11	350.52	2776.47

combination of $|N|$ and #sc, the number of instances solved within the time limit of one hour (#sol), the average running times in seconds (time) and the average number of nodes (#nodes). We recall that the averages are taken considering the results on 30 instances each. For a further comparison, we also report in Table 8 the median, the minimum and the maximum running time attained in seconds for both `BB-SD` and `CPLEX`. Performance profiles are presented in Fig. 6.

We can notice that `BB-SD` strongly ouperforms `CPLEX` when considering instances with 10 and 100 scenarios. As already highlighted before, the higher the number of scenarios, the harder the instances become. Still, also when dealing with instances containing 1000 scenarios, `BB-SD` shows better performance with respect to `CPLEX`. On instances with 10 scenarios, we see that `BB-SD` is able to solve all instances within the time limit, while `CPLEX` fails in 46 cases. For instances on 100 scenarios, `BB-SD` and `CPLEX` show 50 and 75 failures, respectively, while for instances with 1000 scenarios, `BB-SD` and `CPLEX` show 88 and 95 failures.

4.2 Traveling Salesman Problem

Given an undirected, complete, and weighted graph $G = (N, E)$, the Traveling Salesman problem consists in finding a path starting and ending at a given vertex $v \in N$ such that all vertices in the graph are visited exactly once and the sum of the weights of its constituent edges is minimized. Our approach uses the following formulation of the Traveling Salesman problem:

$$
\begin{aligned}
\min \ & \max_{c \in U} c^\top x \\
\text{s.t.} \ & \sum_{e \in \delta(i)} x_e = 2, && \forall i \in N \\
& \sum_{e \in \delta(X)} x_e \geq 2, && \forall \emptyset \neq X \subsetneq N \\
& x \in \{0, 1\}^{|E|}
\end{aligned}
\qquad \text{(r-TSP)}
$$

Fig. 5 Comparison of `BB-SD`
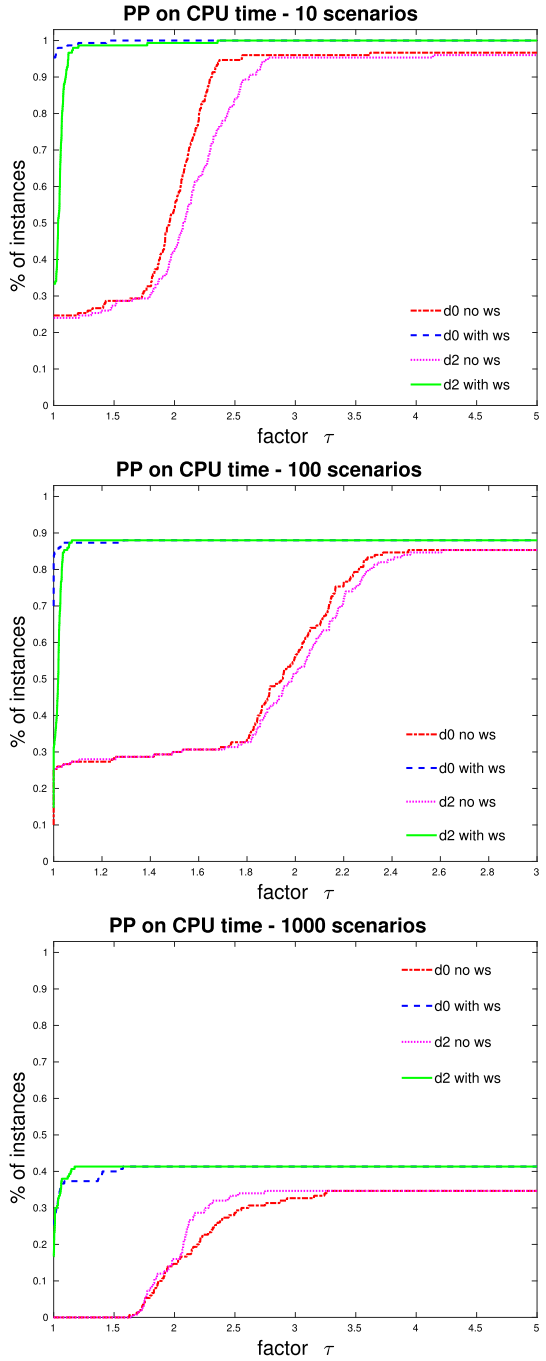with and without warmstart

