A Cut-and-Branch Algorithm for the Quadratic Knapsack Problem

Franklin Djeumou Fomeni* Konstantinos Kaparis†
Adam N. Letchford‡

To appear in *Discrete Optimization*

Abstract

The *Quadratic Knapsack Problem* (QKP) is a well-known $\mathcal{NP}$-hard combinatorial optimisation problem, with many practical applications. We present a 'cut-and-branch' algorithm for the QKP, in which a cutting-plane phase is followed by a branch-and-bound phase. The cutting-plane phase is more sophisticated than the existing ones in the literature, incorporating several classes of cutting planes, two primal heuristics, and several rules for eliminating variables and constraints. Computational results show that the algorithm is competitive.

Keywords: knapsack problems, cutting planes, integer programming.

1 Introduction

The *Quadratic Knapsack Problem* (QKP) takes the form:

$$\max \quad \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{1}$$

$$x \in \{0,1\}^n, \tag{2}$$

where n is the number of *items*, c is the (positive and integral) *capacity*, the w_i are the (positive and integral) *weights*, and the q_{ij} are the (integral)

*Department of Management and Technology, University of Quebec in Montreal, CIRRELT, Montreal, Canada. E-mail: franklin@aims.ac.za

†Department of Business Administration, University of Macedonia, Egnatia Street, 54636 Thessaloniki, Greece. E-mail: k.kaparis@uom.edu.gr

‡Department of Management Science, Lancaster University, Lancaster LA1 4YX, United Kingdom. E-mail: A.N.Letchford@lancaster.ac.uk

profits. The QKP includes as a special case the standard (linear) knapsack problem (KP), which is obtained when $q_{ij} = 0$ for all $i \neq j$.

The QKP, introduced in Gallo *et al.* [13], has applications in finance, logistics and telecommunications. It is $\mathcal{NP}$-hard in the strong sense, by reduction from the maximum clique problem [5]. For good surveys of applications, bounding procedures, heuristics and exact algorithms, see [19, 22]. At present, the most effective exact algorithms are the ones in [5, 23], which are based on Lagrangian relaxation.

The main contribution of this paper is a 'cut-and-branch' framework for the QKP. This framework incorporates four families of cutting planes, two primal heuristics, and several rules for eliminating variables and constraints. We also present the results of some extensive computational experiments. As well as testing our algorithm on standard QKP instances, such as those used in [5,23], we also test it on some hard special cases of the QKP, namely, the *dispersion problem* [23], the *densest sub-graph problem* [23] and the *hidden clique problem* [27]. All of these were recently shown to be extremely challenging for existing QKP algorithms [27]. Our computational results show that our algorithm is competitive with the best existing algorithms.

The paper is structured as follows. In Section 2, we review the relevant literature. In Section 3, we present several *separation routines* (i.e., routines for generating cutting planes). In Section 4, we describe the other components of the cut-and-branch algorithm. The computational results are given in Section 5. Finally, some concluding remarks are made in Section 6.

Throughout the paper, we assume that the reader is familiar with the basics of integer programming, including cutting planes, branch-and-bound and separation algorithms (see [30] for an introduction). We also let N denote $\{1, \ldots, n\}$ and E denote $\{S \subset N : |S| = 2\}$. (One can think of N and E as the nodes and edges, respectively, of a complete undirected graph.)

2 Literature Review

Since the literature on the QKP is vast, we review here only results that are of direct relevance, and refer the reader to the surveys [19, 22].

2.1 Standard formulation of the QKP

Using a standard construction [14], one can formulate the QKP as a 0-1 *linear* program (0-1 LP). Let the x variables be defined as above. For all $\{i, j\} \in E$, let y_{ij} be an additional binary variable, representing the product

$x_i x_j$. The formulation is then:

$$\begin{aligned}
\max \quad & \textstyle\sum_{i\in N} q_{ii}x_i + \sum_{\{i,j\}\in E}(q_{ij}+q_{ji})y_{ij} && && (3)\\
\text{s.t.} \quad & (1), (2) && && \\
& y_{ij} \le x_i && \big(\{i,j\}\in E\big) && (4)\\
& y_{ij} \le x_j && \big(\{i,j\}\in E\big) && (5)\\
& y_{ij} \ge x_i + x_j - 1 && \big(\{i,j\}\in E\big) && (6)\\
& y \in \{0,1\}^E. && &&
\end{aligned}$$

This formulation is used in [2, 12, 16]. We will call the inequalities (4)–(6) *trivial* inequalities.

2.2 Cutting planes for the KP

The convex hull of vectors $x \in \{0,1\}^n$ satisfying the knapsack constraint (1) is called the *knapsack polytope*, and has been studied in depth (see, e.g., [1, 15, 18, 20, 28, 29]).

The most well-known valid inequalities for the knapsack polytope are the *lifted cover* (LC) inequalities. A set $C \subset N$ is called a *cover* if $\sum_{i\in C} w_i > c$. The cover is *minimal* if no proper subset of it is a cover. Balas [1] and Wolsey [29] showed that, given any minimal cover $C \subset N$, there exists at least one facet-defining LC inequality of the form:

$$\sum_{i\in C} x_i + \sum_{i\in N\setminus C} \alpha_i x_i \le |C| - 1$$

with $\alpha_i \ge 0$ for all $i \in N \setminus C$.

Balas [1] defined a simple family of (not necessarily facet-defining) LC inequalities. For a given cover C, let $E(C)$ denote the set of items in $N \setminus C$ that weigh at least as much as the heaviest item in C. Then one can simply set α_i to 1 if $i \in E(C)$, and to 0 otherwise. The resulting LC inequality is called an *extended* cover (EC) inequality.

2.3 Cutting planes for the QKP

The convex hull of pairs $(x, y) \in \{0,1\}^{N+E}$ satisfying (1) and (4)–(6) is called the *quadratic knapsack polytope* [16, 24, 25]. One key class of valid inequalities, mentioned in [2, 16], is obtained via the *reformulation-linearization technique* (RLT) of Sherali & Adams [26]. Take the knapsack constraint (1) and multiply it by a variable x_k or by its complement $1 - x_k$, and then replace any products of the form $x_i x_k$ with y_{ij}. This yields the following $2n$ inequalities:

$$\begin{aligned}
\textstyle\sum_{i\in N\setminus\{k\}} w_i y_{ik} &\le (c - w_k)x_k \quad (k \in N) && (7)\\
\textstyle\sum_{i\in N\setminus\{k\}} w_i (x_i - y_{ik}) &\le c - c x_k \quad (k \in N). && (8)
\end{aligned}$$

We will call these *RLT* inequalities.

It is pointed out in [16] that one can apply the RLT to any valid inequality for the knapsack polytope, not only the knapsack constraint itself. So, for example, given any LC inequality for the KP, one can derive $2n$ distinct valid inequalities for the QKP. It is shown in [12] that, if the original inequality defines a facet of the knapsack polytope, then the resulting inequalities define facets of the quadratic knapsack polytope, under mild conditions.

Another important class of inequalities for the QKP are the following *triangle* inequalities:

$$x_i + x_j + x_k \leq y_{ij} + y_{ik} + y_{jk} + 1 \quad \big(\{i,j,k\} \subset N\big) \tag{9}$$

$$y_{ik} + y_{jk} \leq x_k + y_{ij} \quad \big(\{i,j\} \in E,\ k \in N \setminus \{i,j\}\big). \tag{10}$$

These inequalities are well known in the literature on 0-1 quadratic problems; see, e.g., [8, 21]. They were applied to the QKP in [2, 16].

