Computational Optimization and Applications (2021) 79:561–599
https://doi.org/10.1007/s10589-021-00282-7

Computing mixed strategies equilibria in presence of switching costs by the solution of nonconvex QP problems

G. Liuzzi[1] · M. Locatelli[2] · V. Piccialli[3] · S. Rass[4]

Received: 8 September 2020 / Accepted: 3 May 2021 / Published online: 22 May 2021

Abstract
In this paper we address game theory problems arising in the context of network security. In traditional game theory problems, given a defender and an attacker, one searches for mixed strategies which minimize a linear payoff functional. In the problems addressed in this paper an additional quadratic term is added to the minimization problem. Such term represents *switching costs*, i.e., the costs for the defender of switching from a given strategy to another one at successive rounds of a Nash game. The resulting problems are nonconvex QP ones with linear constraints and turn out to be very challenging. We will show that the most recent approaches for the minimization of nonconvex QP functions over polytopes, including commercial solvers such as `CPLEX` and `GUROBI`, are unable to solve to optimality even test instances with $n = 50$ variables. For this reason, we propose to extend with them the current benchmark set of test instances for QP problems. We also present a spatial branch-and-bound approach for the solution of these problems, where a predominant role is played by an optimality-based domain reduction, with multiple solutions of LP problems at each node of the branch-and-bound tree. Of course, domain reductions are standard tools in spatial branch-and-bound approaches. However, our contribution lies in the observation that, from the computational point of view, a rather aggressive application of these tools appears to be the best way to tackle the proposed instances. Indeed, according to our experiments, while they make the computational cost per node high, this is largely compensated by the rather slow growth of the number of nodes in the branch-and-bound tree, so that the proposed approach strongly outperforms the existing solvers for QP problems.

Keywords Game theory · Nonconvex quadratic programming problems · Branch-and-bound · Bound-tightening

M. Locatelli
marco.locatelli@unipr.it

Extended author information available on the last page of the article

1 Introduction

Consider a finite two-person zero-sum game Γ, composed from a player set $N = \{1, 2\}$, each member thereof having a finite strategy space S_1, S_2 associated with it, and a utility function $u_i : S_1 \times S_2 \to \mathbb{R}$ for all $i \in N$. We assume a zero-sum Nash game, making $u_2 := -u_1$ hereafter, and letting the players choose their actions simultaneously and stochastically independent of one another (contrary to a Stackelberg game, where one player would follow the other, which we do not consider here). The game is then the triple $\Gamma = (N, \mathcal{S} = \{S_1, S_2\}, H = \{u_1, -u_1\})$, and is most compactly represented by giving only the payoff function u_1 in matrix form (since the strategy spaces are finite) as

$$\mathbf{A} \in \mathbb{R}^{|S_1|\times|S_2|} = \big(u_1(x, z)\big)_{(x,z)\in S_1\times S_2}.$$

An equilibrium in Γ is a simultaneous optimum for both players w.r.t. u_1. Assuming a maximizing first player, an equilibrium is a pair (x^*, z^*) satisfying the saddle-point condition

$$u_1(x, z^*) \le u_1(x^*, z^*) \le u_1(x^*, z) \quad \forall (x, z) \in S_1 \times S_2.$$

It is well known that many practical games do not have such an equilibrium point; as one of the simplest instances, consider the classical rock-scissors-paper game, represented by the payoff matrix

$$\begin{array}{r c} & \begin{array}{ccc} \text{rock} & \text{scissors} & \text{paper} \end{array} \\ \begin{array}{r} \text{rock} \\ \text{scissors} \\ \text{paper} \end{array} & \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}. \end{array}$$

This game has no equilibria in pure strategies: any fixed choice of rock, scissors or paper would imply a constant loss for the first player (and likewise for the second player). This means that player 1 is forced to *randomize* its actions in every round of the game, and this concept leads to the idea of mixed extensions of a game, which basically changes the above optimization problem into one over the convex hulls $\Delta(S_1), \Delta(S_2)$ of the action spaces, rather than the finite sets S_1, S_2. An element of $\Delta(S_i)$ is then a probability distribution over the elements of the support S_i, and prescribes to pick a move at random whenever the game is played.

The game rewards its players after each round, and upon every new round, both players are free to choose another element from their action space at random. Implicitly, this choice is without costs, but what if not? Many real life instances of games *do incur* a cost for changing one's action from $a_1 \in S_1$ in the first to some distinct $a_2 \in S_1$ in the next round. Matrix games cannot express such costs in their payoff functions, and more complex game models such as sequential or stochastic games come with much more complicated models and equilibrium concepts. The goal of this work is to retain the simplicity of matrix games but endow them with the ability to include switching costs with the minimal natural (modeling) effort.

The area of system security [2, 30] offers rich examples of such instances, such as (among many):

- Changing passwords [25]: if the currently chosen password is p_1 and we are obliged to pick a fresh password (say, different from the last couple of passwords that we had in the past), the use of the new password $p_2 \neq p_1$ induces quite some efforts, as we have to memorize the password, while choosing it as hard as possible to guess. The "cost" tied to the change is thus not monetary, but the cognitive efforts to create and memorize a new password. This effort can make people reluctant to change their passwords (or write them down, or use a very similar password for the new one).
- Changing computer/server configurations: this usually means taking a computer (e.g., a server) offline for a limited time, thus cutting down productivity perhaps, and hence causing costs. If security is drawn from randomly changing configurations (and passwords, resp. password changing rules are only one special case here), then this change incurs costs by temporal outages of IT infrastructure for the duration of the configuration change, and the efforts (person-hours) spent on applying this change. This is why server updates or patches are usually done over nights or weekends, when the loads are naturally low. If the optimization would, however, prescribe a rather frequent change of configurations at random intervals, this can quickly become a practical inhibitor, unless the switching costs are accounted for by optimization.
- Patrolling and surveillance [4, 24]: consider a security guard on duty to repeatedly check a few locations, say A, B, ..., E, which are connected at distances as depicted in Fig. 1. This is a chasing-evading game with the guard acting as player 1 against an intruder being player 2, and with the payoff function u being an indicator of whether the guard caught the intruder at location $i \in \{A,...,E\}$, or whether the two missed each other. This is yet another instance of a game with all equilibria in mixed strategies, but with the unpleasant side-effect for the guard that gets the prescription to randomly spot check distant locations to "play" the equilibrium $\mathbf{x}^*$, the guard would have to move perhaps long distances between the locations. For example, if it is at A in round 1 and the next sample from the random distribution $\mathbf{x}^* \in \Delta(\{A,...,E\})$ tells to check point E next, the shortest path would be of length $1 + 3 + 2 = 6$ over C. Starting from A, however, it would be

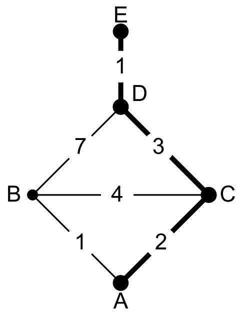


Fig. 1 Example of spot checking game on a graph

shorter and hence more convenient for the guard to check location B first along the way, but this would mean deviating from the equilibrium! A normal game theoretic equilibrium calculation does not consider this kind of investment to change the current strategy. This may not even count as bounded rationality, but simply as acting "economic" from the guard's perspective. But acting economically here is then not governed by a utility maximizing principle, but rather by a cost minimization effort.

Generalizing the patrolling game example, the issue applies to all sorts of moving target defense: for example, changing the configuration of a computer system so as to make it difficult for an attacker to break in, often comes with high efforts and even risks for the defending player 1 (the system administrator), since it typically means taking off machines from the network, reconfiguring them to close certain vulnerabilities, and then putting them back to work hoping that everything restarts and runs smoothly again. A normal game theoretic model accounts only for the benefits of that action, but not for the cost of *taking* the action.

Including the cost to switch from one action to the next is more complicated than just assigning a cost function $c : S_1 \to \mathbb{R}$ and subtracting this from the utilities to redefine them as $u_1'(i,j) = u_1(i,j) - c(i)$, since the cost to play a_i will generally depend on the previous action a_j played in the previous round.

We can model this sort of payoff by another function $s : S_1 \times S_1 \to \mathbb{R}$ that we call the *switching cost*. The value of $s(i, j)$ is then precisely the cost incurred to change the current action $i \in S_1$ into the action $j \in S_1$ in the next round of the game. Intuitively, this adds another payoff dimension to the game, where a player, w.l.o.g. being player 1 in the following, plays "against itself", since the losses are implied by its own behavior. While the expected payoffs in a matrix game $\mathbf{A}$ under mixed strategies $\mathbf{x} \in \Delta(S_1), \mathbf{z} \in \Delta(S_2)$ are expressible by the bilinear functional $\mathbf{x}^T\mathbf{A}\mathbf{z}$, the same logic leads to the hypothesis that the switching cost should on average be given by the quadratic functional $\mathbf{x}^T\mathbf{S}\mathbf{x}$, where the switching cost matrix is given, like the payoff matrix above, as

$$\mathbf{S} \in \mathbb{R}^{|S_1|\times|S_1|} = (s(x, w))_{(x,w)\in S_1\times S_1}.$$

This intuition is indeed right [26], but for a rigorous problem statement, we will briefly recap the derivation given independently later by [32] to formally state the problem.

1.1 Paper Outline

The paper is structured as follows. In Sect. 2 we give a formal description of the problem as a nonconvex QP one with linear constraints, and we report a complexity result, proved in Appendix 1 In Sect. 3 we present a (spatial) branch-and-bound approach for the problem, putting a particular emphasis on the bound-tightening procedure, which turns out to be the most effective tool to attack it. In Sect. 4 we present a real-life instance. In Sect. 5 we present some computational experiments. We first describe the set of test instances. Next, we discuss the performance of existing solvers over these instances. Finally, we present and comment the computational results attained by the proposed approach. In Sect. 6 we draw some conclusions and discuss possible future developments.

1.2 Statement of contribution

The main contributions of this work are:

- addressing an application of game theory arising in the context of network security, where switching costs come into play, and showing that the resulting problem can be reformulated as a challenging nonconvex QP problem with linear constraints;
- introducing a large set of test instances, which turn out to be very challenging for existing QP solvers and, for this reason, could be employed to extend the current benchmark set of QP problems (see [11]);
- proposing a branch-and-bound approach for the solution of the addressed QP problems, based on standard tools, but with the empirical observation that a very aggressive use of bound-tightening techniques, with a high computational cost per node of the branch-and-bound tree, is the key for an efficient solution of these problems.

2 Formal description of the problem

Let the game come as a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$, where n and m are the number of strategies for player 1 and 2, respectively, with equilibrium $(\mathbf{x}^*, \mathbf{z}^*)$, and let it be repeated over the time $t \in \mathbb{N}$. At each time t, let $X_t \sim \mathbf{x}^*$ be the random action sampled from the equilibrium distribution over the action space (with $\mathbf{x}^*$ being the optimal distribution). In a security setting and zero-sum game, neither player has an interest of being predictable by its opponent, so we assume stochastic independence of the action choices by both players between any two repetitions of the game. Then, we have $\Pr(X_{t-1} = i, X_t = j) = \Pr(X_{t-1} = i) \cdot \Pr(X_t = j)$, so that any future system state remains equally predictable whether or not the current state of the system is known. Hence, the switching cost can be written as

$$\begin{aligned} s(X_{t-1}, X_t) &= \sum_{i=1}^{n}\sum_{j=1}^{n} s_{ij} \cdot \Pr(X_{t-1} = i, X_t = j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} s_{ij} \cdot \Pr(X_{t-1} = i) \cdot \Pr(X_t = j) = \mathbf{x}^T\mathbf{S}\mathbf{x}. \end{aligned}$$

With this, player 1’s payoff functional becomes vector-valued now as

$$\mathbf{u}_1 : \Delta(S_1) \times \Delta(S_2) \to \mathbb{R}^2, \quad (\mathbf{x}, \mathbf{z}) \mapsto \begin{pmatrix} u_1(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{A}\mathbf{z} \\ s(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{S}\mathbf{x} \end{pmatrix}, \tag{1}$$

and the game is multi-objective for the first player. As we are interested mostly in the best behavior for player 1 and the analysis would be symmetric from player 2's perspective, we shall not explore the view of the second player hereafter.