Table 7 Comparison between `BB-SD` and `CPLEX` on r-MST instances

N	E	#sc	BB-SD			CPLEX		
			#sol	time (s)	#nodes	#sol	time (s)	#nodes
20	190	10	30	0.88	6.81e+2	30	0.36	2.15e+3
		100	30	34.38	6.88e+3	30	11.85	3.65e+4
		1000	27	592.45	1.24e+4	30	295.54	1.47e+5
30	435	10	30	7.58	3.52e+3	30	39.19	6.80e+4
		100	27	461.70	5.04e+4	27	399.98	7.91e+5
		1000	14	576.20	7.51e+3	17	649.76	2.42e+5
40	780	10	30	53.22	1.56e+4	21	304.25	3.56e+5
		100	18	473.66	3.88e+4	11	897.02	9.29e+5
		1000	9	856.06	9.13e+3	4	1159.91	3.29e+5
50	1225	10	30	229.78	4.71e+4	12	855.83	7.43e+5
		100	13	580.32	3.41e+4	4	685.22	5.17e+5
		1000	9	848.74	5.92e+3	3	3247.92	5.36e+5
60	1770	10	30	584.95	9.07e+4	11	596.26	4.24e+5
		100	12	718.41	3.50e+4	3	1167.98	9.92e+5
		1000	3	1188.03	6.47e+3	1	577.20	8.76e+4

Table 8 Comparison between `BB-SD` and `CPLEX` on r-MST instances – focus on CPU time

| N	E	#sc	BB-SD				CPLEX			
			#sol	median (s)	min (s)	max (s)	#sol	median (s)	min (s)	max (s)
20	190	10	30	0.29	0.01	4.96	30	0.20	0.01	1.58
		100	30	6.79	0.13	380.00	30	4.83	0.16	110.94
		1000	27	97.18	1.01	3444.12	30	38.80	1.22	2908.79
30	435	10	30	1.96	0.05	67.20	30	3.38	0.27	369.54
		100	27	71.67	1.01	2481.13	27	91.24	1.04	3197.07
		1000	14	107.23	26.80	2097.43	17	319.39	15.54	3194.27
40	780	10	30	20.70	0.14	376.32	21	59.16	5.66	1793.91
		100	18	85.29	0.75	2556.65	11	431.04	6.17	2639.18
		1000	9	790.28	95.72	1798.67	4	793.67	126.42	2925.89
50	1225	10	30	53.99	0.31	884.55	12	700.54	9.56	2249.47
		100	13	56.10	8.44	3495.11	4	774.44	159.68	1032.33
		1000	9	427.07	79.56	2027.78	3	3496.58	2748.87	3498.32
60	1770	10	30	154.38	1.93	2411.76	11	167.14	24.79	2799.27
		100	12	84.66	19.25	3229.65	3	485.80	317.63	2700.50
		1000	3	437.11	350.52	2776.47	1	577.20	577.20	577.20
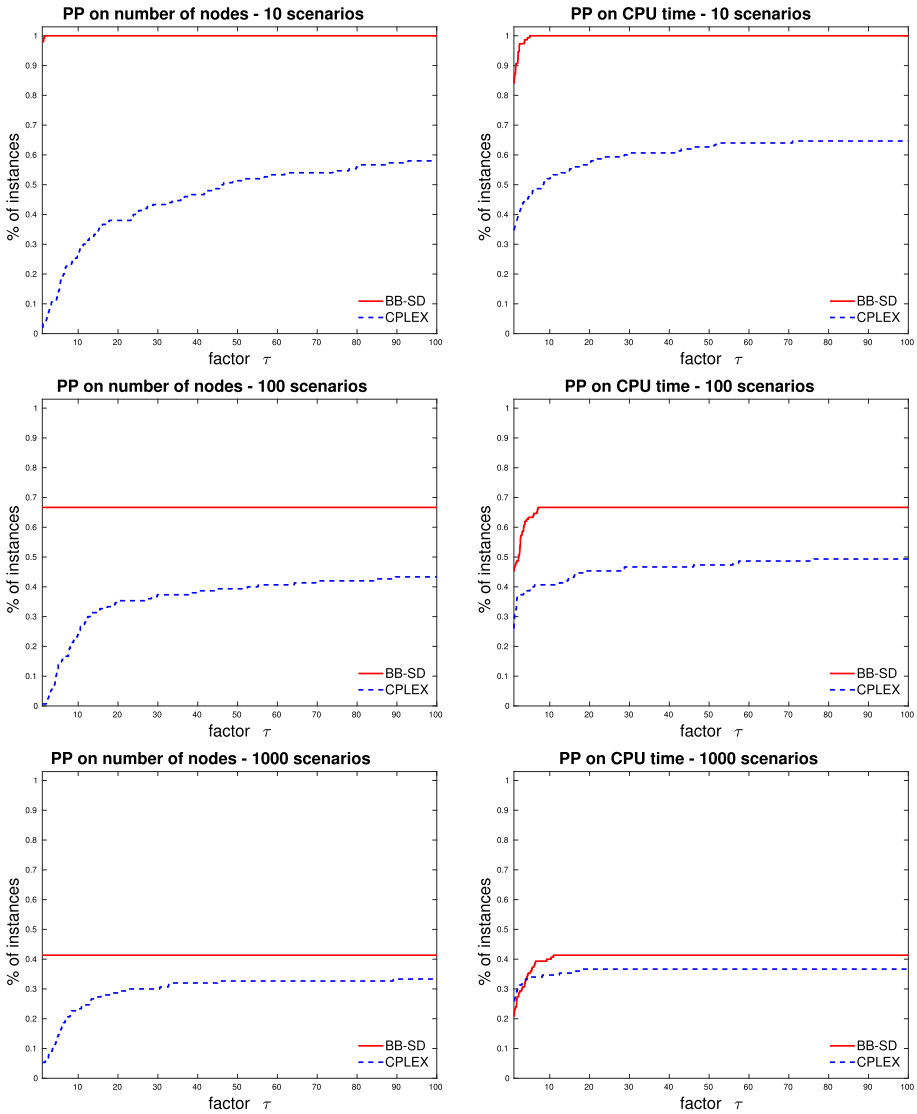
Fig. 6 Comparison between `BB-SD` and `CPLEX` on r-MST instances