Finally, we mention the *cover-tree* (CT) inequalities [17]. Given a minimal cover C, let K_C be a complete graph with vertex set C, and let T be the set of edges in a spanning tree in K_C. For all $i \in C$, let d_i denote the degree of i in the tree. Then, the CT inequality takes the form:

$$\sum_{\{i,j\} \in T} y_{ij} \leq \sum_{i \in C} (d_i - 1) x_i. \tag{11}$$

Conditions for these to define facets are given in [24, 25]. Some weaker inequalities, called *matching-cover* inequalities, were given in [16].

2.4 Algorithms for the QKP

Finally, we mention the main exact and heuristic algorithms for the QKP.

The exact algorithms are all of branch-and-bound type, the chief difference being the procedure that is used to compute the upper bounds at each node. In [13], the upper bounds are obtained by solving KPs. In [3, 5, 6], they are obtained via Lagrangian relaxation. In [2], they are obtained with a cutting-plane algorithm, that uses the inequalities (1), (4)–(6), (7) and (9). In [4], they are based on a compact linearisation with only $O(n)$ variables.

Broadly speaking, the algorithm in [5] is the most effective, although the algorithms in [3, 4] are competitive for very sparse instances (i.e., instances in which the majority of the q_{ij} are zero). Pisinger *et al.* [23] presented an enhanced version of the algorithm in [5]. The idea is to apply several procedures for eliminating variables, and thereby reduce the size of the problem, before calling the algorithm in [5].

As for heuristics, some good early examples can be found in [2, 6, 13]. More recently, two of the present authors presented a new heuristic [11], which significantly outperformed the ones mentioned. It is based on the construction of an initial solution via dynamic programming, followed by

the local search procedure from [13]. We will use this heuristic at the start of our cut-and-branch algorithm, to obtain an initial lower bound.

3 Separation Routines

Our algorithm uses four different families of valid inequalities as cutting planes: the RLT inequalities (7), (8), the triangle inequalities (9), (10), the CT inequalities (11), and the inequalities obtained by applying the RLT to LC inequalities, which we call "RLT-LC" inequalities. In this section, we describe our separation routines for these four families.

Throughout this section, we let (x^*, y^*) denote the LP solution to be separated, and we assume that $(x^*, y^*) \in [0,1]^{N+E}$. Given a valid inequality $\alpha^T x + \beta^T y \leq \gamma$, the "violation" of the inequality is $\max\left\{\alpha^T x^* + \beta^T y^* - \gamma, 0\right\}$. For a given *family* of inequalities, the "most-violated" inequality is the one with maximum violation.

3.1 RLT inequalities

Since there are only $2n$ RLT inequalities, and each one has only $\mathcal{O}(n)$ non-zero coefficients, the RLT separation problem can be solved exactly in $\mathcal{O}(n^2)$ time by mere enumeration. It often happens that there exist several violated RLT inequalities. In that case, we add to the LP only the most-violated inequality of the form (7), if any, and the most-violated inequality of the form (8), if any. This is to prevent the LP from becoming too large.

3.2 Triangle inequalities

Since there are $\mathcal{O}(n^3)$ triangle inequalities, and each one has only $\mathcal{O}(1)$ non-zero coefficients, the triangle separation problem can be solved exactly in $\mathcal{O}(n^3)$ time by enumeration. Again, to prevent the LP from becoming too large, we add to the LP only $\mathcal{O}(n^2)$ violated inequalities in any given cutting-plane iteration. Specifically, for a given $\{i, j\} \in E$, we add to the LP only the most-violated triangle inequality of the form (9), if any, and the most-violated triangle inequality of the form (10), if any.

3.3 Linearisation operator

For our remaining separation routines, we have found it helpful to begin by mapping (x^*, y^*) onto $[0,1]^N$, i.e., onto the space of the standard (linear) knapsack polytope. This is done by applying one of the following three mappings:

- Just take x^* itself.

- For each $i \in N$ such that $x_i^* > 0$, create a point $\bar{x}^i \in [0,1]^N$ by setting $\bar{x}_i^i := x_i^*$ and setting $\bar{x}_j^i := y_{ij}^*/x_i^*$ for all $j \in N \setminus \{i\}$.
- For each $i \in N$ such that $x_i^* < 1$, create a point $\tilde{x}^i \in [0,1]^N$ by setting $\tilde{x}_i^i := x_i^*$ and setting $\tilde{x}_j^i := (x_j^* - y_{ij}^*)/(1 - x_i^*)$ for all $j \in N \setminus \{i\}$.

In this way, we obtain up to $2n+1$ points in the x-space. The idea underlying the second and third steps is that, in a feasible QKP solution, $x_j = y_{ij}/x_i$ when $x_i = 1$, and $(x_j - y_{ij})/(1 - x_i)$ when $x_i = 0$. So there is a chance that at least some of the points will lie close to the boundary of the knapsack polytope.

We call this whole approach the *linearisation operator*. Note that it takes $\mathcal{O}(n^2)$ time. The following two subsections explain how it is used.

3.4 CT inequalities

A simple separation heuristic for the CT inequalities, based on building the tree greedily, is presented in [17]. We, however, use a completely different heuristic, in which we first generate a list of candidate covers, and then compute the best tree for each candidate.

To generate the covers, one can simply take each of the fractional points generated by the linearisation operator, and feed it into any exact or heuristic separation algorithm for cover inequalities. It is important to note that, even if a cover inequality generated in this way is not violated, the associated cover could still lead to a violated CT inequality.

Now, the separation problem for CT inequalities, for a *fixed* minimal cover C, is easy. Re-write the CT inequality as:

$$\sum_{\{i,j\} \in T} (x_i + x_j - y_{ij}) \geq \sum_{i \in C} x_i.$$

There exists a violated CT inequality for the given cover C, if and only if there exists a tree T^* such that:

$$\sum_{\{i,j\} \in T} (x_i^* + x_j^* - y_{ij}^*) < \sum_{i \in C} x_i^*. \tag{12}$$

Since the right-hand side of (12) is a constant for fixed C and x^*, it suffices to find a tree T^* that minimises the left-hand side. This can be done by finding a minimum weight spanning tree in the graph G_C, where the weight of each edge $\{i,j\}$ is set to $y_{ij}^* - x_i^* - x_j^*$. This spanning tree can be found in $\mathcal{O}(|C|^2)$ time using any of several well-known algorithms.

To generate the covers, we use the separation heuristic in [7], which runs in $\mathcal{O}(n \log n)$ time. Since the linearisation operator creates $\mathcal{O}(n)$ fractional points, the time taken by this scheme is $\mathcal{O}(n^2 \log n)$ to create the covers, plus $\mathcal{O}(n^3)$ for the spanning tree computations. All violated CT inequalities found, if any, are added to the LP.

3.5 RLT-LC inequalities

Finally, we consider separation for the RLT-LC inequalities. One simple heuristic is to solve the separation problem for standard LC inequalities, using x^* as input, and then check the $2n$ corresponding RLT-LC inequalities for violation. The running time is then determined by the time taken by the LC separator, together with the time taken to check the RLT-LC inequalities, which is $\mathcal{O}(n^2)$.

A more interesting, but more time-consuming, heuristic is the following: take each of the points created by the linearisation operator, and feed it into an LC separator. Then, for each of the LC inequalities created in this way, check the $2n$ corresponding RLT-LC inequalities for violation. This increases the running time by a factor of n.