Remark 1 The game could be equally well multi-objective for the second player too, and in fact a practical instance of such a situation may also come from security: it could be in an adversary's interest to "keep the defender busy", thus causing much friction by making the defender move fast from one place to the other. This is yet just another instance of a denial-of-service attack, to which such a game model would apply.

For the sake of computing a multi-objective equilibrium, more precisely a Pareto-Nash equilibrium, the algorithm in [27] based on the method laid out in [16] proceeds by scalarizing (1) by choice of some $\alpha \in (0, 1)$, to arrive at the real-valued goal function

$$\alpha \cdot \mathbf{x}^T\mathbf{A}\mathbf{z} + (1 - \alpha) \cdot \mathbf{x}^T\mathbf{S}\mathbf{x},$$

for the first player to optimize. Now, the usual way from here to an optimization problem for player 1 involving a rational opponent applies as for standard matrix games [26]: let $\mathbf{e}_i \in \mathbb{R}^m$ be i-th unit vector, then $\arg\max_{\mathbf{z}\in\Delta(S_2)}(\mathbf{x}^T\mathbf{A}\mathbf{z}) = \arg\max_i(\mathbf{x}^T\mathbf{A}\mathbf{e}_i)$. After introducing the additional variable v, the resulting problem becomes

$$\begin{array}{lll} \min & (1-\alpha)\cdot \mathbf{x}^T\mathbf{S}\mathbf{x} + \alpha v & \\ \text{s.t.} & v \geq \mathbf{x}^T\mathbf{A}\mathbf{e}_i & i = 1, \dots, m \\ & \sum_{j=1}^n x_j = 1 & \\ & x_j \geq 0 & j = 1, \dots, n, \end{array} \tag{2}$$

which is almost the familiar optimization problem to be solved for a Nash equilibrium in a finite matrix game. It differs from the well known linear program only in the quadratic term, and, in fact, the equilibrium problem for matrix games is recovered by substituting $\alpha = 1$ in (2). Note that the matrix $\mathbf{S}$ in the quadratic term will (in most cases) have a zero diagonal, nonnegative off-diagonal entries, be indefinite and not symmetric in general (patrolling game example given above already exhibits a variety of counterexamples leading to nonsymmetric distance matrices $\mathbf{S}$ if the graph is directed). Of course, symmetry of $\mathbf{S}$ can be easily recovered, so in what follows we will assume that $\mathbf{S}$ is symmetric. The two extreme values $\alpha = 0$ and $\alpha = 1$ give rise to simple problems. Indeed, as already commented, for $\alpha = 1$ the problem is an LP one, while for $\alpha = 0$ is a Standard QP (StQP) problem, which is in general NP-hard (e.g., in view of the reformulation of the max clique problem as an StQP problem, see [18]), but is trivial in the case of zero diagonal and nonnegative off-diagonal entries (each vertex of the unit simplex is a globally optimal solution). For what concerns the intermediate values $\alpha \in (0, 1)$ we can prove the following result, stating the complexity of problem (2) .

Theorem 1 *Problem* (2)*is NP-hard.*

Proof See Appendix 1. □

Remark 2 The dependence of next actions on past ones extends to other scenarios too: for example, if the game is about coordination in wireless settings (e.g., collaborative drones), the players, e.g., drones, share a common communication channel. Every exchange of information occupies that channel for a limited period of time, thus constraining what the other players can do at the moment. Such effects can be described by stochastic games, but depending on how far the effect reaches in the future, backward inductive solution methods may become computationally infeasible [14]; likewise, extending the strategy space to plan ahead a fixed number of k steps (to account for one strategy determining the next k repetitions of the game) may exponentially enlarge the strategy space (by a factor of $2^{O(k)}$, making the game infeasible to analyze if k is large). Games with switching cost offer a neat bypass to that trouble: if an action is such that it occupies lots of resources for a player, thus preventing it from taking further moves in the next round of the game, we can express this as a switching cost. Assume, for instance, that an action in a game Γ is such that the player is blocked for the next k rounds, then the switching cost is k-times the expected utility $\overline{u}$ (with the expectation taken over the equilibrium distribution played by the participants) that these next k rounds would give. Virtually, the situation is thus like if the player would have paid the total average gain over the next rounds where it is forced to remain idle (thus gaining nothing):

$$\overline{u} \underbrace{-k \cdot \overline{u}}_{\text{switching cost}} + \underbrace{\overline{u} + \cdots + \overline{u}}_{\substack{\text{virtual payoffs} \\ \text{over } k \text{ rounds}}} = \overline{u} + \underbrace{0 + 0 + \ldots + 0}_{\substack{\text{practical payoffs} \\ \text{by being idle} \\ \text{for } k \text{rounds}}} \tag{3}$$

Expression (3) will in practice be only an approximate identity, since we assumed that the game, viewed as a stochastic process, has already converged to stationarity (so that the equilibrium outcome $\overline{u}$ is actually rewarded). The speed of convergence, indeed, can itself be of interest to be controlled in security applications using moving target defenses [32]. The crucial point of modeling a longer lasting effect of the current action like described above, however, lies in the avoidance of complexity: expression (3) has no issues with large k, while more direct methods of modeling a game over k rounds, or including a dependency on the last k moves, is relatively more involved (indeed, normal stochastic games consider a first-order Markov chain, where the next state of the game depends on the last state; the setting just described would correspond to an order k chain, whose conversion into a first order chain is also possible, but complicates matters significantly).

3 A branch and bound approach

After incorporating parameter α into the definitions of matrix $\mathbf{S}$ and vectors $\mathbf{A}_j$, $j = 1, \dots, m$, and after introducing the vector of variables $\mathbf{y}$, problem (2) can be rewritten as the following problem with bilinear objective function and linear constraints:

$$\begin{array}{lll} \min & F(\mathbf{x}, \mathbf{y}, v) := \frac{1}{2} \sum_{i=1}^{n} x_i y_i + v & \\ & v \geq \mathbf{A}_j^T \mathbf{x} & j = 1, \dots, m \\ & y_i = \mathbf{S}_i \mathbf{x} & i = 1, \dots, n \\ & \mathbf{x} \in \Delta_n, & \end{array} \tag{4}$$

where $\mathbf{S}_i$ denotes the i-th row of matrix $\mathbf{S}$ and Δ_n denotes the n-dimensional unit simplex. In what follows we will denote by P the feasible region of this problem, and by $P_{\mathbf{x},\mathbf{y}}$ its projection over the space of $\mathbf{x}$ and $\mathbf{y}$ variables.

Each node of the branch-and-bound tree is associated to a box $B = [\boldsymbol{\ell}_\mathbf{x}, \mathbf{u}_\mathbf{x}] \times [\boldsymbol{\ell}_\mathbf{y}, \mathbf{u}_\mathbf{y}]$, where $\boldsymbol{\ell}_\mathbf{x}, \mathbf{u}_\mathbf{x}$ and $\boldsymbol{\ell}_\mathbf{y}, \mathbf{u}_\mathbf{y}$ denote lower and upper bound vectors for variables $\mathbf{x}$ and $\mathbf{y}$, respectively. An initial box B_0, containing $P_{\mathbf{x},\mathbf{y}}$ is easily computed. It is enough to set $\boldsymbol{\ell}_\mathbf{x} = \mathbf{0}$, $\mathbf{u}_\mathbf{x} = \mathbf{e}$ (the vector whose entries are all equal to one), and

$$\ell_{y_i} = \min_{k=1,\dots,n} S_{ik}, \quad \ell_{u_i} = \max_{k=1,\dots,n} S_{ik}.$$

Note that, although not strictly necessary, we can also bound variable v to belong to an interval. Indeed, we can impose $v \geq 0$ (due to nonnegativity of the entries of vectors $\mathbf{A}_j$, $j = 1, \dots, m$), and

$$v \leq \max_{j=1,\dots,m,\ k=1,\dots,n} A_{jk},$$

which certainly holds at optimal solutions of problem (4). In what follows we describe in detail each component of the branch-and-bound approach, whose pseudo-code is then sketched in Algorithm 1.

3.1 Lower bounds

Given box $B = [\boldsymbol{\ell}_\mathbf{x}, \mathbf{u}_\mathbf{x}] \times [\boldsymbol{\ell}_\mathbf{y}, \mathbf{u}_\mathbf{y}]$, then the well known McCormick underestimating function (see [17])

$$\max \left\{ \ell_{x_i} y_i + \ell_{y_i} x_i - \ell_{x_i} \ell_{y_i}, u_{x_i} y_i + u_{y_i} x_i - u_{x_i} u_{y_i} \right\},$$

can be employed to limit from below the bilinear term $x_i y_i$ over the rectangle $[\ell_{x_i}, u_{x_i}] \times [\ell_{y_i}, u_{y_i}]$. In fact, it turns out that the McCormick underestimating function is the convex envelope of the bilinear term over the given rectangle. Then, after introducing the additional variables f_i, we have that the optimal value of the following LP problem is a lower bound for problem (4) over the box B:

$$L(B) = \min \quad \frac{1}{2}\sum_{i=1}^{n} f_i + v \tag{5a}$$

$$\mathbf{x} \in \Delta_n \tag{5b}$$

$$v \geq \mathbf{A}_j^T \mathbf{x} \quad j = 1, \dots, m \tag{5c}$$

$$y_i = \mathbf{S}_i \mathbf{x} \quad i = 1, \dots, n \tag{5d}$$

$$(\mathbf{x}, \mathbf{y}) \in B \tag{5e}$$

$$f_i \geq \ell_{y_i} x_i + \ell_{x_i} y_i - \ell_{x_i} \ell_{y_i} \quad i = 1, \dots, n \tag{5f}$$

$$f_i \geq u_{x_i} y_i + u_{y_i} x_i - u_{y_i} u_{x_i} \quad i = 1, \dots, n. \tag{5g}$$

The optimal solution of the LP problem will be denoted by $(\mathbf{x}^\star(B), \mathbf{y}^\star(B), \mathbf{f}^\star(B), v^\star(B))$.

3.2 Upper bound

The global upper bound (GUB in what follows) can be initialized with $+\infty$ or, alternatively, if a local search procedure is available, one may run a few local searches from randomly generated starting points, and take the lowest local minimum value as initial GUB value, although, according to our experiments, there is not a significant variation in the computing times if such local searches are performed. During the execution of the branch-and-bound algorithm, each time we compute the lower bound (5) over a box B, its optimal solution is a feasible solution for problem (4) and, thus, we might update the upper bound as follows:

$$GUB = \min\{GUB, F(\mathbf{x}^\star(B), \mathbf{y}^\star(B), v^\star(B))\}.$$

3.3 Branching

The branching strategy we employed is a rather standard one. Given node B, we first compute the quantities:

$$g_i = x_i^\star(B) y_i^\star(B) - f_i^\star(B), \tag{6}$$

measuring the error of McCormick underestimator for each bilinear term $x_i y_i$ at the optimal solution of the relaxed problem (5). Then, we select $r \in \arg\max_{i=1,\dots,n} g_i$, i.e., the index corresponding to the bilinear term where we have the largest error at the optimal solution of the relaxation. Next, we might define the following branching operations for box B:

Branching on x and y: Define four children nodes by adding constraints $\{x_r \leq x_r^\star(B),\ y_r \leq y_r^\star(B)\}$, $\{x_r \leq x_r^\star(B),\ y_r \geq y_r^\star(B)\}$, $\{x_r \geq x_r^\star(B),\ y_r \leq y_r^\star(B)\}$, $\{x_r \geq x_r^\star(B),\ y_r \geq y_r^\star(B)\}$, respectively (quaternary branching);

Branching on x: Define two children nodes by adding constraints $x_r \leq x_r^\star(B)$ and $x_r \geq x_r^\star(B)$, respectively (binary branching);

Branching on y: Define two children nodes by adding constraints $y_r \leq y_r^\star(B)$ and $y_r \geq y_r^\star(B)$, respectively (binary branching).

Note that all choices above, with the new McCormick relaxation given by the new limits on the variables, reduce to zero the error for bilinear term $x_r y_r$ at the optimal solution of problem (5). It is worthwhile to remark that the computed lower bound tends to become exact even when branching is always performed with respect to variables of the same type (say, always variables x_i, $i = 1, \dots, n$). Indeed, it is enough to have that $\|\mathbf{u_x} - \boldsymbol{\ell}_\mathbf{x}\| \to 0$ or, alternatively, that $\|\mathbf{u_y} - \boldsymbol{\ell}_\mathbf{y}\| \to 0$ in order to let the underestimating function values converge to the original objective function values. This is a consequence of the fact that the McCormick underestimation function tends to the value of the corresponding bilinear term even when only one of the two intervals on which it is defined shrinks to a single point. In the computational experiments we tried all three possibilities discussed above and it turns out that the best choice is the binary branching obtained by always branching on y variables.