The constraints in (r-TSP) define the set of feasible cycles starting and ending at a given vertex $v \in N$, also called *tours*. The first constraint is known as *degree constraint* and guarantees that the tour visits each vertex in the graph exactly once. The remaining constraints, known as *subtour elimination constraints*, guarantee that the solution does not decompose into several subtours. The number of inequalities is again exponential and for `CPLEX` we use essentially the same separation algorithm as for the Spanning Tree problem; see Sect. 4.1.

For our tests, we consider 10 instances from the TSPLIB library [23]. For each instance, we generate different scenarios by adding to the nominal costs a random unit vector multiplied by some coefficient. This vector has non-negative components, to avoid negative distances, and

the coefficients are 1, 2 and 3, as for the r-MST case. We again consider three different numbers of scenarios, namely 10, 100, and 1000, thus producing a benchmark of 90 instances in total. As mentioned above, we realized the linear oracle (Oracle) by using the solver `Concorde` (release 03.12.19) [10]. We used the default version and solved each linear problem exactly. In particular, in each linear oracle call an NP-hard problem is solved, so that the time needed by Oracle is now much larger than the time needed by Oracle , unlike in the MST case. Therefore, eliminating variables is not effective: it would slightly reduce the overall running time for Oracle , while increasing the number of iterations and hence increasing the overall running time for Oracle . For this reason, for our tests, we only consider `SD` where no dropping rule is applied.

In the following, we compare the performance of `SD` applied to solve the continuous relaxation of the instances considered and the performance of `CPLEX` at the root node. We notice that `CPLEX` minimizes the non-linear objective function $\max_{c \in U} c^\top x$ over the subtour relaxation of the problem, while in our formulation we implicitly optimize the same function over the convex hull of feasible tours, thus obtaining a tighter lower bound. However, our approach needs to solve NP-hard problems to achieve this. It is thus not surprising that the computing time needed by `SD` to solve the relaxation is often larger than the time needed by `CPLEX` to solve its weaker relaxation. However, when requiring `CPLEX` to obtain the same stronger bound, the required computational time increases significantly. In Table 9, we show the results for the TSP instances. For every instance and every number of scenarios, we report the average bound and computing time in seconds obtained by `SD` (SD root node) and by `CPLEX` (CPLEX root node) to solve the continuous relaxation. In the last column, we report the time in seconds needed by `CPLEX` to obtain the same bound as the SD bound (CPLEX – SD bound). The table shows that, on average, the SD bound is much stronger than the subtour relaxation bound, but it is obtained in a longer time. Furthermore, the time needed by `CPLEX` to compute the same bound as `SD` is often much larger for instances with a large number of scenarios, e.g. $\#sc = 1000$.