We found that, in the QKP context, it was usually enough to use only the inequalities obtained by applying the RLT to EC inequalities. We call the resulting inequalities *RLT-EC* inequalities. In our separation scheme, we feed the points from the linearisation operator into the EC separation heuristic described in Subsection 3.3 of [18]. That EC heuristic runs in $\mathcal{O}(n^2)$ time, which means that our RLT-EC heuristic runs in $\mathcal{O}(n^3)$ time. All violated RLT-EC inequalities found, if any, are added to the LP.

4 The Cut-and-Branch Algorithm

As stated in the introduction, we have devised a cut-and-branch algorithm for the QKP. In addition to the separation routines described in the previous section, the algorithm includes two primal heuristics and several rules for eliminating y variables. A high-level description of the algorithm is as follows:

1. Run the primal heuristic described in [11], yielding an initial lower bound L.
2. Solve an initial LP relaxation, and let U be the resulting upper bound.
3. Call the LP-based primal heuristic. If an improved primal solution is found, update L.
4. Call some separation routines. If any violated inequalities are found, go to step 5. Else go to step 12.
5. Add the inequalities to the LP and re-optimise via dual simplex.
6. If the upper bound has decreased, update U.
7. Call the LP-based primal heuristic. If an improved primal solution is found, update L.
8. If $L = \lfloor U \rfloor$, output the optimal solution and stop.
9. If any of Lemmas 1 to 3 apply (see Subsection 4.5), then mark the corresponding y variables for elimination (but don't actually delete them).

10. Delete any cutting planes whose slack exceeds some threshold.
11. Go to step 4.
12. Delete any cutting planes whose dual price is zero.
13. Eliminate all marked y variables from the formulation.
14. Change all x variables to binary and solve the resulting 0-1 LP via branch-and-bound.

The key components of this algorithm are described in more detail in the following subsections. As in the previous section, (x^*, y^*) denotes the optimal solution to the current LP relaxation.

4.1 Initial LP relaxation

The initial LP relaxation consists of the objective function (3) and the constraints (1) and (4)–(6) (plus non-negativity as usual). We do *not* include the upper bounds of 1 on the variables in the relaxation, either explicitly or implicitly, since (a) they are implied by the constraints (4)–(6) and non-negativity, and (b) their inclusion would substantially increase the amount of degeneracy present in the LP.

4.2 Separation order

We use four different classes of inequalities in the cutting-plane phase of our algorithm: RLT, triangle, CT and RLT-EC inequalities. Of these, the triangle inequalities are the sparsest, but they also tend to create primal degeneracy. Moreover, the CT and RLT-EC inequalities are being separated heuristically rather than exactly. For these reasons, the order in which the inequalities are separated can substantially affect the number of cutting-plane iterations, the time taken to solve each LP, the quality of the bounds obtained, the number of eliminated y variables, and the total computing time.

In some preliminary computational experiments, we tried several different combinations and orderings of the separation routines, in order to choose the best. (Details can be found in Appendix 1.) In the end, we decided to call RLT separation first. Then, if no violated RLT inequalities are found, we call triangle separation. If that fails, we call RLT-EC separation. Finally, if that fails too, we call CT separation.

4.3 Cut deletion

To prevent the LP from getting too large, we delete some cutting planes after each re-optimisation of the LP. More precisely, we delete all RLT, triangle, CT and RLT-EC inequalities whose slack exceeds a certain threshold. We do not delete the constraints (1) or (4)–(6), however, since those are part of the 0-1 LP formulation itself.

4.4 LP-based primal heuristic

Each time the current LP relaxation is solved, we call a primal heuristic that attempts to construct a good QKP solution using the information contained in the LP solution. This is a simple greedy heuristic that proceeds as follows. Sort the items in N in non-increasing order of x^* value. Start with an empty knapsack, and then examine each item in the sorted list. If there is sufficient remaining space in the knapsack for the item, then place the item in the knapsack.

For example, suppose that $n = 5$, $c = 100$, $w = (50, 40, 30, 20, 10)^T$ and $x^* = (0.6, 0.5, 0.2, 0.4, 0.3)^T$. The sorted list of items is $(1, 2, 4, 5, 3)$. Initially, the available capacity is 100, so we insert item 1. The remaining capacity is 50, so we insert item 2. The remaining capacity is 10, so we cannot insert item 4. We can however insert item 5. The knapsack is now full, so we terminate with items $1, 2$ and 5 in the knapsack.

4.5 Reduction procedures

After each major iteration of the cutting-plane algorithm, it helps to apply *reduction* procedures, that eliminate variables and constraints from the LP without causing any optimal solutions to be lost. One simple reduction procedure, usually attributed to Crowder *et al.* [7], is *reduced-cost fixing*: any variable that takes a zero value at (x^*, y^*), and has a reduced cost larger than $U - L$, can be permanently fixed to 0, and therefore removed from the problem. Our reduction procedures are more sophisticated.

Recall that, at any stage of the cutting-plane phase, the LP relaxation will always include the inequalities (4)–6 For any pair $\{i, j\} \in E$, we let λ^1_{ij}, λ^2_{ij} and λ^3_{ij} denote the dual prices of the constraints (4), (5) and (6), respectively. Also, for any $i \in N$, we let ρ_i denote the reduced cost of x_i. Finally, for any pair $\{i, j\} \in E$, we let ϕ_{ij} be the reduced cost of y_{ij}.

Our reduction procedure is then based on the following three lemmas:

Lemma 1 *If, for some $\{i, j\} \in E$, we have $\rho_i + \rho_j + \phi_{ij} > U - L$, then we can permanently remove y_{ij} from the problem (or, equivalently, fix y_{ij} to zero). Moreover, we can replace the constraints (4)–(6) with the single constraint $x_i + x_j \leq 1$.*

Proof. From the definition of reduced costs, it follows that, if we added to the LP the constraints $x_i \geq 1$, $x_j \geq 1$ and $y_{ij} \geq 1$, then the optimal profit would decrease by at least $\rho_i + \rho_j + \phi_{ij}$. Under the stated assumption, the profit would drop below L, thus showing that no optimal solution of the QKP could have $x_i = x_j = y_{ij} = 1$. As a result, y_{ij} must be zero in an optimal QKP solution. Setting y_{ij} to 0 and simplifying the constraints (4)–(6) yields the constraint $x_i + x_j \leq 1$. □

Lemma 2 *If, for some ordered pair (i, j), we have $\rho_i + \lambda^1_{ij} > U - L$, then we can replace y_{ij} with x_i everywhere in the formulation. Moreover, we can replace the constraints (4)–(6) with the constraints $x_i \leq x_j \leq 1$.*

Proof. Similar to that of Lemma 1. □

Lemma 3 *If, for some pair $\{i, j\} \in E$, we have $\lambda^3_{ij} > U - L$, then we can replace y_{ij} with $x_i + x_j - 1$ everywhere in the formulation. In particular, the constraints (4)–(6) can be replaced with the constraints $x_i \leq 1$, $x_j \leq 1$ and $x_i + x_j \geq 1$.*

Proof. Similar to that of Lemma 1. □

The computational results given later show that a substantial proportion of the y variables can be eliminated in this way. Moreover, even a relatively small amount of elimination can be highly beneficial in the branch-and-bound phase, because, each time a y variable is eliminated, the corresponding constraints (4)–(6) are eliminated and/or simplified as well.

4.6 Final 0-1 LP formulation

Once the cutting-plane algorithm has terminated and y variables have been eliminated, we take the LP, re-optimise it, and then delete all RLT, triangle, CT and RLT-EC inequalities that have zero dual price. This is to make the final LP as small as possible, while maintaining the quality of the upper bound. Then we add the condition that all variables must be binary, yielding a 0-1 LP formulation of the QKP. This formulation is then fed into a branch-and-bound solver.