3.4 Bound-tightening technique

A reduction of the boxes merely based on the above branching strategy would lead to a quite inefficient algorithm. It turns out that performance can be strongly enhanced by an Optimality-Based Bound-Tightening (OBBT in what follows) procedure (see, e.g., [8, 31]). An OBBT procedure receives in input a box B and returns a tightened box in output, removing feasible points which do not allow to improve the current best feasible solution. More formally, let $\mathcal{B}$ be the set of n-dimensional boxes. Then:

$$OBBT\,:\,\mathcal{B} \to \mathcal{B}\,:\,OBBT(B) \subseteq B \quad \text{and} \quad F(\mathbf{x}, \mathbf{y}, v) \\ \geq GUB \quad \forall\,(\mathbf{x}, \mathbf{y}) \in [B \cap P_{\mathbf{x},\mathbf{y}}] \setminus OBBT(B).$$

In our approach, we propose to employ an OBBT procedure, which is expensive but, as we will see, also able to considerably reduce the number of branch-and-bound nodes. The lower and upper limits $\ell_{x_i}, \ell_{y_i}, u_{y_i}, u_{x_i}$, $i = 1, \dots, n$ are refined through the solution of LP problems having the feasible set defined by constraints (5b)-(5g) and the additional constraint

$$\frac{1}{2}\sum_{i=1}^{n} f_i + v \leq GUB, \tag{7}$$

stating that we are only interested at feasible solutions where the underestimating function, i.e., the left-hand side of the constraint, corresponding to the objective function (5a), is not larger than the current upper bound *GUB*. Thus, each call of this OBBT procedure requires the solution of $4n$ LP problems with the following objective functions:

$$\ell_{x_i}/u_{x_i} = \min / \max \ x_i, \quad \ell_{y_i}/u_{y_i} = \min / \max \ y_i, \quad i = 1, \dots, n.$$

Note that all these problems are bounded in view of the fact that $\mathbf{x}$ is constrained to belong to the unit simplex. In fact, what we observed through our computational experiments is that it is not necessary to solve all $4n$ LPs but it is enough to concentrate the effort on the most 'critical' variables. More precisely, in order to reduce the number of LPs without compromising the performance, we employed the following strategies (see also [12] for strategies to reduce the effort). Taking into account the quantities g_i computed in (6), we notice that the larger the g_i value, the higher is the need for a more accurate underestimation of the corresponding bilinear term. Then, we solved the following LP problems.

- $\lceil 0.2n \rceil$ LP problems with objective function min y_i, for all i corresponding to the $\lceil 0.2n \rceil$ largest g_i values;
- a fixed number $\lceil 0.1n \rceil$ of LP problems with objective function max y_i, for all i corresponding to the $\lceil 0.1n \rceil$ largest g_i values;
- again $\lceil 0.1n \rceil$ LP problems with objective function max x_i, for all i corresponding to the $\lceil 0.1n \rceil$ largest g_i values;
- no LP problem with objective function min x_i.

These choices have been driven by some experimental observations. In particular, we noticed that the lower limit for y_i is the most critical for the bound computation or, stated in another way, constraint

$$f_i \geq \ell_{y_i} x_i + \ell_{x_i} y_i - \ell_{x_i} \ell_{y_i},$$

is often the active one. For this reason a larger budget of LP problems is allowed to improve this lower limit with respect to the upper limits. Instead, we never try to improve the lower limit ℓ_{x_i} because it is experimentally observed that this limit can seldom be improved.

This way, the overall number of LPs to be solved at each call of the OBBT procedure is reduced to approximately $0.4n$. Note that rather than solving all LP problems with the same feasible set, we could solve each of them with a different feasible region by incorporating all previously computed new limits in the definition of the feasible region for the next limit to be computed. That leads to sharper bounds, however we excluded this opportunity since we observed that LP solvers strongly benefit from the opportunity of solving problems over the same feasible region.

The underestimating function depends on the lower and upper limits $\ell_{x_i}, \ell_{y_i}, u_{y_i}, u_{x_i}$. Thus, once we have updated all such limits, we can call procedure OBBT again in order to further reduce the limits. These can be iteratively reduced until some stopping condition is fulfilled. Such iterative procedure has been proposed and theoretically investigated, e.g., in [8]. That obviously increases the computational cost per node, since the overall number of LPs to be solved at each node is now approximately $0.4n$ times the number of calls to the procedure OBBT, which depends on the stopping condition. But, again, we observed that the additional computational cost is compensated by a further reduction of the overall number of nodes in the branch-and-bound tree.

It is important to stress at this point that OBBT procedures in general and the one proposed here in particular, are not new in the literature. The main contribution of this work lies in the observation that a very aggressive application of the proposed OBBT, while increasing considerably the computational cost per node, is the real key for an efficient solution of the addressed problem. Indeed, we will see through the experiments, that our approach is able to significantly outperform commercial QP solvers like `CPLEX` and `GUROBI`, and a solver like `BARON`, which is strongly based on tightening techniques.

3.5 Pseudo-code of the branch-and-bound approach

In this section we collect all the previously described tools and present the pseudo-code of the proposed branch-and-bound approach. In Line 1 an initial box B_0 is introduced and the collection of branch-and-bound nodes $\mathcal{C}$ still to be explored is initialized with it. In Line 2 a lower bound over B_0 is computed, while in Line 3 the current best observed feasible point $\mathbf{z}^\star$ and the current GUB value are initialized. Take into account that such values can also be initialized after running a few local searches from randomly generated starting points. Lines 4–21 contain the main loop of the algorithm. Until the set of nodes still to be explored is not empty, the following operations are performed. In Line 5 one node in $\mathcal{C}$ with the lowest lower bound is selected. In Line 6 the index k of the branching variable is selected as the one with the largest gap g_i as defined in (6). In Line 7 the branching operation is performed. In Line 8 the selected node $\bar{B}$ is removed from $\mathcal{C}$, while in Lines 9–19 the following operations are performed for each of its child nodes. In the loop at Lines 10–17, first procedure OBBT is applied and then the lower bound over the tightened region is computed, until a stopping condition is satisfied. In particular, in our experiments we iterate until the difference between the (non-decreasing) lower bounds at two consecutive iterations fall below a given threshold ϵ ($\epsilon = 10^{-3}$ in our experiments). In Lines 13–16 both $\mathbf{z}^\star$ and GUB are possibly updated through the optimal solution of the relaxed problem. In Line 18 we add the child node to $\mathcal{C}$. Finally, in Line 20 we remove from $\mathcal{C}$ all nodes with a lower bound not lower than $(1 - \varepsilon)GUB$, where ε is a given tolerance value. In all the experiments, we fixed a relative tolerance $\varepsilon = 10^{-3}$, which is considered adequate for practical applications.

Note that we do not discuss convergence of the proposed branch-and-bound approach, since it easily follows by rather standard and general arguments which can be found in [15].

In Algorithm 1 we highlighted with a frame box, both the stopping condition in Line 10 and the call to the OBBT procedure at Line 11, since the performance of the proposed algorithm mainly depends on how these two lines are implemented.

In what follows, in order to stress the importance of bound-tightening, we will refer to the proposed approach as Branch-and-Tight (B&T), which belongs to the class of Branch-and-Cut approaches, since tightening the bound of a variable is a special case of cutting plane.

```
   Data   : $\mathbf{S} \in \mathbb{R}^{n\times n}$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\varepsilon > 0$;
1  Let $B_0$ be an initial box and set $\mathcal{C} = \{B_0\}$ ;
2  Compute the lower bound $L(B_0)$ through (5) ;
3  $\mathbf{z}^\star = \mathbf{x}^\star(B_0)$ and $GUB = F(\mathbf{x}^\star(B_0), \mathbf{y}^\star(B_0), v^\star(B_0))$ ;
4  while $\mathcal{C} \neq \emptyset$ do
5      $\bar{B} \in \arg\min_{B\in\mathcal{C}} L(B)$ ;
6      $k \in \arg\max_{i=1,\dots,n} x_i^\star(\bar{B})y_i^\star(\bar{B}) - f_i^\star(\bar{B})$ ;
7      Branch $\bar{B}$ into $B_1 = \bar{B} \cap \{y_k \leq y_k^\star(\bar{B})\}$ and $B_2 = \bar{B} \cap \{y_k \geq y_k^\star(\bar{B})\}$ ;
8      $\mathcal{C} = \mathcal{C} \setminus \{\bar{B}\}$ ;
9      for $i \in \{1, 2\}$ do
10         while $\boxed{\textit{A stopping condition is not satisfied}}$ do
11             $\boxed{B_i = OBBT(B_i)}$ ;
12             Compute the lower bound $L(B_i)$ through (5) ;
13             if $F(\mathbf{x}^\star(B_i), \mathbf{y}^\star(B_i), v^\star(B_i)) < GUB$ then
14                 $\mathbf{z}^\star = \mathbf{x}^\star(B_i)$ ;
15                 $GUB = F(\mathbf{x}^\star(B_i), \mathbf{y}^\star(B_i), v^\star(B_i))$ ;
16             end
17         end
18         $\mathcal{C} = \mathcal{C} \cup \{B_i\}$ ;
19     end
20     $\mathcal{C} = \mathcal{C} \setminus \{B \in \mathcal{C} \ : \ L(B) \geq (1-\varepsilon)GUB\}$ ;
21 end
22 return $\mathbf{z}^\star$, $GUB$;
```

Algorithm 1: Branch-and-bound algorithm

4 Example: Crime prevention by patrolling (of the Police)

In this section we present an example of real-life instance where player 1 is the police, patrolling around in a certain geographic area. In this area, we consider the roads as defining a directed graph that the police traverses seeking to prevent crimes. They do so by randomly checking locations in the area, reachable over the roads, and traveling from one location to the other induces a certain travel time. For the crime distribution, we assume that "more public space means more witnesses, so less likelihood of a crime committed". Applying this intuition to a roadmap, we take the probability of crimes to occur at a location as inverse proportional to the number of ways leading to this point. That is, the "more remote" a place is, the more likely is a crime to happen there, and places that are reachable over many ways are more likely to be crowded, so crimes are less likely there (of course, not all sorts of crimes, such as pickpockets would certainly prefer crowded places, but mugging is preferably done where a victim cannot easily escape). Note, however, the probability can also be defined in different ways, e.g., by taking into account police records. Our showcase region is the Hampi village in India, shown as a screenshot map in Figure ??. The respective geographic information ships with the package `dodgr` (distances on directed graphs) [19] for the R environment, so that we can conveniently use geographic information and digitalized roadmaps to define the playground for the game model (see [20] for how this is done step-by-step). In our case, the `dodgr` package directly lets us compute travel times or distances, representing the costs incurred to realize a random spot checking, which is precisely the matrix **S** above. The actual payoff structure in the game depends on the locations and their likelihoods to become crime scenes. The road network for the Hampi village, has 1901 vertices and 3891 edges. At this point a subset of vertices of the road network is selected. The resulting vertices in the graph are then defined as pure strategies in the game, i.e., places for the police to be checked, with inter-vertex travel times taken as the switching cost in the matrix **S**. For practical purposes, it is reasonable to take a subset with cardinality between 50 and 100. Indeed, this size also appears reasonable in light of the fact that police patrols can certainly not check arbitrarily many places in reasonable time and efficiency, thus problem instances between 50 and 100 places appear as the most that is physically feasible.

Note that games of a shape like that of our police patrolling example exist in manifold versions in the literature, such as regarding the patrolling of coast guards [10], border patrol [21], pipeline protection [3], airport surveillance [22], and game theoretic models against environmental crime (so-called green security games) [5]; see [28] for a more extensive overview of related game models. Common to all these is their goal of optimizing patrolling (and to an extent also surveillance), but none of this past work accounts for the efforts of realizing the patrolling in practice, in light of the efforts practically invested (for the moving), but not theoretically counted in the optimization model.