5 Conclusion

We presented an algorithm for the exact solution of strictly robust counterparts of combinatorial optimization problems, entirely based on a linear optimization oracle for the underlying problem. Concentrating on the discrete scenario case, our experimental evaluation shows that the approach is competitive both in case the underlying problem is very easy to solve, as in the MST case, and in case it is a hard problem, as in the TSP case. In particular, in the latter case, we have seen that solving the underlying problem to optimality can be beneficial even when it is NP-hard: in the same amount of time, our approach produces much better dual bounds than CPLEX based on a linearized IP formulation of the problem, using the standard subtour formulation.

We emphasize again that our approach is not restricted to the case of discrete uncertainty. However, Oracle must be adapted when considering other classes of uncertainty sets and an empirical evaluation is an open topic for future research. In case of ellipsoidal uncertainty, Oracle turns out to be a second-order cone program. As mentioned above, since f is a smooth function in this case, cycling is not possible even if the most aggressive dropping rule (drop) is used. For the case of polyhedral uncertainty, Oracle can again be realized as a linear program, and the statement of Theorem 5 holds analogously.

Table 9 Results for r-TSP, continuous relaxations (averaged on the different scenarios)

instance	#sc	SD root node		CPLEX		CPLEX
				root node		SD bound
		bound	time (s)	bound	time (s)	time (s)
brazil58	10	46031.5	2.09	41982.9	0.03	1.77
	100	48086.6	12.35	43722.6	0.21	9.75
	1000	49278.8	19.09	44810.6	2.68	256.33
dantzig42	10	1158.0	1.59	1093.1	0.02	0.26
	100	1203.6	0.97	1143.7	0.17	0.99
	1000	1230.9	6.92	1166.3	3.50	15.33
fri26	10	1622.8	0.45	1550.9	0.01	0.09
	100	1691.7	1.32	1624.7	0.07	0.39
	1000	1721.6	1.44	1663.8	0.64	2.75
gr120	10	9801.5	15.96	9564.9	0.16	203.93
	100	9977.8	68.51	9759.4	1.03	672.07
	1000	10131.3	68.74	9886.2	18.34	> 3600.00
gr17	10	3911.2	0.28	3623.6	0.01	0.04
	100	4053.9	0.57	3676.5	0.02	0.19
	1000	4248.6	0.73	3925.4	0.16	1.94
gr21	10	4928.4	0.22	4903.8	0.01	0.01
	100	5138.5	0.22	5104.6	0.03	0.15
	1000	5301.1	0.23	5277.2	0.24	0.76
gr24	10	2202.9	0.30	2153.9	0.01	0.04
	100	2272.6	0.55	2251.5	0.04	0.16
	1000	2359.8	1.76	2318.3	0.37	1.52
gr48	10	7642.8	0.92	7417.6	0.02	0.43
	100	7907.4	6.38	7668.2	0.14	3.10
	1000	8034.1	8.38	7789.8	1.57	31.01
hk48	10	18545.0	0.55	17895.9	0.02	0.40
	100	18889.0	1.50	18377.3	0.14	2.18
	1000	19190.6	6.74	18854.7	1.63	21.43
swiss42	10	2051.0	1.71	1970.7	0.03	0.50
	100	2128.4	1.41	2064.1	0.16	1.25
	1000	2185.3	9.20	2117.2	2.13	19.31

The investigation of other generalizations of our approach is left as future work. In particular, it may be interesting to extend it to more general classes of uncertain objective functions, e.g., of the form $c^\top g(x)$, where a convex function $g\colon \mathbb{R}^n \to \mathbb{R}^m$ is given and the coefficients $c \in \mathbb{R}^m_+$ are uncertain. In this case, the function f describing the worst case over all scenarios is still a convex function, and it suffices to adapt the oracle Oracle .

Data availability The datasets generated and analysed during the current study are provided by the corresponding author by reasonable request.