5 Computational Results

In this section, we present our computational results. All our routines were coded in the C programming language and compiled with *gcc 4.6*. All the results were obtained by running our code on a single cluster node of the super computer of Compute Canada[1] with processor at 2.26 GHz and 24 GB of RAM while requiring only 8 GB for our code. The LP relaxations and the final 0-1 LPs were solved respectively, with the simplex and MIP solvers of `CPLEX` version 12.5.

5.1 Test instances

Until recently, the "standard" scheme for creating test instances for the QKP was the one proposed by Gallo *et al.* [13]. For a given value of n, each weight

[1] www.computecanada.ca

w_i is an integer uniformly distributed between 1 and 100. The knapsack capacity c is an integer uniformly distributed between 50 and $\sum_{i \in N} w_i$. Finally, for a given choice of *density parameter* $\Delta\%$, each profit term q_{ij} is set to zero with probability $(100 - \Delta)\%$, and set to an integer uniformly distributed between 1 and 100 with probability $\Delta\%$.

For each value of $n \in \{10, 20, \ldots, 100\}$ and $\Delta \in \{25\%, 50\%, 75\%, 100\%\}$, we created 10 random standard instances. We call the resulting 400 instances "small". These small instances were used only to explore the effect of different separation orders (see Appendix 1). To test our overall algorithm, we used much larger standard instances. In detail, we created 10 random standard instances for each combination of $n \in \{50, 100, 150, \ldots, 800\}$ and $\Delta \in \{25\%, 50\%, 75\%, 100\%\}$. This makes an additional 640 instances, which we call "large".

As pointed out in [23,27], standard instances are relatively easy for modern exact algorithms. Therefore, we also consider several other families of instances, as described in the following subsections.

5.1.1 Dispersion problem instances

The dispersion problem consists of locating q facilities at n possible locations, while maximising the sum of the pairwise distances between facilities. The QKP formulation of this problem is as follows [23]:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_i x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = q \\ & x \in \{0,1\}^n, \end{aligned}$$

where d_{ij} is the distance between locations i and j. Several variants of this problem, which differ by the distribution of the d_{ij}, are proposed in [23]:

- *GEO* (*Geometrical problems*): the n locations are randomly located in a 100×100 square, and d_{ij} is the Euclidean distance between locations i and j.
- *WGEO* (*Weighted geometrical problems*): the locations are again randomly located in a square, but each location i is assigned a weight α_i in the interval $[5, \ldots, 10]$. The distance d_{ij} is then $\alpha_i \alpha_j$ times the Euclidean distance between locations i and j.
- *EXPO* (*Exponential problems*): for each pair $\{i, j\}$ of locations, d_{ij} is randomly drawn from a negative exponential distribution with mean 50.

- *RAN* (*Random problems*): for each pair $\{i, j\}$, d_{ij} is uniformly distributed in $[1, \ldots, 100]$.

For all these instances, $d_{ii} = 0$ for $i = 1, \ldots, n$, and the number of facilities q is randomly chosen in $[2, \ldots, n-2]$.

Pisinger *et al.* [23] also presented a knapsack-like version of the four above-mentioned problem types. These are obtained by generating a weight w_i (randomly in $[1, \ldots, 100]$) for each location i, setting q to $\lfloor \frac{1}{2} \sum_{i=1}^{n} w_i \rfloor$, and changing the knapsack constraint accordingly. These instances will be denoted $KP - \{EXPO, GEO, WGEO, RAN\}$.

We generated ten instances for each combination of problem type and $n \in \{25, 50, 100, 200, 400\}$, for a total of 400 instances.

5.1.2 Densest subgraph instances

The densest subgraph problem was also formulated as a QKP by Pisinger *et al.* [23]. Given a graph $G = (V, E)$, this problem amounts to finding a set of nodes $U \subseteq V$ of cardinality q for which the induced subgraph contains the most possible edges. Variants of this problem are obtained by varying the density of G. We experimented with the following settings:

- $DSUB25$: Here $d_{ij} = 1$ with probability 25%, $d_{ij} = 0$ otherwise.
- $DSUB50$: Here $d_{ij} = 1$ with probability 50%, $d_{ij} = 0$ otherwise.
- $DSUB75$: Here $d_{ij} = 1$ with probability 75%, $d_{ij} = 0$ otherwise.
- $DSUB90$: Here $d_{ij} = 1$ with probability 90%, $d_{ij} = 0$ otherwise.

Again, $d_{ii} = 0, i = 1, \ldots, n$ and the number of nodes q is randomly chosen in $[2, \ldots, n-2]$.

We generated ten instances for each combination of problem type and $n \in \{25, 50, 100, 200, 400\}$, for a total of 200 instances.

5.1.3 Planted-clique instances

To further test the limits of our procedure, we also created some "planted clique" (PC) instances. Instances of this type were recently introduced by Schauer [27], who showed that they are very challenging for existing exact and heuristic QKP algorithms. For a given n, one generates a random (so-called Erdős-Rényi) graph, in which each edge is present with probability 1/2. One then "plants" a clique in it, by selecting a random set of $\lfloor n \rfloor$ nodes, and adding edges, where necessary, so that those nodes form a clique. The knapsack capacity is then set to $\lfloor n \rfloor$. The weight of each vertex is 1, its linear profit is 0 and the quadratic profit is 1 whenever an edge occurs in the graph, and 0 otherwise. The optimal solution value is then almost

surely $\frac{1}{2}\lfloor n \rfloor \big(\lfloor n \rfloor - 1\big)$. We generated 10 PC instances for each value of $n \in \{20, 40, 60, \ldots, 200\}$, making 100 in total.

5.2 Results for the standard instances

As mentioned above, we used the "small" standard instances only to explore the effect of different separation orders. Table 1 shows the performance of our cut-and-branch algorithm on the "large" standard instances. In this table, we report the percentage gap of the upper bound found by our cutting-plane algorithm, the time taken to compute this upper bound, the percentage gap of the lower bound, the number of branch-and-bound nodes, and the total computing time. The number of y variables reduced reported here is the average over the instances that were *not* already solved during the cutting-plane phase. All the other values are averages over 10 instances. (More detailed statistics are presented in Appendix 2.) For the branch-and-bound phase, we used the standard MIP solver of `CPLEX`, with default settings, except that all internal cutting-plane generators were switched off.

The results in Table 1 are interesting for several reasons. First, the upper bounds found by our algorithm are within 0.5% of optimality for almost all instances. Second, the primal heuristic consistently produces solutions within 0.1% of optimality. These tight bounds lead to a significant proportion of y variables being eliminated. This in turn eases the burden on the branch-and-bound solver, as evidenced by the relatively low number of branch-and-bound nodes required. Finally, looking at the last column of Table 1, it is apparent that all 640 large instances could be solved within a five-hour time limit. In fact, more than 50% of the instances were solved within one hour. Overall, it is clear that our algorithm is capable of solving QKP instances with up to 800 items, regardless of their density, within a reasonable amount time.

5.3 Results for the dispersion and densest subgraph instances

Next, we apply our algorithm to instances of the dispersion and densest subgraph problems. The results are reported in Table 2. Moreover, we report the total time spent in the cutting-plane algorithm (3rd column), the total CPU time including both the cutting-plane and branch-and-bound phases (4th column), the upper bound gap at the root node (5th column), the lower bound gap from the heuristic (6th column), the number of instances out of 10 that were solved in the cutting-plane phase (7th column). Note that we set a time limit of 3 hours for all the instances, and the number of instances out of ten that were solved by the overall cut-and-branch algorithm are reported in the 8th column. For instances that could be solved within the time limit, we calculated the upper bound and lower bound gaps with respect to the best upper bound obtained by the MIP solver.