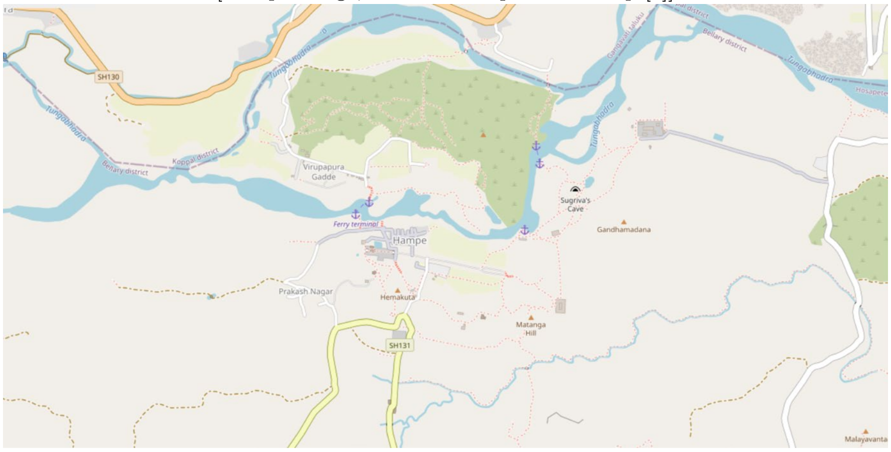
5 Numerical results

In this section we will first describe the test instance generator and, then, we will present and discuss the results of extensive computational experiments with such instances.

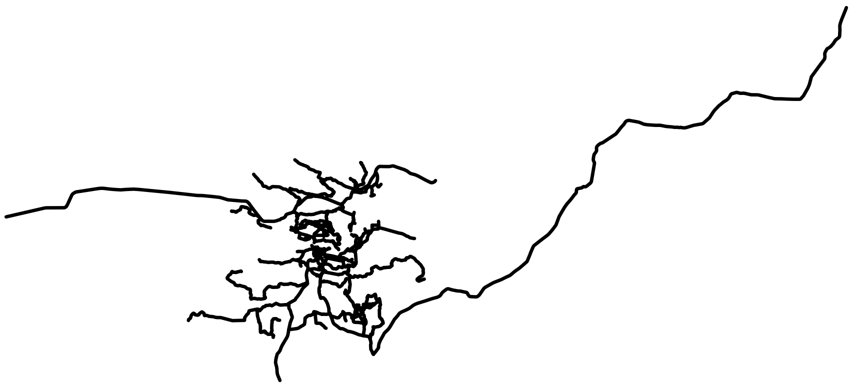
5.1 Test instances description

The game is about spot checking a set of n places to guard them against an adversary. The places are spatially scattered, with a directed weighted graph describing the connections (direct reachability) of place v from place u by an edge $v \to u$ with a random length.

[Hampi village, taken from OpenWorldMap [1]]



[Road graph as extracted from the geographic data provided by [19]]



The payoffs in the game are given by an $n \times n$ matrix $\mathbf{A}$ (so $m = n$ in the above description), and are interpreted as the loss that the defending player 1 suffers when checking place i while the attacker is at place j. Thus, the defender can:

– either miss the attacker ($i \neq j$) in which case there will be a Weibull-distributed random loss with shape parameter 5 and scale parameter 10.63 (so that the variance is 5); (this distribution is a common choice to describe economic losses, among other extreme value distributions);
– or hit the attacker at $i = j$, in which case there is zero loss.

The defender is thus minimizing, and the attacker is maximizing. The problem above is that of the defender. The Nash equilibrium then gives the optimal random choice of spot checks to minimize the average loss. To avoid trivialities, the payoff matrices are constructed not to admit pure strategy equilibria, so that the optimum (without switching cost) is necessarily a mixed strategy.

As for the switching cost, if the defender is currently at position i and next – according to the optimal random choice – needs to check the (non-adjacent) place j, then the cost for the switch from i to j is the shortest path in the aforementioned graph (note that, since the graph is directed, the matrix $\mathbf{S}$ is generally nonsymmetric).

For the random instances, the matrix $\mathbf{S}$ is thus obtained from a (conventional) all-shortest path algorithm applied to the graph. Note that the graph is an Erdös-Renyi type graph with n nodes and $p = 0.3$ chance of any two nodes having a connection. We also made tests with sparser ($p = 0.2$) and denser ($p = 0.4$) graphs, which result in $\mathbf{S}$ matrices with larger entries in the former case and smaller in the latter case (the entries are given by shortest paths which tend to be smaller in denser graphs). The computational results with these different p values (not reported here) show that the instances with $p = 0.2$ are slightly more challenging than those with $p = 0.3$, while those with $p = 0.4$ are slightly simpler. However, the differences are not very significant. Also note that all entries of matrix $\mathbf{S}$ are positive except for the diagonal ones, so that the matrix is dense. We underline this fact since, according to the experimental results reported, e.g., in [7, 33], the density of the Hessian matrix is a relevant factor to assess the difficulty of non-convex QP problems.

Remark 3 The Erdös-Renyi model is here a suitable description of patrolling situations in areas where moving from any point to any other point is possible without significant physical obstacles in between. Examples are water areas (e.g., coasts) or natural habitats (woods, ...), in which guards are patrolling. It goes without saying that implementing the physical circumstances into the patrolling problem amounts to either a particular fixed graph topology or class of graphs (e.g., trees as models for harbor areas, or general scale-free networks describing communication relations). Such constrained topologies, will generally induce likewise constrained and hence different (smaller) strategy spaces, but leave the problem structure as such unchanged, except for the values involved.

The weights in the graph were chosen exponentially distributed with rate parameter $\lambda = 0.2$, and the Weibull distribution for the losses has a shape parameter 5 and scale parameter ~ 10.63, so that both distributions have the same variance of 5.

Remark 4 The choice of the Weibull distribution is because of its heavy tails, useful to model extreme events (in actuarial science, where it appears as a special case of the generalized extreme value distribution). If the graph is an attack graph, we can think of possibly large losses that accumulate as the adversary traverses an attack path therein (but not necessarily stochastically independent, which the Weibull-distribution sort of captures due to its memory property). Besides, both the exponential and the Weibull distribution only take non-negative values, and thus lend themselves to a meaningful assignment of weights as "distances" in a graph.

The graph sizes considered are $n = 50, 75, 100$ and for each graph size we consider ten random instances. We restricted the attention to α values in $\{0.3, 0.4, \dots, 0.9\}$ since problems with α values smaller than 0.3 and larger than 0.9 turned out to be simple ones. The overall number of instances is, thus, 210 (70 for each size $n = 50, 75, 100$).

Note that all the data of the test instances are available at the web site http://www.iasi.cnr.it/~liuzzi/StQP.

5.2 Description of the experiments

The problem discussed in this paper belongs to the class of nonconvex QP problems with linear constraints, which is a quite active research area. Even well-known commercial solvers, like `CPLEX` and `GUROBI`, have recently included the opportunity of solving these problems. In [33] different solution approaches have been compared over different nonconvex QP problems, namely: Standard Quadratic Programming problems (StQP), where the feasible region is the unit simplex; BoxQP, where the feasible region is a box; and general QPs, where the feasible region is a general polytope (in [7] an extensive comparison has also been performed more focused on BoxQPs). The approaches tested in [33] have been the nonconvex QP solver of `CPLEX`, `quadprogBB` (see [9]), `BARON` (see [29]), and `quadprogIP`, introduced in [33]. According to the computational results reported in that work, solvers `quadprogIP` and `quadprogBB` have quite good performance on some subclasses. More precisely, `quadprogIP` works well on the StQP problem (see also [13] for another approach working well on this subclass), while `quadprogBB` performs quite well on BoxQPs, especially when the Hessian matrix of the objective function is dense. However, they do not perform very well on QP problems with general linear constraints. Some experiments we performed show that their performance is not good also on the QP subclass discussed in this paper. For this reason we do not include their results in our comparison. Thus, in the comparison we included: the nonconvex QP solver of `CPLEX` (v. 12.10), the best performing over QPs with general linear constraints according to what is reported in [33]; the nonconvex QP solver of `GUROBI` (v. 9.0.0), which has been recently introduced and is not tested in that paper; `BARON` (v. 2019.12.7), since bound-tightening, which, as we will see, is the most relevant operation in the proposed approach, also plays a central role in that solver.

We performed four different sets of experiments:

- Experiments to compare our approach B&T with the above mentioned existing solvers over the subclass of QP problems discussed in this paper (only at dimension $n = 50$, which, as we will see, is already challenging for all the competitors);
- Experiments with B&T by varying the intensity of bound-tightening (no bound-tightening, light bound-tightening, strong bound-tightening), in order to put in evidence that (strong) bound-tightening is the key operation in our approach;

- Experiments with B&T at dimensions $n = 50, 75, 100$, in order to see how it scales as the dimension increases;
- Experiments with the Crime Prevention instance described in Sect. 4.

All the experiments have been carried out on an Intel® Xeon® gold 6136 CPU at 3GHz with 48 cores and 256GB main memory. The algorithm has been coded using the Julia [6] language (version 1.3.1). Doing the implementation we parallelized as much as possible the bound-tightening procedure discussed in Sect. 3.4, where many LPs with the same feasible region need to be solved. The code is available for download at the URL http://www.iasi.cnr.it/~liuzzi/StQP.

5.2.1 Comparison with the existing literature

As a first experiment, we compare our method with the commercial solvers `BARON`, `CPLEX` and `Gurobi`. We run all these methods over ten instances at dimension $n = 50$ with $\alpha \in \{0.3, 0.4, \dots, 0.9\}$ (thus, overall, 70 instances). We set a time limit of 600 seconds. A relative tolerance $\varepsilon = 10^{-3}$ is required for all solvers since, as already previously commented, it is considered adequate for this application. In Table 1, we report the average performance. For each method we report the number of nodes (column `nn`), the percentage gap after the time limit and in brackets the computational time needed to reach it (column `GAP % (s)`), and finally the percentage of success, i.e. the percentage of instances solved to optimality within the time limit (column `Succ %`). In our opinion, this table reports the most important finding of this paper. It can be seen that all the commercial solvers fail on most of the instances (apart from 7 out of 10 instances with $\alpha = 0.9$ and, for what concerns `CPLEX`, an instance with $\alpha = 0.8$), whereas B&T solves all the instances with an average time of less than 30 seconds (the complete results are reported in Appendix 1). These results show that, although commercial solvers are fully developed, there is still room for improvements. In particular, it seems that performing bound-tightening in a very intensive way can strongly enhance the performance of a solution approach. In fact, as previously recalled, `BARON` already incorporates bound-tightening techniques but, as we will see in the following set of experiments, the intensity with which bound-tightening is applied also makes a considerable difference. Before that, however, for the sake of completeness, we report in Table 2, the results of B&T over the $n = 50$ test instances when a lower tolerance value is employed, namely $\varepsilon = 10^{-5}$, in order to evaluate the impact of this parameter on the performance of B&T. It is interesting to notice that the lower value has a clear impact on the instances with small α value (say $\alpha \le 0.5$), even causing a failure with $\alpha = 0.4$, while the impact is much milder for larger α values. In this table we also include column `time/LP`, where we report the average time for the solution of each LP. Such value is almost equal to the overall CPU time divided by the number of LPs solved. This is due to the fact that the computing times of B&T are almost entirely due to the OBBT procedure and, in particular, to the LPs needed to apply it. An analogous consideration applies to all the experiments we reported in this paper.