References

1. Bertsekas, D.P.: Convex Optimization Theory. Athena Scientific, Belmont, MA (2009)
2. Bertsekas, D.P.: Convex Optimization Algorithms. Athena Scientific, Belmont, MA (2015)
3. Bertsekas, D.P., Huizhen, Yu.: A unifying polyhedral approximation framework for convex optimization. SIAM J. Optim. **21**(1), 333–360 (2011). https://doi.org/10.1137/090772204
4. Bettiol, E., Létocart, L., Rinaldi, F., Traversi, E.: A conjugate direction based simplicial decomposition framework for solving a specific class of dense convex quadratic programs. Comput. Optim. Appl. **75**(2), 321–360 (2020). https://doi.org/10.1007/s10589-019-00151-4
5. Buchheim, C.: A note on the nonexistence of oracle-polynomial algorithms for robust combinatorial optimization. Discret. Appl. Math. **285**, 591–593 (2020). https://doi.org/10.1016/j.dam.2020.07.002
6. Buchheim, C., Kurtz, J.: Min-max-min robust combinatorial optimization. Math. Program. **163**(1–2), 1–23 (2017). https://doi.org/10.1007/s10107-016-1053-z
7. Buchheim, C., Kurtz, J.: Robust combinatorial optimization under convex and discrete cost uncertainty. EURO J. Comput. Optim. **6**(3), 211–238 (2018). https://doi.org/10.1007/s13675-018-0103-0
8. Buchheim, C., De Santis, M.: An active set algorithm for robust combinatorial optimization based on separation oracles. Math. Program. Comput. **11**(4), 755–789 (2019). https://doi.org/10.1007/s12532-019-00160-8
9. Buchheim, C., De Santis, M., Rinaldi, F., Trieu, L.: A Frank-Wolfe based branch-and-bound algorithm for mean-risk optimization. J. Global Optim. **70**(3), 625–644 (2018). https://doi.org/10.1007/s10898-017-0571-4
10. Concorde TSP solver. https://www.math.uwaterloo.ca/tsp/concorde/index.html
11. Dolan, E.D., More, J.J.: Benchmarking optimization software with performance profiles. Math. Progr. **91**, 201–213 (2002)
12. Fischetti, M., Monaci, M.: Cutting plane versus compact formulations for uncertain (integer) linear programs. Math. Progr. Comput. (2012). https://doi.org/10.1007/s12532-012-0039-y
13. Hearn, Donald W, Lawphongpanich, S., Ventura, Jose A.: Restricted simplicial decomposition: Computation and extensions. In: Computation Mathematical Programming, pp 99–118. Springer, (1987). https://doi.org/10.1007/BFb0121181
14. Holloway, C.A.: An extension of the frank and wolfe method of feasible directions. Math. Program. **6**(1), 14–27 (1974). https://doi.org/10.1007/BF01580219
15. IBM ILOG CPLEX Optimizer, (2021). https://www.ibm.com/it-it/analytics/cplex-optimizer
16. Kämmerling, N., Kurtz, J.: Oracle-based algorithms for binary two-stage robust optimization. Comput. Optim. Appl. **77**(2), 539–569 (2020). https://doi.org/10.1007/s10589-020-00207-w
17. Kouvelis, P., Gang, Y.: Robust Discrete Optimization and its Applications. Springer, Berlin (1996)
18. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedi. Am. Math. Soc. **7**(1), 48–50 (1956). https://doi.org/10.2307/2033241
19. Kurtz, J.: New complexity results and algorithms for min-max-min robust combinatorial optimization. arXiv:2106.03107, (2021)
20. Lan, G.: First-order and Stochastic Optimization Methods for Machine Learning. Springer Nature, New York (2020)
21. Larsson, T., Patriksson, M.: Simplicial decomposition with disaggregated representation for the traffic assignment problem. Transp. Sci. **26**(1), 4–17 (1992). https://doi.org/10.1287/trsc.26.1.4
22. Mutapcic, A., Boyd, S.: Cutting-set methods for robust convex optimization with pessimizing oracles. Optim. Methods Softw. **24**(3), 381–406 (2009). https://doi.org/10.1080/10556780802712889
23. TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95

24. Ventura, J.A., Hearn, D.W.: Restricted simplicial decomposition for convex constrained problems. Math. Program. **59**(1), 71–85 (1993). https://doi.org/10.1007/BF01581238
25. Von Hohenbalken, B.: Simplicial decomposition in nonlinear programming algorithms. Math. Program. **13**(1), 49–68 (1977). https://doi.org/10.1007/BF01584323

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.