Δ(%)	n	Cutting Time (s)	UB Gap (%)	LB Gap (%)	BnB Nodes	Reduction (%)	Total Time (s)
	50	1.91	0.39	0.11	22	28.35	3.68
	100	42.04	0.22	0.08	33	26.66	58.24
	150	491.03	0.03	0.07	93	15.05	779.86
	200	3368.64	0.14	0.02	518	9.72	5338.38
	250	1507.48	0.04	0.01	210	12.34	2388.08
	300	3675.79	0.03	0.06	359	3.03	5342.42
	350	6454.23	0.03	0.05	394	15.97	10890.19
	400	6407.30	0.01	0.01	27	14.77	8054.03
25	450	6709.94	0.01	0.00	266	6.91	8509.11
	500	6397.59	0.12	0.00	51	11.53	10590.97
	550	9629.05	0.01	0.00	40	17.09	11206.73
	600	7109.43	0.00	0.00	0	17.77	10664.14
	650	6646.07	0.00	0.00	458	7.10	9969.10
	700	6366.09	0.00	0.00	0	19.89	9549.14
	750	6314.09	0.01	0.01	77	0.02	12454.92
	800	3701.10	0.02	0.05	41	0.01	5551.64
	50	0.72	0.28	0.29	15	32.01	1.92
	100	11.58	0.16	0.02	23	22.76	26.42
	150	251.99	0.10	0.01	85	6.21	398.33
	200	872.90	0.04	0.01	132	6.46	1576.72
	250	1683.50	0.03	0.02	51	11.73	2482.08
	300	6493.80	0.04	0.00	1402	6.21	9435.95
	350	12552.06	0.01	0.00	109	6.40	15947.56
	400	7051.79	0.01	0.00	1270	10.05	9143.78
50	450	8803.78	0.00	0.00	112	14.18	10560.85
	500	8072.94	0.00	0.00	5	14.93	12109.41
	550	5892.19	0.05	0.02	20	4.94	8307.77
	600	10101.32	0.00	0.00	25	1.76	11847.15
	650	10396.68	0.00	0.00	0	60.03	10396.68
	700	9768.79	0.00	0.00	50	3.51	11215.08
	750	6289.28	0.00	0.00	261	0.03	9433.92
	800	8585.88	0.01	0.00	51	0.02	10313.07
	50	0.37	0.23	0.00	3	30.31	0.59
	100	5.06	0.06	0.00	0	26.75	6.69
	150	136.31	0.21	0.01	66	1.93	247.49
	200	207.91	0.13	0.01	342	0.43	866.09
	250	567.83	0.08	0.00	65	25.48	1232.04
	300	1857.38	0.02	0.00	61	23.02	2614.41
	350	2043.55	0.07	0.00	595	19.54	3907.94
	400	4378.30	0.02	0.00	0	63.34	5004.30
75	450	4910.31	0.02	0.00	1036	13.20	7194.18
	500	4834.58	0.01	0.00	155	2.33	7394.69
	550	8205.37	0.04	0.00	303	28.68	14391.89
	600	9051.54	0.02	0.00	35	49.75	12561.62
	650	6689.55	0.03	0.00	1	72.78	10034.33
	700	8898.24	0.03	0.00	190	67.28	11656.54
	750	9136.86	0.03	0.01	22	80.83	11792.41
	800	8697.62	0.00	0.00	30	43.61	13046.44
	50	0.86	0.06	0.00	0	54.32	1.26
	100	19.11	0.10	0.01	0	36.37	28.86
	150	63.70	0.03	0.00	12	19.92	81.27
	200	92.01	0.04	0.00	93	23.10	237.20
	250	449.76	0.13	0.00	85	12.27	832.60
	300	1358.03	0.00	0.00	1	31.14	1465.88
	350	2625.03	0.03	0.00	74	19.95	4105.06
	400	2511.12	0.05	0.00	176	18.34	3612.41
100	450	5839.82	0.00	0.00	0	12.04	7441.90
	500	5947.79	0.01	0.00	16	13.45	8273.26
	550	5135.90	0.01	0.00	107	44.77	6408.54
	600	7346.82	0.00	0.00	72	44.27	9037.06
	650	8642.65	0.01	0.00	42	38.81	11110.36
	700	6944.93	0.09	0.00	589	53.73	10417.39
	750	7259.71	0.07	0.00	397	60.64	9402.97
	800	7852.13	0.15	0.00	350	78.52	9972.48

Table 1: Results for the standard QKP instances

The table reveals that of the 600 instances tested, 194 instances (nearly 1/3) were solved during the cutting-plane phase. Moreover, a total of 484 instances were solved within the time limit. In general, our algorithm struggles with the "plain" dispersion and densest subgraph instances, but it was able to solve all instances of the knapsack-like dispersion problem (KP-EXPO, KP-GEO, KP-WGEO and KP-RAN) in less than 1 hour.

5.4 Results for the planted clique instances

Finally, we consider the planted clique (PC) instances. Figure 1 shows the upper and lower bound gaps, the CPU time at the root node, and the CPU time at the end of the whole algorithm. It can be seen in Figure 1a that the upper bound gaps obtained by our algorithm are of great quality (almost always zero in fact). The lower bounds on the other hand are very poor. This is however to be expected, since the weakness of existing QPK heuristics for PC instances was already noted in [27].

As for the branch-and-bound phase, Figure 1b shows that it was very time consuming for these instances. This is almost certainly due to the poor lower bounds. Despite this, however, our algorithm is capable of finding the optimal solution to PC instances with up to 200 items within five hours. This can be seen as a milestone, since the state-of-the-art algorithm of Caprara *et al.* [5] failed to do so, according to [27].

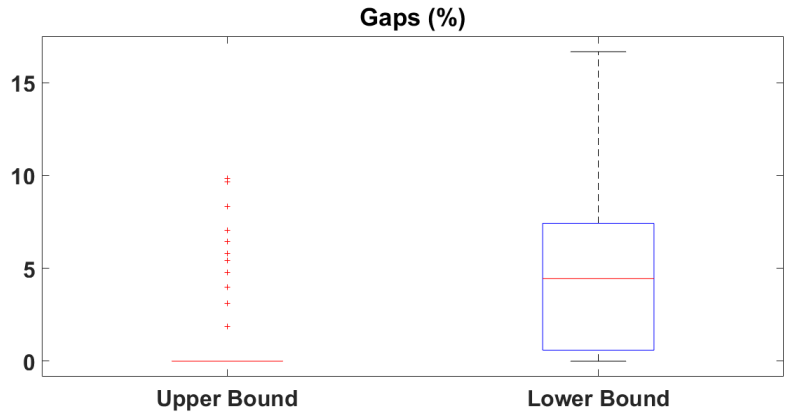
6 Conclusion

We have presented a new exact 'cut-and-branch' algorithm for the QKP. Our algorithm incorporates a cutting-plane phase, primal heuristics, rules for eliminating variables and constraints, and a branch-and-bound phase. Our computational results show that our algorithm is capable of solving much larger "standard" QKP instances than those solvable by the algorithm of Caprara *et al.* [5]. More precisely, our algorithm is capable of solving standard QKP instances with up to 800 items within a five hour time limit. Additionally, our algorithm performs fairly well on both the dispersion problem and the densest sub-graph instances of Pisinger *et al.* [23]. Indeed, almost a third of these instances are solved within the cutting plane phase, while a large majority is solved within a 3 hour time limit. For the much more difficult planted-clique instances, our proposed algorithm is able to find the optimal solution for instances with up to 200 items within five hours. Moreover, for these instances, the upper bounds found by the cutting-plane phase are almost always optimal.