Table 1 Average performance of all the solvers on ten instances for each value of α when $n = 50$. The column *nn* represents the average number of nodes, the column *GAP%(s)* reports the percentage GAP after at most 600 seconds and in brackets the average CPU time in seconds. The column *S%* represents the percentage of success among the ten instances

α	BARON			CPLEX			GUROBI			B&T		
	nn	GAP%(s)	S%	nn	GAP%(s)	S%	nn	GAP%(s)	S%	nn	GAP%(s)	S%
0.3	306.5	4.60 (600.0)	0	250086.7	1.50 (613.2)	0	1122759.8	3.04 (600.0)	0	11.2	0.0 (6.7)	100
0.4	326.4	6.88 (600.0)	0	235696.6	2.73 (613.8)	0	1031329	4.87 (600.0)	0	70	0.0 (32.6)	100
0.5	362	9.59 (600.0)	0	232083.7	4.37 (613.7)	0	1036844.2	6.99 (600.0)	0	80.6	0.0 (32.7)	100
0.6	358.1	12.81 (600.0)	0	232240.6	7.34 (616.6)	0	1179379.3	9.48 (600.0)	0	45.8	0.0 (18.5)	100
0.7	332.1	16.23 (600.0)	0	236429.9	9.57 (617.0)	0	1351356.6	11.99 (600.0)	0	57.6	0.0 (18.0)	100
0.8	252.1	19.37 (600.0)	0	243393.8	10.14 (599.1)	10	1657120.1	11.66 (600.0)	0	54.6	0.0 (11.2)	100
0.9	40.5	4.58 (411.8)	70	197096.8	2.62 (426.6)	70	978047.8	0.70 (237.2)	70	61.6	0.0 (7.6)	100

Table 2 Average performance of B&T on all the 10 instances for each value of α when $n = 50$ with tolerance equal to 10^{-5} with strong bound tightening

α	nn	#LPs	time	time/LP	GAP%	Succ %
0.3	91.2	57095	66.73	$1.16e^{-3}$	0.00	100
0.4	391.8	205684.1	231.21	$1.11e^{-3}$	$2.2e^{-4}$	90
0.5	230.2	95043.3	103.19	$1.08e^{-3}$	0.00	100
0.6	70.4	26367.3	28.81	$1.09e^{-3}$	0.00	100
0.7	90.2	22009.1	23.35	$1.04e^{-3}$	0.00	100
0.8	59	10500.8	10.61	$1.00e^{-3}$	0.00	100
0.9	58.8	6260.3	5.95	$9.41e^{-4}$	0.00	100

5.2.2 Importance of bound-tightening

As already stressed many times, the quite good performance of B&T is due to the bound-tightening procedure. It is now time to show it with numbers. To this end, besides the already proposed setting for our approach, we ran it under two different settings:

No bound-tightening at each node we do not apply the procedure *OBBT*, but we simply compute the lower bound by solving problem (5);
Light Bound-Tightening] at each node we only solve the following LPs once (and, consequently, we compute the solution of problem (5) only once)

- $\lceil 0.1n \rceil$ LP problems with objective function min y_i, for all i corresponding to the $\lceil 0.1n \rceil$ largest g_i values;
- a fixed number $\lceil 0.05n \rceil$ of LP problems with objective function max y_i, for all i corresponding to the $\lceil 0.05n \rceil$ largest g_i values;
- again $\lceil 0.05n \rceil$ LP problems with objective function max x_i, for all i corresponding to the $\lceil 0.05n \rceil$ largest g_i values;
- no LP problem with objective function min x_i.

Of course, this strongly reduces the effort per node. With no bound-tightening a single LP is solved per node, while with light bound-tightening $\lceil 0.2n \rceil + 1$ LPs are solved at each node. In fact, light bound-tightening already requires a considerable computational effort per node (and, as we will see, it is already enough to perform better than existing solvers). However, the originally proposed strong bound-tightening procedure, where the OBBT procedure is iteratively applied and at each iteration $\lceil 0.4n \rceil$ LPs are solved, delivers better results. In Table 3, we report the average performance on the instances with $n = 50$ in terms of number of nodes, number of LPs solved, CPU time in seconds and percentage gap of the three levels of bound-tightening. It is evident from the table that the OBBT procedure is what really makes the difference: most of the instances are not solved without bound-tightening, whereas the number of nodes and the CPU time needed to solve the instances decrease as we increase the level of bound-tightening. In Appendix 1 we also report the full table with all the results.

Table 3 Average performance on all the 10 instances for each value of α when $n = 50$ for the different levels of bound tightening

	No Bound Tightening				Light Bound Tightening				Strong Bound Tightening			
α	nn	#LPs	Time	GAP%	nn	#LPs	Time	GAP%	nn	#LPs	Time	GAP%
0.3	106934.8	106937.8	600.00	1.899	423	19252.7	26.24	0.00	11.2	4665	6.7	0.00
0.4	111981.2	111984.2	600.00	2.93	2883.4	129670.2	169.431	$1.3e^{-2}$	70	27077.5	32.58	0.00
0.5	117565.6	117568.6	600.00	4.19	1949.8	83921.9	108.489	0.00	80.6	26668.6	32.661	0.00
0.6	117942.4	117945.4	600.00	4.932	1106.8	45581.3	57.73	0.00	45.8	14882.5	18.46	0.00
0.7	113859.4	113862.4	600.00	6.601	813.8	30621.5	39.447	0.00	57.6	14604.9	17.973	0.00
0.8	90646.8	90649.8	600.00	6.77	307	10600.6	13.23	0.00	54.6	9247	11.199	0.00
0.9	89502.6	89505.6	531.13	1.48	183.2	5810.2	7.43	0.00	61.6	6477.2	7.56	0.00

5.2.3 Computational results over the proposed test instances as *n* increases

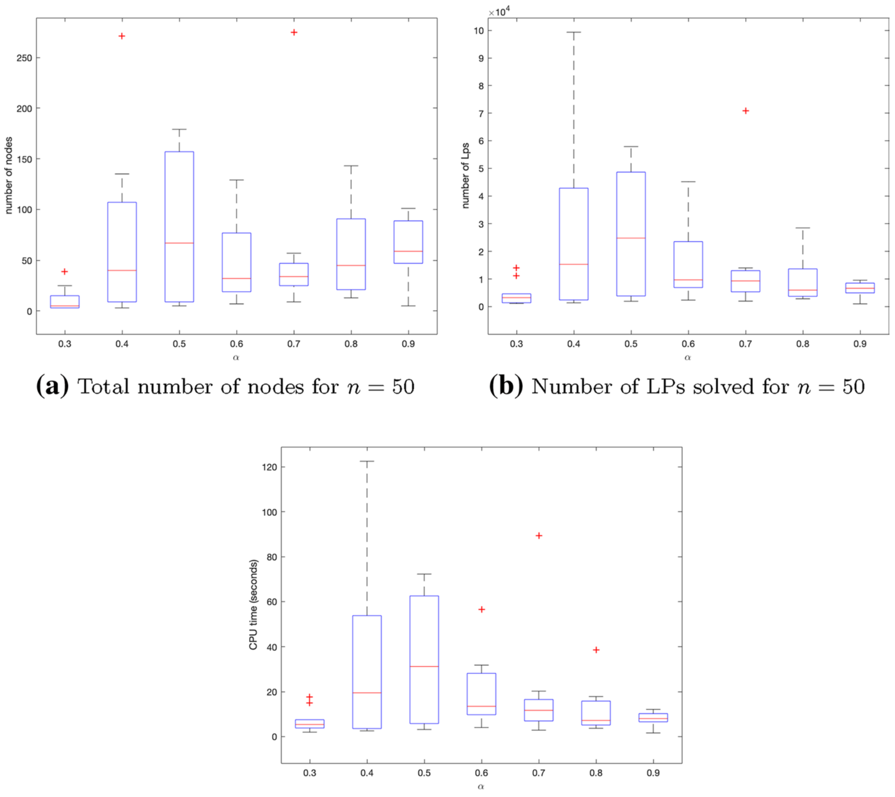
In this subsection we show the behavior of B&T as the dimension n increases. We have solved ten instances for three different sizes $n = 50$, 75, 100 and the usual values of $\alpha = \{0.3, 0.4, \dots, 0.9\}$ (thus, overall 210 instances). Note that, for all the different values of n, lower and larger values of α with respect to those we tested give rise to much simpler instances (recall that the problem becomes polynomially solvable for the extreme values $\alpha = 0$ and $\alpha = 1$). We set a time limit of 10800s for all instances. For $n = 50$ and $n = 75$ we solve all the instances to optimality (in fact, the largest time to solve an instance with $n = 50$ is about 2 minutes, whereas for $n = 75$ the largest time is below 1 hour, but most of the problems are solved within 10 minutes). In Figures 2a–c we report the box plot of number of nodes, number of LPs and CPU time needed for the different values of α when $n = 50$. The figure shows that the hardest instances are the ones corresponding to the central values of α and this will turn out to hold true also at larger dimensions. We also observe that the overall number of nodes is extremely limited, thus confirming that, while computationally expensive, the bound-tightening procedure allows to considerably reduce the size of the branch-and-bound tree (again, this fact is observed also at larger dimensions).

In Fig. 3, we report the different box plots for all the instances at $n = 75$. It is worthwhile to remark that we solve most of them within ten minutes and exploring less than 300 nodes.

Finally, in Fig. 4 we report the performance of B&T on problems of dimension $n = 100$. In this case there are seven instances we are not able to solve within the time limit. These occur for values $\alpha \in \{0.6, 0.7, 0.8\}$, thus confirming that the central values of this parameter give rise to the most challenging instances. With respect to $n = 50$ and $n = 75$, we have the additional box plot displayed in Fig. 4d reporting the final percentage gap when the time limit is reached. Note that it is never larger than 1.2% and most of the times it is lower than 0.5%, thus showing that, even when the algorithm does not terminate, the quality of the returned solution is guaranteed to be high.

5.2.4 Numerical results for the Crime Prevention application

In this subsection we solve an instance of the Crime Prevention application, described in Sect. 4, where 50 check spots have been identified (i.e., $n = 50$ according to our notation). In Table 4, we report the results for different values of α obtained by B&T and by the commercial solvers CPLEX, Gurobi and BARON. We report for each method the CPU time (we set a time limit of 600 seconds) followed in brackets by the gap at the end of the time, the number of nodes, and for our method also the number of LPs solved, since it is our major computational burden. Although this instance turns out to be less challenging than the random ones previously considered, and also the other solvers are able to return the solution within the time limit for most (but not all) α values, B&T is the only method able to solve the instance for all the values of α and in all cases within 3 seconds. Therefore, also for this real world application, our method outperforms



(a) Total number of nodes for $n = 50$

(b) Number of LPs solved for $n = 50$

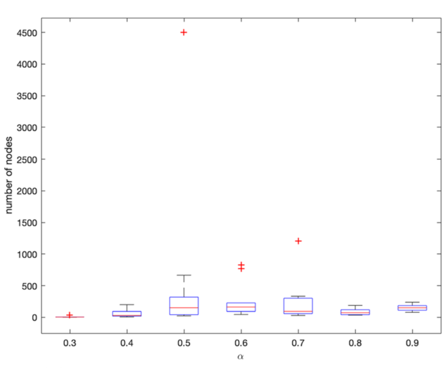
(c) CPU times in seconds

Fig. 2 Box plots for different performance measures for $n = 50$

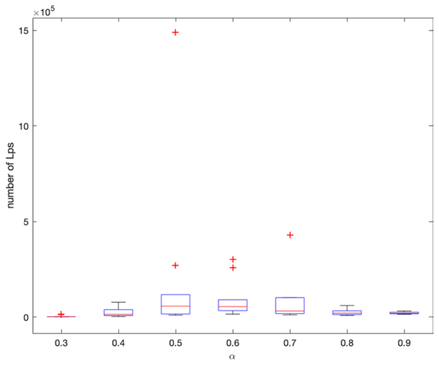
the commercial solvers, confirming the effectiveness of the approach. We also run B&T with a larger number of check spots (namely, 75 e 100) and the usual different α values. The method is able to solve them all within 11 s, thus confirming once again the efficiency of B&T.

6 Conclusions and future work

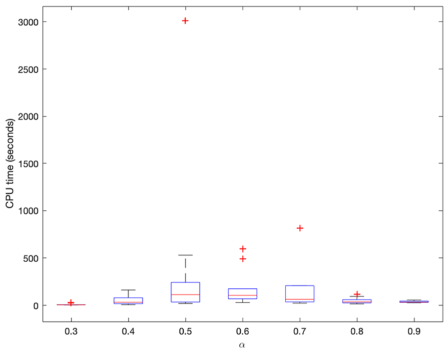
In this paper we addressed some game theory problems arising in the context of network security. In these problems there is an additional quadratic term, representing *switching costs*, i.e.. the costs for the defender of switching from a given strategy to another one at successive rounds of the game. The resulting problems can be reformulated as nonconvex QP with linear constraints. Test instances of these problems turned out to be very challenging for existing solvers, and we propose to extend with them the current benchmark set of test instances for QP problems. We presented a spatial branch-and-bound approach to tackle these problems and we have shown that a rather



(a) Total number of nodes for $n = 75$



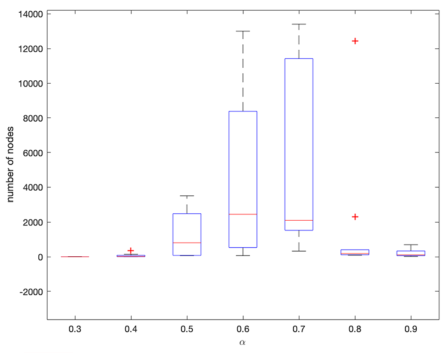
(b) Number of LPs solved for $n = 75$



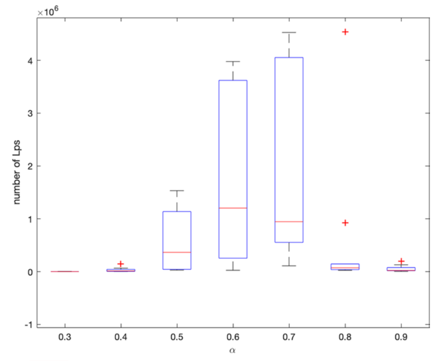
(c) CPU times in seconds

Fig. 3 Box plots for different performance measures for $n = 75$

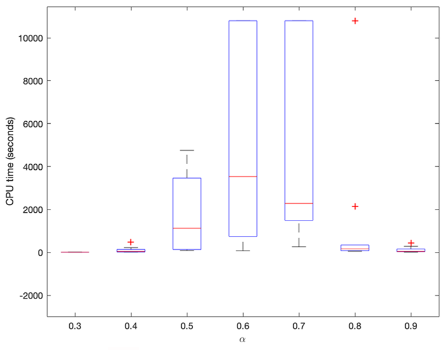
aggressive application of an OBBT procedure is the key for their efficient solution. The procedure is expensive, since it requires multiple solutions of LP problems at each node of the branch-and-bound tree. But we empirically observed that the high computational costs per node of the branch-and-bound tree are largely compensated by the low number of nodes to be explored. We recall that the code of B&T and all the data of the test instances are available at the web site http://www.iasi.cnr.it/~liuzzi/StQP. As a topic for future research, we would like to further investigate the use of OBBT procedures in the solution of QP problems, and we would like to identify other cases, besides those addressed in this paper, where their intensive application may considerably enhance the performance of branch-and-bound approaches. We have actually already performed some experiments and obtained promising results over test instances of nonconvex QPs with general linear constraints. We plan to present the results in a future work. Moreover, it would be interesting to evaluate whether the intensive application of OBBT procedures is also able to enhance the performance of commercial nonconvex QP solvers.



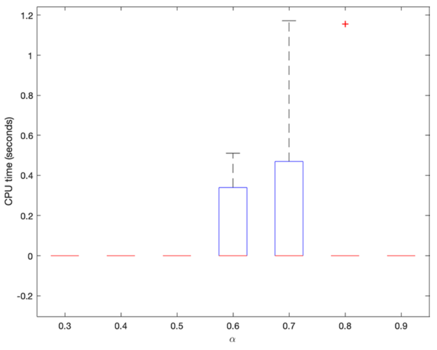
(a) Total number of nodes for $n = 100$



(b) Number of LPs solved for $n = 100$



(c) CPU times in seconds



(d) % Gap at the time limit

Fig. 4 Box plots for different performance measures for $n = 100$

Table 4 Results on the instance of the Crime Prevention application with 50 check spots

α	GUROBI		CPLEX		BARON		B&T		
α	CPU time	#nodes	CPU time	#nodes	CPU time	#nodes	CPU time	#nodes	#LPs
0.3	600.00(17.05)	3604392	600.00(30.28)	373403	600.00(2.37)	117	2.48(0)	19	2485
0.4	175.26(0)	1186711	600.00(26.41)	415455	600.00(1.88)	78	2.30(0)	19	2141
0.5	31.39(0)	201557	222.83(0)	244784	600.00(1.42)	85	2.21(0)	21	2053
0.6	6.22(0)	26632	32.84(0)	34536	207.26(0)	43	2.27(0)	23	2239
0.7	2.28(0)	5711	16.38(0)	20414	66.74(0)	31	1.93(0)	23	1705
0.8	1.39(0)	2812	5.1(0)	3247	15.05(0)	6	1.22(0)	3	631
0.9	1.17(0)	746	2.82(0)	701	3.87(0)	3	1.05(0)	1	362

A Proof of Theorem 1

In this section we consider the complexity of problem (2). Such problem is a nonconvex QP with linear constraints. NP-hardness of QP problems has been established for different special cases like, e.g., the already mentioned StQP problems (see [18]) and the Box QP problems (see, e.g., [23]). However, due to its special structure, none of the known complexity results can be applied to establish the NP-hardness of problem (2). Thus, in what follows we formally prove that its corresponding decision problem is NP-complete. Let

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{S}\mathbf{x} + \max_{k=1,\dots,m} \mathbf{A}_k^T\mathbf{x}, \tag{8}$$

where $S_{ii} = 0$ for each $i = 1, \dots, n$ and $S_{ij} \geq 0$ for each $i \neq j$, while $\mathbf{A}_k \geq \mathbf{0}$, $k = 1, \dots, m$. Moreover, let

$$\Delta_n = \{\mathbf{x} \in \mathbb{R}^n_+ \ : \ \mathbf{e}^T\mathbf{x} = 1\},$$

be the n-dimensional unit simplex. After incorporating α and $(1-\alpha)$ respectively into $\mathbf{S}$ and $\mathbf{A}_k$, $k = 1, \dots, m$, problem (2) is equivalent to $\min_{\mathbf{x}\in\Delta_n} f(\mathbf{x})$. Then, we would like to establish the complexity of the following decision problem:

$$\text{Given a constant } \xi \geq 0 \ \ \exists\, \mathbf{x} \in \Delta_n \ : \ f(\mathbf{x}) \leq \xi? \tag{9}$$

We prove the result by providing a polynomial transformation of the max clique decision problem: Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, does there exist a clique C in G with cardinality at least k? We define the following instance of the decision problem (9). Let

$$S_{ij} = \begin{cases} 0 & \text{if } i = j \text{ or } (i,j) \in E \\ n^4 & \text{otherwise.} \end{cases}$$

Moreover, let $m = n$ and for each $k = 1, \dots, n$ let $\mathbf{A}_k = \mathbf{e}_k$, where $\mathbf{e}_k$ is the vector with all components equal to 0, except the k-th one, which is equal to 1. Stated in another way, the piece-wise linear part is $\max_{k=1,\dots,n} x_k$. Finally, let $\xi = \frac{1}{k}$. We claim that the minimum value of f over Δ_n is not larger than $\xi = \frac{1}{k}$ if and only if G contains a clique with cardinality k. The *if* part is very simple. Indeed, let us consider the feasible solution $x_i = \frac{1}{k}$ if $i \in C$, where C is a clique of cardinality k, and let $x_i = 0$ otherwise. Then, the value of f at this point is equal to $\frac{1}{k}$. Indeed , the value of the quadratic part is 0, while the value of the piece-wise linear part is $\frac{1}{k}$. The proof of the *only if* part is a bit more complicated. We would like to prove that, in case no clique with cardinality at least k exists, then the minimum value of f over the unit simplex is larger than $\frac{1}{k}$. Let us denote by $\mathbf{x}^*$ the minimum of f over the unit simplex. Let

$$K = supp(\mathbf{x}^*) = \{i \ : \ x_i^* > 0\},$$

and let C be the maximum clique over the sub-graph induced by K, whose cardinality is at most $k-1$. We first remark that if, for some $i \in K \setminus C$, it holds that

$$x_i^* \geq \frac{1}{n^2} \quad \text{and} \quad \sum_{j \in C \ : \ (i,j) \notin E} x_j^* \geq \frac{1}{n^2},$$

then the quadratic part contains the term

$$n^4 x_i^* \left(\sum_{j \in C \ : \ (i,j) \notin E} x_j^* \right) \geq 1,$$

which concludes the proof. Therefore, for $i \in K \setminus C$ we assume that either $x_i^* < \frac{1}{n^2}$ or

$$\sum_{j \in C \ : \ (i,j) \notin E} x_j^* < \frac{1}{n^2}. \tag{10}$$

Now, let

$$K_1 = \left\{ i \ : \ i \in K \setminus C \ \text{ and } \ x_i^* \geq \frac{1}{n^2} \right\}.$$

If $\exists\, k_1, k_2 \in K_1$ and $(k_1, k_2) \notin E$, then $n^4 x_{k_1}^* x_{k_2}^* \geq 1$, which concludes the proof. Then, we assume that for each $k_1, k_2 \in K_1$, $(k_1, k_2) \in E$, i.e., K_1 itself is a clique. Now let us consider the following subset of C

$$C_1 = \left\{ i \in C \ : \ (i,k) \notin E \ \text{ for at least one } k \in K_1 \right\}.$$

It must hold that $|C_1| \geq |K_1|$. Indeed, if $|C_1| < |K_1|$, then $(C \setminus C_1) \cup K_1$ is also a clique with cardinality larger than C, which is not possible in view of the fact that C has maximum cardinality. Then, in view of (10) we have that

$$x_i^* < \frac{1}{n^2} \quad \forall\, i \in C_1,$$

and, moreover, by definition of K_1, we also have

$$x_i^* < \frac{1}{n^2} \quad \forall\, i \in K \setminus (K_1 \cup C).$$

Since $|C_1| \geq |K_1|$, we have that

$$T = \left\{ i \in K \ : \ x_i^* \geq \frac{1}{n^2} \right\},$$

is such that $|T| \leq |K_1| + |C \setminus C_1| \leq |C_1| + |C \setminus C_1| = |C| \leq k-1$. Thus, taking into account that

$$\sum_{i \in K \setminus T} x_i^* < \frac{1}{n},$$

we must have that

$$\sum_{i \in T} x_i^* > 1 - \frac{1}{n},$$

and, consequently, taking into account that $|T| \leq k-1$, for at least one index $j \in T$ it must hold that

$$x_j^* > \frac{1 - \frac{1}{n}}{k-1} \geq \frac{1}{k},$$

so that the piece-wise linear part of f is larger than $\frac{1}{k}$, which concludes the proof.

B Detailed numerical results

See Tables 5, 6, and 7

Table 5 Complete results for the solvers BARON, CPLEX, GUROBI, and B&T over random instances of problems with dimension $n = 50$ and α ranging between 0.3 and 0.9

α	BARON			CPLEX			GUROBI			B&T		
	Nodes	Time	GAP %	Nodes	Time	GAP%	Nodes	Time	GAP%	Nodes	Time	GAP%
0.3	341	600.00	4.20	259491	614.92	1.41	1068069	600.01	2.71	3	7.54	0.00
0.3	287	600.00	3.19	304348	615.16	0.94	1461048	600.02	1.90	5	3.65	0.00
0.3	373	600.00	5.99	181699	610.30	2.51	807648	600.01	4.31	3	2.23	0.00
0.3	460	600.00	5.02	210663	611.36	1.92	1086852	600.01	3.46	25	14.24	0.00
0.3	290	600.01	5.38	265178	615.30	1.43	1290539	600.02	3.53	39	16.67	0.00
0.3	262	600.00	5.32	263640	613.71	1.52	1184896	600.01	3.42	15	6.15	0.00
0.3	297	600.00	4.96	197876	611.30	2.65	845896	600.01	3.50	9	5.65	0.00
0.3	252	600.00	4.34	198566	610.53	1.69	941140	600.01	2.84	5	4.37	0.00
0.3	269	600.00	3.53	300960	612.69	0.38	1290251	600.02	2.14	5	4.67	0.00
0.3	234	600.00	4.11	318446	616.40	0.60	1251259	600.02	2.55	3	1.84	0.00
0.4	321	600.00	6.43	256323	616.10	2.07	1017803	600.02	4.22	3	2.49	0.00
0.4	343	600.00	5.11	251678	614.52	1.81	1272585	600.03	3.55	11	5.79	0.00
0.4	442	600.00	8.79	148880	608.94	5.63	784327	600.04	6.84	19	8.79	0.00
0.4	323	600.00	8.02	199177	612.36	3.97	947567	600.03	5.90	135	73.05	0.00
0.4	300	600.00	7.20	293459	617.71	1.33	1134036	600.02	4.79	271	115.94	0.00
0.4	437	600.00	7.67	223940	614.49	3.18	948556	600.01	5.51	9	3.58	0.00
0.4	284	600.01	7.78	163925	609.75	4.66	1050362	600.01	5.93	79	33.54	0.00
0.4	312	600.01	6.21	190619	610.96	3.13	822111	600.02	4.76	107	51.43	0.00
0.4	300	600.00	5.06	327483	615.01	0.43	1201337	600.01	3.11	61	28.38	0.00
0.4	202	600.00	6.56	301482	618.08	1.05	1134606	600.01	4.08	5	2.77	0.00
0.5	422	600.00	8.29	277712	618.36	2.75	951881	600.02	6.62	9	5.63	0.00
0.5	330	600.00	7.49	223351	614.05	3.58	1293884	600.01	5.14	9	4.23	0.00