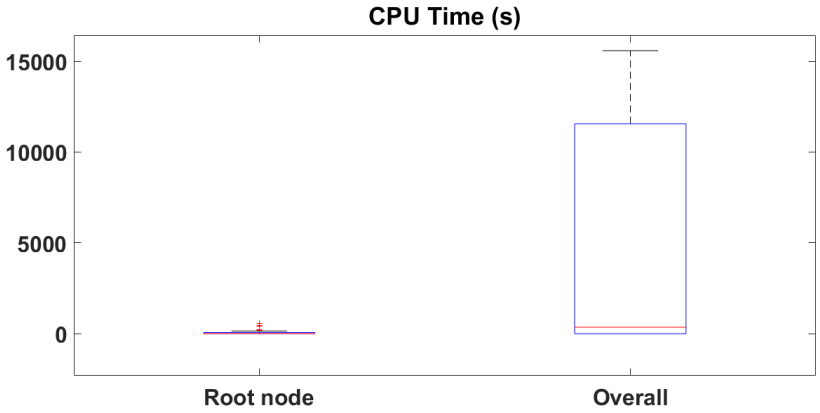
For future research, one could of course derive new inequalities and/or separation algorithms (see for example [9, 10]) and implement a full branch-and-cut algorithm. It would also be very worthwhile trying to use an "aggressive reduction" scheme similar to that of Pisinger *et al.* [23] as a "pre-

Instance Type	n	Cutting Time (s)	Total Time (s)	UB Gap (%)	LB Gap (%)	# Cutting Solved	Total Solved
DSUB25	25	0.07	0.09	0.00	1.07	8	10
	50	0.99	2.18	0.01	0.09	7	10
	100	451.94	11962.83	1.22	9.96	0	7
	200	7948.49	8525.80	0.00	8.54	3	3
	400	2973.87	2973.87	0.00	0.00	1	1
DSUB50	25	0.09	2.14	0.23	0.56	6	10
	50	0.97	1.39	0.00	0.86	6	10
	100	925.75	7158.64	1.12	7.94	3	7
	200	4366.20	7871.98	0.00	9.70	5	6
	400	14385.08	14719.58	0.09	0.19	1	3
DSUB75	25	0.07	0.64	0.50	1.09	8	10
	50	0.51	0.87	0.00	0.34	7	10
	100	148.27	4353.53	0.77	3.67	5	6
	200	7755.13	13758.79	0.00	4.93	2	4
	400	10122.23	13685.08	0.02	1.59	2	4
DSUB90	25	0.04	0.04	0.00	0.00	10	10
	50	1.17	2.08	0.01	0.15	8	10
	100	49.92	2852.10	0.77	2.70	5	8
	200	4702.18	6307.91	0.00	1.89	6	9
	400	9157.49	9157.49	0.00	0.00	5	5
EXPO	25	0.10	1.07	0.20	0.17	7	10
	50	3.72	67.72	1.61	0.43	1	10
	100	357.44	10539.74	3.91	3.76	1	4
	200	1391.47	7769.38	1.34	8.45	2	5
	400	4083.09	4083.09	0.00	0.00	1	1
GEO	25	0.07	3.25	1.78	0.14	6	10
	50	0.66	12.68	5.36	0.00	5	10
	100	69.27	3159.24	1.82	0.42	5	8
	200	499.12	2499.29	0.24	2.22	7	9
	400	3166.45	6668.17	0.43	5.48	3	3
WGEO	25	0.08	1.63	0.28	0.00	7	10
	50	0.73	2.67	0.13	0.02	5	10
	100	2.66	15.85	0.07	0.00	4	8
	200	94.74	114.61	0.02	0.00	4	9
	400	2808.82	2935.16	0.00	0.00	3	7
RAN	25	0.11	1.94	0.85	0.50	6	10
	50	5.47	345.49	3.92	0.42	1	10
	100	343.84	12160.68	2.82	8.21	0	2
	200	5737.27	12757.59	0.34	4.88	2	3
	400	12949.48	13431.56	0.38	13.92	0	1
KP-EXPO	25	0.13	1.50	0.46	0.00	8	10
	50	1.10	18.35	0.45	0.00	0	10
	100	3.61	96.43	0.14	0.00	0	10
	200	474.20	2202.49	0.06	0.00	0	10
	400	2382.81	4361.97	0.04	0.00	0	10
KP-GEO	25	0.07	2.18	0.44	0.00	4	10
	50	0.76	11.80	0.39	0.00	2	10
	100	4.23	79.91	0.20	0.00	0	10
	200	35.24	625.64	0.05	0.00	0	10
	400	1205.51	3841.88	0.02	0.00	0	10
KP-WGEO	25	0.08	7.57	0.71	0.00	2	10
	50	0.44	5.34	0.58	0.00	0	10
	100	3.59	69.56	0.17	0.00	0	10
	200	35.93	237.64	0.05	0.00	0	10
	400	1097.65	1555.66	0.01	0.00	2	10
KP-RAN	25	0.06	2.69	0.15	0.00	6	10
	50	3.07	14.94	0.33	0.00	0	10
	100	33.42	260.39	0.23	0.00	0	10
	200	1248.97	1556.26	0.05	0.00	0	10
	400	913.94	1056.82	0.01	0.00	2	10

Table 2: Results for the dispersion problem and densest sub-graph problem instances



(a) Upper bound and lower bound gaps



(b) CPU times

Figure 1: Summary results for planted clique instances

processor", and then running our algorithm on the reduced problem. We believe that this approach would enable one to solve even larger instances.

References

[1] E. Balas (1975) Facets of the knapsack polytope. *Math. Prog.*, 8, 146–164.

[2] A. Billionnet & F. Calmels (1996) Linear programming for the 0–1 quadratic knapsack problem. *Eur. J. Oper. Res.*, 92, 310–325.

[3] A. Billionnet & E. Soutif (2004) An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem. *Eur. J. Oper. Res.*, 157, 565–575.

[4] A. Billionnet & E. Soutif (2004) Using a mixed-integer programming tool for solving the 0-1 quadratic knapsack problem. *INFORMS J. Comp.*, 16, 188–197.

[5] A. Caprara, D. Pisinger & P. Toth (1998) Exact solution of the quadratic knapsack problem. *INFORMS J. Comput.*, 11, 125–137.

[6] P. Chaillou, P. Hansen & Y. Mahieu (1986) Best network flow bound for the quadratic knapsack problem. In B. Simeone (ed.) *Combinatorial Optimization*, pp. 225–235. Lecture Notes in Mathematics, vol. 1403. Berlin: Springer.

[7] H. Crowder, E. Johnson & M. Padberg (1983) Solving large-scale 0-1 linear programming problems. *Oper. Res.*, 31, 803–834.

[8] M.M. Deza & M. Laurent (1997) *Geometry of Cuts and Metrics*. Berlin: Springer-Verlag.

[9] F. Djeumou Fomeni (2017), A new family of facet defining inequalities for the maximum edge-weighted clique problem. *Optim. Lett.*, 11(1), 47-54

[10] F. Djeumou Fomeni, K. Kaparis & A. N. Letchford (2015) Cutting planes for first-level RLT relaxations of mixed 0-1 programs. *Math. Program.*, 151, 639–658.

[11] F. Djeumou Fomeni & A.N. Letchford (2014) A dynamic programming heuristic for the quadratic knapsack problem. *INFORMS J. Comput.*, 26, 173–183.