Table 5 (continued)

α	BARON			CPLEX			GUROBI			B&T		
	Nodes	Time	GAP %	Nodes	Time	GAP%	Nodes	Time	GAP%	Nodes	Time	GAP%
0.5	347	600.00	12.02	167573	610.90	8.52	751575	600.00	9.88	41	18.15	0.00
0.5	460	600.01	11.20	195195	612.44	6.90	901137	600.02	8.55	155	65.84	0.00
0.5	311	600.00	9.62	303748	609.29	0.43	1353263	600.02	6.13	93	41.94	0.00
0.5	252	600.00	11.03	247464	618.26	5.03	1174272	600.02	7.39	5	3.04	0.00
0.5	558	600.00	10.95	153238	609.17	7.34	805812	600.01	8.49	167	60.57	0.00
0.5	374	600.00	9.08	174729	610.75	5.44	855007	600.01	6.97	137	52.99	0.00
0.5	300	600.00	6.67	303274	614.59	0.51	1204530	600.01	4.44	179	68.47	0.00
0.5	266	600.00	9.52	274553	618.96	3.23	1077081	600.01	6.27	11	5.75	0.00
0.6	310	600.01	10.89	252753	618.41	5.66	979956	600.01	9.64	19	9.6	0.00
0.6	394	600.00	10.63	209382	613.95	5.96	1577641	600.01	6.96	7	3.88	0.00
0.6	351	600.00	16.14	184145	611.93	10.90	871458	600.02	13.28	35	13.14	0.00
0.6	419	600.00	15.09	200061	614.31	10.05	1042226	600.01	12.02	129	54.31	0.00
0.6	338	600.00	12.65	283490	621.91	6.23	1538267	600.02	8.20	77	27.16	0.00
0.6	327	600.00	14.50	239577	617.91	8.48	1305936	600.02	9.67	19	13.02	0.00
0.6	385	600.00	13.37	205419	613.76	8.32	845484	600.01	11.99	29	13.19	0.00
0.6	397	600.00	12.45	197769	613.27	7.31	1054022	600.02	8.48	35	7.24	0.00
0.6	360	600.00	8.99	299891	621.71	3.24	1447226	600.01	5.69	83	31.1	0.00
0.6	300	600.00	13.38	249919	619.09	7.26	1131577	600.01	8.85	25	11.96	0.00
0.7	255	600.00	13.42	276222	620.71	6.19	1137346	600.01	11.50	25	8.75	0.00
0.7	434	600.00	13.56	228522	615.59	7.67	1201021	600.01	9.56	9	2.81	0.00
0.7	391	600.00	20.55	183096	611.59	15.07	1058074	600.01	16.08	31	11.91	0.00
0.7	420	600.00	19.24	219961	615.33	11.10	1141047	600.01	15.44	275	85.43	0.00
0.7	239	600.00	15.41	299591	623.11	6.21	2414741	600.01	10.14	57	14.34	0.00
0.7	390	600.00	18.15	243125	618.40	11.49	1390127	600.01	13.32	29	6.72	0.00

Table 5 (continued)

α	BARON			CPLEX			GUROBI			B&T		
	Nodes	Time	GAP %	Nodes	Time	GAP%	Nodes	Time	GAP%	Nodes	Time	GAP%
0.7	273	600.00	15.23	236451	616.06	8.18	1080864	600.01	13.95	47	15.88	0.00
0.7	277	600.00	16.08	214992	615.18	10.95	1293717	600.01	10.41	21	4.02	0.00
0.7	286	600.00	12.55	261691	620.10	7.17	1560435	600.02	7.19	37	10.66	0.00
0.7	356	600.00	18.16	200648	613.53	11.70	1236194	600.02	12.35	45	19.21	0.00
0.8	170	600.00	13.63	198946	433.14	0.10	1487063	600.02	9.38	13	3.63	0.00
0.8	278	600.00	16.69	231555	615.48	10.23	1479539	600.02	10.27	19	4.93	0.00
0.8	281	600.00	23.47	241550	617.06	13.66	1372072	600.02	13.81	43	6.88	0.00
0.8	345	600.00	22.64	265972	619.72	14.52	1630755	600.02	13.48	21	6.98	0.00
0.8	165	600.00	15.40	257408	616.26	6.32	2877958	600.01	7.35	91	15.06	0.00
0.8	336	600.00	24.06	257779	618.89	14.30	1587465	600.01	14.68	25	4.34	0.00
0.8	196	600.00	19.45	205404	612.87	10.65	1365979	600.01	12.65	143	36.72	0.00
0.8	277	600.00	20.62	239183	616.55	10.93	1555594	600.02	11.37	51	6.49	0.00
0.8	234	600.00	15.52	272562	620.25	8.54	1841198	600.02	7.60	47	9.83	0.00
0.8	239	600.00	22.25	263579	620.49	12.09	1373578	600.01	15.98	93	17.13	0.00
0.9	23	365.29	0.10	47455	71.99	0.10	444863	122.88	0.00	27	6.54	0.00
0.9	87	600.00	17.69	264740	617.79	10.69	2891948	600.01	6.35	47	5.58	0.00
0.9	18	280.98	0.10	188473	441.33	0.10	267670	84.72	0.00	47	6.33	0.00
0.9	22	363.67	0.10	157699	287.12	0.10	365266	85.57	0.00	5	1.56	0.00
0.9	24	331.10	0.10	187821	372.63	0.10	425755	79.92	0.00	101	8.37	0.00
0.9	23	309.88	0.10	128276	283.31	0.10	303730	72.72	0.00	55	7.05	0.00
0.9	14	318.22	0.10	212681	471.67	0.10	224379	60.36	0.07	63	8.22	0.00
0.9	48	600.00	14.28	258838	615.16	7.49	2209403	600.03	0.47	97	10.84	0.00
0.9	126	600.01	13.17	271329	491.68	0.10	2129791	532.29	0.09	85	11.54	0.00
0.9	20	348.66	0.10	253656	613.46	7.30	517673	133.18	0.00	89	9.59	0.00

Table 6 Complete results for B&T over random instances of problems with dimension $n = 50$, α ranging between 0.3 and 0.9, and tolerance 10^{-5}

α	nn	#LPs	Time	GAP%
0.3	11	5283	11.14	0.00
0.3	29	17606	20.63	0.00
0.3	11	5754	7.05	0.00
0.3	277	195089	234.67	0.00
0.3	205	101779	112.15	0.00
0.3	65	25861	27.90	0.00
0.3	87	62146	72.52	0.00
0.3	73	55425	64.15	0.00
0.3	139	92675	106.65	0.00
0.3	15	9332	10.39	0.00
0.4	11	6040	7.36	0.00
0.4	119	56925	63.32	0.00
0.4	105	53535	58.64	0.00
0.4	887	515650	595.72	0.00
0.4	1099	538834	600.37	0.22
0.4	13	6142	7.19	0.00
0.4	507	230814	253.32	0.00
0.4	545	289835	326.33	0.00
0.4	621	353526	393.11	0.00
0.4	11	5540	6.73	0.00
0.5	41	24084	27.46	0.00
0.5	21	10952	12.89	0.00
0.5	113	51903	57.15	0.00
0.5	299	119675	130.64	0.00
0.5	167	76745	83.22	0.00
0.5	5	3415	4.62	0.00
0.5	707	281019	300.12	0.00
0.5	327	123507	133.36	0.00
0.5	581	234135	255.79	0.00
0.5	41	24998	26.62	0.00
0.6	51	22901	24.94	0.00
0.6	11	4304	6.19	0.00
0.6	53	21124	23.88	0.00
0.6	189	78217	82.27	0.00
0.6	99	31888	33.13	0.00
0.6	33	13745	16.54	0.00
0.6	51	20718	23.38	0.00
0.6	39	9034	9.63	0.00
0.6	127	45572	49.18	0.00
0.6	51	16170	18.91	0.00
0.7	35	8701	10.04	0.00
0.7	9	3406	4.98	0.00
0.7	41	10444	11.07	0.00

Table 6 (continued)

α	nn	#LPs	Time	GAP%
0.7	487	115543	118.45	0.00
0.7	71	16075	15.33	0.00
0.7	31	6536	7.17	0.00
0.7	95	25500	28.54	0.00
0.7	23	3829	4.29	0.00
0.7	43	11448	12.20	0.00
0.7	67	18609	21.39	0.00
0.8	15	3754	4.23	0.00
0.8	19	3547	3.83	0.00
0.8	43	7331	7.22	0.00
0.8	29	7413	8.12	0.00
0.8	99	14061	12.50	0.00
0.8	21	4294	4.55	0.00
0.8	179	33030	34.59	0.00
0.8	39	6700	6.58	0.00
0.8	51	9861	9.96	0.00
0.8	95	15017	14.47	0.00
0.9	25	4767	5.08	0.00
0.9	47	5387	5.14	0.00
0.9	55	6550	6.15	0.00
0.9	5	1852	2.44	0.00
0.9	101	7792	6.83	0.00
0.9	49	5847	5.78	0.00
0.9	63	6949	6.74	0.00
0.9	63	5687	5.03	0.00
0.9	89	9347	8.75	0.00
0.9	91	8425	7.56	0.00

Table 7 Complete numerical results for the comparison between various bound tightening strategies for algorithm B&T

	No bound tightening				Light bound tightening				Strong bound tightening			
α	Nodes	#lps	Time	GAP%	Nodes	#lps	Time	GAP%	nodes	#lps	Time	GAP%
0.3	110411	110414	600.01	1.38	45	2081	7.73	0.00	3	1389	7.54	0.00
0.3	104983	104986	600.01	1.37	117	5275	7.21	0.00	5	2382	3.65	0.00
0.3	108667	108670	600.01	1.15	75	3343	4.83	0.00	3	1259	2.23	0.00
0.3	103993	103996	600.01	2.66	1333	61116	83.63	0.00	25	11219	14.24	0.00
0.3	105677	105680	600.00	2.29	911	41552	53.47	0.00	39	13992	16.67	0.00
0.3	109691	109694	600.01	2.12	229	10158	13.09	0.00	15	4617	6.15	0.00
0.3	108237	108240	600.01	2.29	489	21752	29.30	0.00	9	4199	5.65	0.00
0.3	105505	105508	600.01	2.43	521	23854	31.75	0.00	5	3101	4.37	0.00
0.3	102763	102766	600.01	1.91	455	20895	27.91	0.00	5	3387	4.67	0.00
0.3	109421	109424	600.00	1.39	55	2501	3.51	0.00	3	1105	1.84	0.00
0.4	114313	114316	600.01	1.96	65	2953	3.92	0.00	3	1381	2.49	0.00
0.4	110321	110324	600.01	2.17	341	14440	19.52	0.00	11	3998	5.79	0.00
0.4	114203	114206	600.01	1.96	769	33128	44.17	0.00	19	6852	8.79	0.00
0.4	110271	110274	600.01	4.17	7603	346771	459.69	0.00	135	60330	73.05	0.00
0.4	110005	110008	600.00	3.88	10303	463434	600.01	0.13	271	99345	115.94	0.00
0.4	112983	112986	600.00	2.63	123	5224	7.01	0.00	9	2404	3.58	0.00
0.4	113319	113322	600.00	3.41	3113	137531	177.20	0.00	79	28258	33.54	0.00
0.4	113269	113272	600.00	3.77	4597	206674	270.09	0.00	107	42825	51.43	0.00
0.4	108125	108128	600.01	3.07	1835	82792	107.60	0.00	61	23760	28.38	0.00
0.4	113003	113006	600.01	2.28	85	3755	5.10	0.00	5	1622	2.77	0.00
0.5	118889	118892	600.00	2.72	233	9712	13.19	0.00	9	3874	5.63	0.00
0.5	115619	115622	600.01	3.02	207	8398	11.43	0.00	9	2965	4.23	0.00
0.5	117685	117688	600.01	3.18	1785	76149	98.70	0.00	41	14242	18.15	0.00

Table 7 (continued)

	No bound tightening				Light bound tightening				Strong bound tightening			
α	Nodes	#lps	Time	GAP%	Nodes	#lps	Time	GAP%	nodes	#lps	Time	GAP%
0.5	117515	117518	600.00	5.66	3939	174038	220.90	0.00	155	55206	65.84	0.00
0.5	116919	116922	600.01	5.70	3231	141810	177.74	0.00	93	35390	41.94	0.00
0.5	120505	120508	600.01	3.34	123	4958	6.74	0.00	5	1925	3.04	0.00
0.5	117217	117220	600.01	4.81	3369	139806	185.61	0.00	167	48601	60.57	0.00
0.5	117795	117798	600.00	5.42	2623	110557	144.71	0.00	137	42937	52.99	0.00
0.5	114831	114834	600.00	4.30	3717	162722	210.93	0.00	179	57501	68.47	0.00
0.5	118681	118684	600.01	3.75	271	11069	14.94	0.00	11	4045	5.75	0.00
0.6	122649	122652	600.00	3.79	343	13535	18.47	0.00	19	6893	9.6	0.00
0.6	120833	120836	600.01	3.83	113	4353	6.06	0.00	7	2361	3.88	0.00
0.6	122259	122262	600.00	4.10	961	40108	50.37	0.00	35	10673	13.14	0.00
0.6	112075	112078	600.00	8.07	4449	184494	235.41	0.00	129	45165	54.31	0.00
0.6	114885	114888	600.00	7.75	1931	81916	98.60	0.00	77	23555	27.16	0.00
0.6	117107	117110	600.01	4.89	505	18465	25.44	0.00	19	9271	13.02	0.00
0.6	117699	117702	600.00	5.47	885	35256	44.75	0.00	29	10215	13.19	0.00
0.6	122133	122136	600.00	5.69	227	9312	11.05	0.00	35	6269	7.24	0.00
0.6	116661	116664	600.00	5.50	1081	46073	57.46	0.00	83	25607	31.1	0.00
0.6	113123	113126	600.00	5.65	573	22301	29.69	0.00	25	8816	11.96	0.00
0.7	118717	118720	600.00	5.33	405	15074	20.44	0.00	25	6485	8.75	0.00
0.7	125675	125678	600.01	3.40	65	2426	3.32	0.00	9	1992	2.81	0.00
0.7	127137	127140	600.01	4.47	341	13609	16.54	0.00	31	9504	11.91	0.00
0.7	125293	125296	600.00	10.91	3671	140527	179.20	0.00	275	70938	85.43	0.00
0.7	126751	126754	600.01	9.02	821	31203	37.62	0.00	57	13047	14.34	0.00
0.7	128313	128316	600.00	4.95	255	8927	11.85	0.00	29	5343	6.72	0.00

Table 7 (continued)

	No bound tightening				Light bound tightening				Strong bound tightening			
α	Nodes	#lps	Time	GAP%	Nodes	#lps	Time	GAP%	nodes	#lps	Time	GAP%
0.7	128455	128458	600.02	6.75	805	29345	38.77	0.00	47	12328	15.88	0.00
0.7	86603	86606	600.01	6.19	133	5124	6.12	0.00	21	3327	4.02	0.00
0.7	85403	85406	600.00	6.42	307	12083	14.88	0.00	37	9075	10.66	0.00
0.7	86247	86250	600.01	8.57	1335	47897	65.73	0.00	45	14010	19.21	0.00
0.8	87179	87182	600.01	6.49	147	5264	6.91	0.00	13	2828	3.63	0.00
0.8	92415	92418	600.01	2.64	63	2326	2.94	0.00	19	3719	4.93	0.00
0.8	92645	92648	600.00	3.22	187	6850	8.20	0.00	43	5984	6.88	0.00
0.8	90021	90024	600.02	11.63	233	8261	10.29	0.00	21	5408	6.98	0.00
0.8	89669	89672	600.01	9.09	515	17362	21.23	0.00	91	13655	15.06	0.00
0.8	91903	91906	600.01	4.36	131	4484	5.79	0.00	25	3540	4.34	0.00
0.8	91247	91250	600.01	9.23	827	26964	35.37	0.00	143	28591	36.72	0.00
0.8	90817	90820	600.01	5.39	141	5029	5.95	0.00	51	5923	6.49	0.00
0.8	89261	89264	600.01	6.20	321	11617	14.22	0.00	47	8291	9.83	0.00
0.8	91311	91314	600.01	9.41	505	17849	21.37	0.00	93	14531	17.13	0.00
0.9	88037	88040	600.00	6.67	165	5300	6.88	0.00	27	4879	6.54	0.00
0.9	105609	105612	600.00	0.16	189	5922	7.50	0.00	47	4956	5.58	0.00
0.9	72451	72454	399.49	0.00	265	8503	10.78	0.00	47	5669	6.33	0.00
0.9	93485	93488	600.00	5.33	35	1229	1.74	0.00	5	999	1.56	0.00
0.9	93089	93092	553.74	0.00	177	5565	7.06	0.00	101	7728	8.37	0.00
0.9	97929	97932	600.01	2.60	175	5352	7.02	0.00	55	6067	7.05	0.00
0.9	74985	74988	439.06	0.00	185	5849	7.42	0.00	63	7175	8.22	0.00
0.9	81611	81614	473.46	0.00	165	5212	6.57	0.00	97	9225	10.84	0.00
0.9	86963	86966	474.28	0.00	265	8471	10.81	0.00	85	9562	11.54	0.00
0.9	100867	100870	571.30	0.00	211	6699	8.55	0.00	89	8512	9.59	0.00

Funding Open access funding provided by Università degli Studi di Parma within the CRUI-CARE Agreement.

References

1. OpenStreetMap (2021). https://www.openstreetmap.org/#map=15/15.3368/76.4601
2. Alpcan, T., Başar, T.: Network security: A decision and game-theoretic approach. Cambridge University Press (2010)
3. Alpern, S., Lidbetter, T., Morton, A., Papadaki, K.: Patrolling a Pipeline. In: Zhu, Q., Alpcan, T., Panaousis, E., Tambe, M., Casey, W. (eds.) Decision and Game Theory for Security. Lecture Notes in Computer Science, pp. 129–138. Springer International Publishing, Cham (2016)
4. Alpern, S., Morton, A., Papadaki, K.: Patrolling games. Operations Research **59**(5), 1246–1257 (2011)
5. Basak, A., Fang, F., Nguyen, T.H., Kiekintveld, C.: Combining Graph Contraction and Strategy Generation for Green Security Games. In: Zhu, Q., Alpcan, T., Panaousis, E., Tambe, M., Casey, W. (eds.) Decision and Game Theory for Security. Lecture Notes in Computer Science, pp. 251–271. Springer International Publishing, Cham (2016)
6. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. SIAM review **59**(1), 65–98 (2017). https://doi.org/10.1137/141000671
7. Bonami, P., Günlük, O., Linderoth, J.: Globally solving nonconvex quadratic programming problems with box constraints via integer programming methods. Mathematical Programming Computation **10**, 333–382 (2018)
8. Caprara, A., Locatelli, M.: Global optimization problems and domain reduction strategies. Mathematical Programming **125**(1), 123–137 (2010)
9. Chen, J., Burer, S.: Globally solving nonconvex quadratic programming problems via completely positive programming. Mathematical Programming Computation **4**(1), 33–52 (2012)
10. Fang, F., Jiang, A.X., Tambe, M.: Optimal patrol strategy for protecting moving targets with multiple mobile resources. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, AAMAS '13, pp. 957–964. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2013)
11. Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A., Gould, N., Liberti, L., Lodi, A., Misener, R., Mittelmann, H., et al.: Qplib: a library of quadratic programming instances. Mathematical Programming Computation **11**(2), 237–265 (2019)
12. Gleixner, A., Berthold, T., Müller, B., Weltge, S.: Three enhancements for optimization-based bound tightening. Journal of Global Optimization **67**, 731–757 (2017)
13. Gondzio, J., Yildirim, E.A.: Global solutions of nonconvex standard quadratic programs via mixed integer linear programming reformulations. Journal of Global Optimization **to appear** (2021)
14. Hansen, K.A., Koucky, M., Lauritzen, N., Miltersen, P.B., Tsigaridas, E.P.: Exact algorithms for solving stochastic games. In: Proceedings of the forty-third annual ACM symposium on Theory of computing, pp. 205–214 (2011)
15. Horst, R., Tuy, H.: Global optimization: Deterministic approaches (2nd edition). Springer Science & Business Media (2013)
16. Lozovanu, D., Solomon, D., Zelikovsky, A.: Multiobjective games and determining pareto-nash equilibria. Buletinul Academiei de Ştiinţe a Republicii Moldova. Matematica **3**, 115–122 (2005)

17. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part i-convex underestimating problems. Mathematical Programming **10**(1), 147–175 (1976)
18. Motzkin, T., Straus, E.: Maxima for graphs and a new proof of a theorem of Turán. Canadian Journal of Mathematics **17**(4), 533–540 (1965)
19. Padgham, M.: dodgr: An r package for network flow aggregation. Transport Findings (2019). 10.32866/6945
20. Padgham, M.: GitHub - ATFutures/dodgr: Distances on Directed Graphs in R (2021). https://github.com/ATFutures/dodgr
21. Papadaki, K., Alpern, S., Lidbetter, T., Morton, A.: Patrolling a Border. Operations Research **64**(6), 1256–1269 (2016). https://doi.org/10.1287/opre.2016.1511
22. Pita, J., Tambe, M., Kiekintveld, C., Cullen, S., Steigerwald, E.: GUARDS - Innovative Application of Game Theory for National Airport Security. IJCAI (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-451
23. P.M., P., S.A., V.: Quadratic programming with one negative eigenvalue is NP-hard. Journal of Global Optimization **1**(1), 15–22 (1991)
24. Rass, S., Alshawish, A., Abid, M.A., Schauer, S., Zhu, Q., De Meer, H.: Physical intrusion games-optimizing surveillance by simulation and game theory. IEEE Access **5**, 8394–8407 (2017)
25. Rass, S., König, S.: Password security as a game of entropies. Entropy **20**(5), 312 (2018)
26. Rass, S., König, S., Schauer, S.: On the cost of game playing: How to control the expenses in mixed strategies. In: International Conference on Decision and Game Theory for Security, pp. 494–505. Springer (2017)
27. Rass, S., Rainer, B.: Numerical computation of multi-goal security strategies. In: International Conference on Decision and Game Theory for Security, pp. 118–133. Springer (2014)
28. Rass, S., Schauer, S., König, S., Zhu, Q.: Cyber-Security in Critical Infrastructures: A Game-Theoretic Approach. SpringerNature (2020)
29. Sahinidis, N.V.: Baron: A general purpose global optimization software package. Journal of Global Optimization **8**(2), 201–205 (1996)
30. Tambe, M.: Security and game theory: algorithms, deployed systems, lessons learned. Cambridge University Press (2011)
31. Tawarmalani, M., Sahinidis, N.V.: Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. Mathematical Programming **99**(3), 563–591 (2004)
32. Wachter, J., Rass, S., König, S.: Security from the adversary's inertia-controlling convergence speed when playing mixed strategy equilibria. Games **9**(3), 59 (2018)
33. Xia, W., Vera, J.C., Zuluaga, L.F.: Globally solving nonconvex quadratic programs via linear integer programming techniques. INFORMS Journal on Computing **32**(1), 40–56 (2020)

Authors and Affiliations

G. Liuzzi[1] · M. Locatelli[2] · V. Piccialli[3] · S. Rass[4]

G. Liuzzi
giampaolo.liuzzi@cnr.iasi.it

V. Piccialli
veronica.piccialli@uniroma2.it

S. Rass
stefan.rass@aau.at

1 Consiglio Nazionale delle Ricerche (CNR), Istituto di Analisi dei Sistemi ed Informatica “Antonio Ruberti” (IASI), Via dei Taurini 19, 00185 Rome, Italy

2 Parco Area delle Scienze, Università degli Studi di Parma, 181/A, 43124 Parma, Italy

3 DICII - University of Rome Tor Vergata, via del Politecnico 1, 00133 Rome, Italy

4 System Security Group, Institute of Applied Informatics, , Universität Klagenfurt, Klagenfurt, Austria