[12] A. Faye & O. Boyer (2003) Construction of facets of the 0-1 quadratic knapsack polytope (in French). *RAIRO Oper. Res.*, 37, 249–271.

[13] G. Gallo, P.L. Hammer & B. Simeone (1980) Quadratic knapsack problems. *Math. Program. Study*, 12, 132–149.

[14] F. Glover & E. Woolsey (1974) Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Oper. Res.*, 22, 180–182.

[15] Z. Gu, G.L. Nemhauser & M.W.P. Savelsbergh (2000) Sequence-independent lifting in mixed integer programming. *J. Comb. Optim.*, 4, 109–129.

[16] C. Helmberg, F. Rendl & R. Weismantel (2000) A semidefinite programming approach to the quadratic knapsack problem. *J. Comb. Optim.*, 4, 197–215.

[17] E.L. Johnson, A. Mehrotra & G.L. Nemhauser (1993) Min-cut clustering. *Math. Program.*, 62, 133–151.

[18] K. Kaparis & A.N. Letchford (2010) Separation algorithms for 0-1 knapsack polytopes. *Math. Program.*, 124, 69–91.

[19] H. Kellerer, U. Pferschy & D. Pisinger (2004) *Knapsack Problems.* Berlin: Springer.

[20] A.N. Letchford & G. Souli (2019) On lifted cover inequalities: a new lifting procedure with unusual properties. *Oper. Res. Lett.*, 47, 83–87.

[21] M.W. Padberg (1989) The Boolean quadric polytope: some characteristics, facets and relatives. *Math. Program.*, 45, 139–172.

[22] D. Pisinger (2007) The quadratic knapsack problem — a survey. *Discr. Appl. Math.*, 155, 623–648.

[23] D. Pisinger, A.B. Rasmussen & R. Sandvik (2007) Solution of large quadratic knapsack problems through aggressive reduction. *INFORMS J. Comput.*, 19, 280–290.

[24] D.J. Rader (1997) Valid inequalities and facets of the quadratic 0-1 knapsack polytope. *RUTCOR Research Report 16–97*, Rutgers University.

[25] D.J. Rader (1997) Lifting results for the quadratic 0-1 knapsack polytope. *RUTCOR Research Report 17–97*, Rutgers University.

[26] H.D. Sherali & W.P. Adams (1990) A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM J. Discr. Math.*, 3, 411–430.

[27] J. Schauer (2016) Asymptotic behavior of the quadratic knapsack problem. *Eur. J. Oper. Res.*, 255, 357–363.

[28] R. Weismantel (1997) On the 0-1 knapsack polytope. *Math. Program.*, 77, 49–68.

[29] L.A. Wolsey (1975) Faces for a linear inequality in 0-1 variables. *Math. Program.*, 8, 165–178.

[30] L.A. Wolsey (1998) *Integer Programming.* New York: Wiley.

Appendix 1: Effect of Separation Order

After coding our cutting-plane algorithm, we were interested in exploring the effect that the *separation order* (i.e., the sequence in which the cutting planes are separated and added) had on performance. To do this, we let the cutting-plane algorithm run to completion, without terminating even if the condition $L = \lfloor U \rfloor$ held. The results obtained with the four most promising

separation orders, on the "small" standard instances, are presented in Table 3 and Table 4. The values in the tables are the averages over 10 instances. For each ordering, we present the percentage gap between the upper bound and the optimum, the time needed to compute that bound, in seconds, and the percentage of y variables eliminated from the problem.

It can be seen from the results of these tables that our cutting-plane algorithm can consistently produce upper bounds within 1% of optimality, in reasonable computing times. The number of y variables reduced is also quite remarkable. It is hard to say anything conclusive about the effect of the separation order on the quality of the results. For the harder QKP instances, we settled on the fourth separation order, as mentioned in Subsection 4.2.

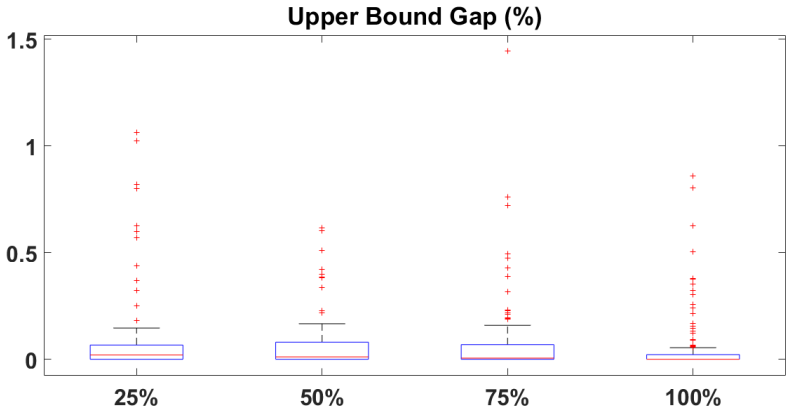
		RLT+CT+RLT-EC+T			RLT+RLT-EC+CT+T		
Δ	n	Gap (%)	Time (s)	Red. (%)	Gap (%)	Time (s)	Red. (%)
25	10	1.67	0.01	60.45	1.67	0.00	59.56
	20	0.29	0.04	60.74	0.16	0.14	60.95
	30	0.80	0.28	49.47	0.78	1.14	47.72
	40	1.90	0.53	31.74	1.91	0.53	31.18
	50	0.99	2.62	44.96	0.99	4.98	48.10
	60	0.86	5.70	36.32	0.54	4.29	37.93
	70	0.23	5.27	29.33	0.20	5.07	34.94
	80	0.26	32.41	32.13	0.26	39.06	31.53
	90	1.63	422.52	22.69	1.65	132.19	22.91
	100	0.15	46.87	32.15	0.15	58.93	36.55
50	10	0.00	0.01	85.78	0.02	0.01	65.78
	20	1.22	0.05	52.42	1.07	0.07	58.95
	30	0.23	0.66	44.78	0.22	0.84	44.41
	40	0.65	1.61	28.87	0.65	1.23	32.33
	50	0.11	5.27	46.14	0.07	4.31	53.26
	60	0.21	11.82	53.21	0.23	5.95	46.85
	70	0.28	27.90	23.79	0.20	22.12	35.68
	80	0.08	23.19	32.44	0.09	17.70	43.60
	90	0.14	50.45	32.81	0.13	50.83	26.64
	100	0.18	44.51	18.95	0.18	40.76	19.21
75	10	0.00	0.01	47.11	0.00	0.00	44.00
	20	1.26	0.05	47.37	1.27	0.05	46.32
	30	0.18	0.82	73.75	0.17	0.53	73.75
	40	0.39	1.83	38.33	0.38	0.99	38.90
	50	0.14	2.49	41.88	0.11	2.52	52.56
	60	0.38	11.58	28.67	0.38	11.59	33.42
	70	0.75	2.58	14.81	0.75	3.79	15.00
	80	0.25	37.29	19.20	0.24	9.75	23.73
	90	0.13	37.65	23.47	0.14	25.41	21.84
	100	0.10	11.96	26.09	0.10	10.41	29.14
100	10	0.00	0.01	74.22	0.00	0.01	58.67
	20	2.07	0.08	56.53	1.99	0.06	71.26
	30	2.35	0.24	62.81	0.59	0.30	76.78
	40	0.08	1.02	62.62	0.08	1.06	67.64
	50	0.04	1.57	40.85	0.02	2.29	51.74
	60	0.10	5.00	29.30	0.07	7.69	52.53
	70	0.15	17.55	28.92	0.14	13.88	23.82
	80	0.06	2.89	35.40	0.05	8.18	42.77
	90	0.19	5.89	11.10	0.19	5.22	19.88
	100	0.02	64.39	45.16	0.02	63.45	46.67

Table 3: Cutting-plane results (part I)

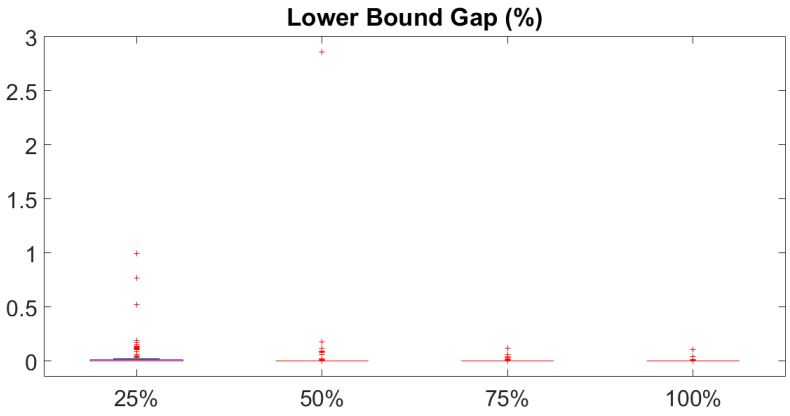
		RLT+RLT-EC+T+CT			RLT+T+RLT-EC+CT		
Δ	n	Gap (%)	Time (s)	Red. (%)	Gap (%)	Time (s)	Red. (%)
25	10	1.81	0.01	59.56	1.67	0.00	60.44
	20	0.29	0.09	58.74	0.32	0.05	58.84
	30	0.78	0.44	47.36	0.78	0.24	49.43
	40	2.12	0.36	30.20	1.67	0.67	30.87
	50	1.00	2.07	42.94	1.00	2.92	49.86
	60	0.73	3.26	32.29	0.69	2.83	31.99
	70	0.23	12.41	28.48	0.22	23.51	29.44
	80	0.27	24.77	31.37	0.27	24.38	32.38
	90	1.66	116.52	21.07	0.15	68.55	22.75
	100	0.15	42.29	36.92	0.15	42.97	37.34
50	10	0.00	0.00	66.22	0.00	0.01	70.67
	20	1.27	0.03	50.63	1.13	0.04	51.58
	30	0.24	0.29	42.25	0.23	0.23	45.24
	40	0.70	1.00	30.08	0.77	0.85	20.13
	50	0.08	3.06	52.16	0.10	2.04	53.41
	60	0.25	5.50	44.79	0.24	4.60	37.95
	70	0.23	24.53	33.87	0.27	12.04	32.16
	80	0.09	16.22	42.48	0.08	15.73	39.67
	90	0.14	23.05	25.48	0.14	18.04	28.54
	100	0.16	62.24	24.93	0.18	25.23	19.00
75	10	0.00	0.00	44.00	0.00	0.00	47.55
	20	1.34	0.04	46.53	1.38	0.04	44.42
	30	0.21	0.28	69.88	0.21	0.21	73.06
	40	0.39	0.99	38.82	0.61	0.48	31.05
	50	0.13	2.30	50.17	0.19	1.20	48.33
	60	0.38	8.71	33.71	0.38	6.57	25.90
	70	0.77	3.07	15.62	0.75	2.08	14.78
	80	0.36	5.07	19.10	0.26	43.79	19.42
	90	0.16	17.39	13.21	0.15	19.56	15.20
	100	0.10	10.73	28.08	0.10	10.91	27.49
100	10	0.00	0.01	58.67	2.07	0.01	65.33
	20	2.02	0.04	69.37	2.05	0.05	68.84
	30	0.59	0.23	74.21	0.60	0.27	75.54
	40	0.05	0.77	70.21	0.05	0.57	70.62
	50	0.02	1.73	52.12	0.02	1.72	52.16
	60	0.03	5.63	66.29	0.06	10.45	61.36
	70	0.14	6.76	23.47	0.14	6.28	32.30
	80	0.05	6.78	51.59	0.05	8.96	51.92
	90	0.19	4.60	19.88	0.19	3.93	19.65
	100	0.02	62.91	44.78	0.02	61.66	23.94

Table 4: Cutting-plane results (part II)

Appendix 2: More Results for Large Standard QKP Instances

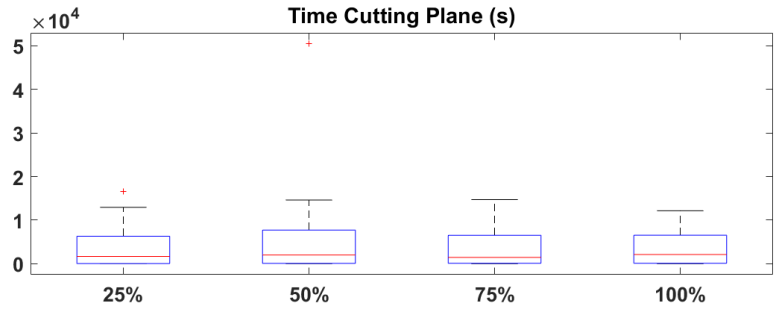


(a) Upper bound gaps

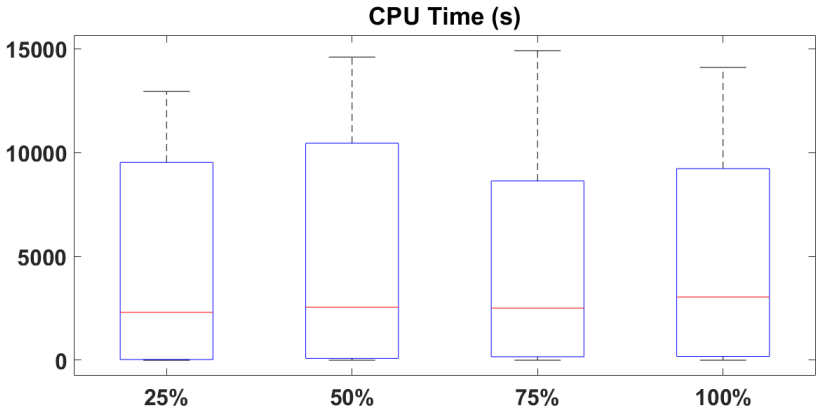


(b) Lower bound gaps

Figure 2: Upper and lower bound gaps for the standard QKP instances



(a) Time in the Cutting plane algorithm



(b) Total time

Figure 3: Summary of the CPU time for the standard QKP instances

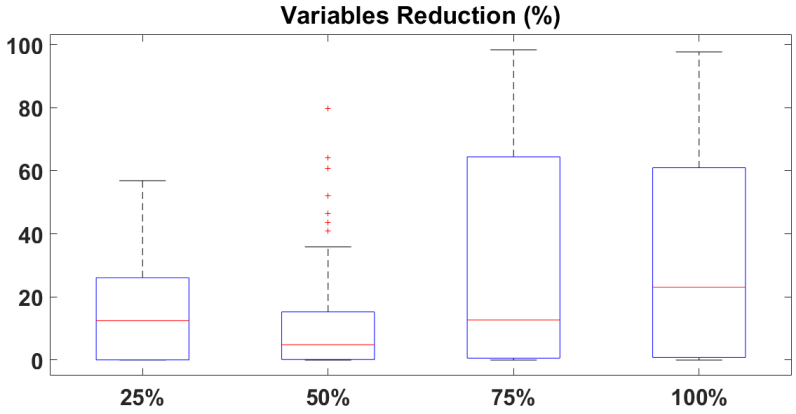


Figure 4: Summary of the percentage of y variables reduced